Journal of Computational Mathematics
Vol.37, No.6, 2019, 889–915.

http://www.global-sci.org/jcm
doi:10.4208/jcm.1910-m2019-0034

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

Haoyang Liu and Zaiwen Wen
Beijing International Center for Mathematical Research, Peking University, Beijing 100871, China
Email: liuhaoyang@pku.edu.cn, wenzw@pku.edu.cn
Chao Yang
Computational Research Division, Lawrence Berkeley National Laboratory, Berkeley, CA, USA
Email: cyang@lbl.gov
Yin Zhang
Department of Computational and Applied Mathematics, Rice University, Houston, USA
Email: yzhang@rice.edu

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. 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 C on a representative set of test problems to evaluate the performance of two variants of our algorithm. 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 that of several state-of-the-art eigensolvers.

Mathematics subject classification: 65F15, 90C06.
Key words: Extreme eigenpairs, Augmented Rayleigh-Ritz projection.

1. Introduction

In this paper, we consider to compute $k \ll n$ eigenpairs corresponding to k largest or smallest eigenvalues of a given large-scale 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 a 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 $i \in [1, n]$,

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

* Received February 10, 2019 / Revised version received May 15, 2019 / Accepted October 9, 2019 / Published online November 18, 2019 /

where 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 $\Lambda = \Lambda_n$.

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 (SU) 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. At present, the predominant methodology for subspace updating is still Krylov subspace methods, as represented by Lanczos type methods [12, 15] 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 multiplied 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 [22], 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. Once X is obtained and orthonormalized, the projection step aims to extract from X a set of approximate eigenpairs (see more details in Section 2) that are optimal in a sense. For the projection step often the method of choice is the Rayleigh-Ritz (RR) procedure. More complete treatments of iterative algorithms for computing subsets of eigenpairs can be found, for example, in the books [1, 3, 19, 25, 30].

The purpose of this work is to construct 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 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 (the sparse matrix A multiplying a vector): (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. [24, 34] 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. The augmented Rayleigh-Ritz (ARR) procedure proposed in [33] is carefully implemented and extensively evaluated in numerical experiments. Our ARR 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). It 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 stopping 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. The polynomial accelerators used by us are given in Section 4. A detailed pseudocode for our algorithm is outlined in Section 5. Numerical results are presented in Section 6. Finally, we conclude the paper in Section 7.

2. Overview of Iterative Algorithms for Eigenpair Computation

Algorithms for the 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 2.1 below, which is also denoted by a map $(Y, \Sigma) = \mathrm{RR}(A, Z)$ where the output (Y, Σ) is a Ritz pair block.

Algorithm 2.1: Rayleigh-Ritz procedure: $(Y, \Sigma) = \mathrm{RR}(A, Z)$

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

It is known (see [19], 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 subspace of order k is $\text{span}\{v, Av, A^2v, \ldots, A^{k-1}v\}$. Typical Krylov subspace methods include Arnoldi algorithm for general matrices (e.g., [14,15]) and Lanczos algorithm for symmetric (or Hermitian) matrices (e.g., [13, 27]). In either algorithm, orthonormal bases for Krylov subspaces are generated through a Gram-Schmidt type process. Some variants of Jacobi-Davidson methods (e.g., [2,28]) 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 of 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 [22, 23, 29, 31], 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 2.2 below, following the description in [30].

Algorithm 2.2: Simple Subspace Iteration (SSI)

```
1 Initialize orthonormal matrix U ∈ R^{n×m} with m = k + q ≥ k.
2 while the number of converged eigenpairs is less than k, do
3   while convergence is not expected, do
4     while the columns of U are sufficiently independent, do
5         Compute U = AU.
6         Normalize each column of U.
7     Orthogonalize the columns of U.
8   Perform an RR step using U.
9   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 reduces average memory access time by increasing arithmetic intensity and allowing better cache utilization. Hence, highly parallelizable computation is possible 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 pointed out in [30], "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:

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

This formulation can be easily extended to solving the generalized eigenvalue problem

$$\max_{X\in\mathbb{R}^{n\times k}} \mathrm{tr}(X^TAX),\ \ \mathbf{s.t.}\ X^TBX = I, \tag{2.2}$$

where B is a symmetric positive definite matrix. When maximization is changed to minimization, one computes an eigenspace associated with k smallest eigenvalues. The algorithm TraceMin [26] solves the trace minimization problem using a Newton type method.

Some block algorithms have been developed based on solving (2.1), including the locally optimal block preconditioned conjugate gradient method (LOBPCG) [10] and more recently the limited memory block Krylov subspace optimization method (LMSVD) [16]. At each iteration, these methods solve a subspace trace maximization problem of the form

$$Y = \arg\max_{X\in\mathbb{R}^{n\times k}} \Big\{\mathrm{tr}(X^TAX) : X^{\mathrm{T}}X = I,\ X\in\mathcal{S}\Big\}, \tag{2.3}$$

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

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

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.,

$$\mathcal{S} = \mathrm{span}\left\{X^{(i)}, X^{(i-1)}, ..., X^{(i-p)}\right\}. \tag{2.5}$$

In general, the subspace $\mathcal{S}$ should be constructed such that the cost of solving (2.3) 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, [6, 24, 30, 34]). 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

$$\rho(A) = Q\rho(\Lambda)Q^T = \sum_{i=1}^{n} \rho(\lambda_i) q_i q_i^T, \tag{2.6}$$

where $\rho(\Lambda) = \mathrm{diag}(\rho(\lambda_1), \rho(\lambda_2), \ldots, \rho(\lambda_n))$. 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 (2.6) 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:

$$\rho_{d+1}(t) = 2t\rho_d(t) - \rho_{d-1}(t), \quad d \geq 1, \tag{2.7}$$

where $\rho_0(t) = 1$ and $\rho_1(t) = t$. Some recent works that use Chebyshev polynomials include [6, 34], for example.

2.5. A Gauss-Newton Algorithm

A Gauss-Newton (GN) algorithm is recently proposed in [17] 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.

Algorithm 2.3: A GN Algorithm: $X = \mathrm{GN}(A, X)$

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

It is shown in [17] 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 [17] 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 2.3, named simply as GN. For more theoretical and numerical results on this GN algorithm, we refer readers to [17].

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 (2.3) 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 [10] and LMSVD [16] augment $X^{(i)}$ by additional blocks (see (2.4) and (2.5), respectively). Not surprisingly, such augmentations are the main reason why algorithms like LOBPCG 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

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

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 [33] for more details). 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 (2.3), restricted in the subspace $\mathcal{S}$ in (3.1), can be computed via the RR procedure, i.e., Algorithm 2.1. We formalize our augmented RR procedure as Algorithm 3.1, which will often be referred to simply as ARR. The analysis of ARR in [33, Corollary 4.6] shows that the convergence rate of SSI is improved from $|\rho(\lambda_{k+1})/\rho(\lambda_k)|$ for RR ($p = 0$) to $|\rho(\lambda_{(p+1)k+1})/\rho(\lambda_k)|$ for ARR ($p > 0$). Therefore, a significant improvement is possible with a suitably chosen polynomial $\rho(\cdot)$ such that $|\rho(\lambda_{(p+1)k+1})| \ll |\rho(\lambda_{k+1})|$.

Algorithm 3.1: ARR: $(Y, \Sigma) = \textsc{Arr}(A, X, p)$

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

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

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 intentionally 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 2.3, 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 3.2. The two variants, corresponding to "inner solvers" MPM and GN, will be named ARRABIT-MPM and ARRABIT-GN, or simply MPM and GN.

Algorithm 3.2: Algorithm ARRABIT (abstract version)

```
Input A ∈ R^{n×n}, k, p and ρ(t). Initialize X ∈ R^{n×m} with m = k + q ≥ k.
while not "converged", do
  while "inner criteria" are not met, do
    if MPM is the inner solver, then
        X = ρ(A)X, then normalize columns individually.
    if GN is the inner solver, then
        X = GN(ρ(A), X), as is given by Algorithm 2.3.
  ARR projection: (X, Σ) = ARR(A, X, p), as in Algorithm 3.1.
  Possibly adjust p, the degree of ρ(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 3.2. Let $\mathcal{U}$ and $\mathcal{U}^\perp$ be the subspace spanned by the wanted and unwanted eigenvectors of A, respectively. With proper polynomial filters, the projection of X onto $\mathcal{U}^\perp$ shrinks faster than that onto $\mathcal{U}$ as MPM or GN is applied. 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 since it most likely implies that we have succeeded in eliminating the unwanted eigenspace. In this case an RR step is needed immediately to extract the approximate eigenvalues and eigenvectors, otherwise a part of the wanted eigenspace will probably also be sacrificed. More details on the proposed "inner criteria" will be given in Algorithm 5.1 in Section 5. The "inner criteria" and the use of ARR together can generally improve the qualities of both SU and RR steps, hence reducing the number of outer iterations (or ARR calls) required to reach a relatively high accuracy. Of course, there are extra computational costs in SU (and ARR) steps, but they are predominantly SpMVs.

4. 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

$$t_j = -\cos(j\pi/N), \quad 0 \le j \le N, \tag{4.1}$$

where $N \ge 1$ is an integer. Obviously, this set of $N+1$ points are in the interval $[-1, 1]$ inclusive of the two end-points. 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

$$f_d(t) = (f_1(t))^d \quad \text{where} \quad f_1(t) = \max(0, t)^{10}, \tag{4.2}$$

and d is a positive integer. Obviously, $f_d(t) \equiv 0$ for $t \leq 0$ 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 (4.2) are many times differentiable so that their Chebyshev interpolants converge relatively fast, see [18]. 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,

$$\psi_d(t) = \gamma_1 t^d + \gamma_2 t^{d-1} + \ldots + \gamma_d t + \gamma_{d+1}. \tag{4.3}$$

Suppose that we want to dampen the eigenvalues in an interval $[a, b]$, where $a \leq \lambda_n$ 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,

$$\rho_d(t) = \psi_d\left(\frac{2t - a - b}{b - a}\right). \tag{4.4}$$

Let $\Gamma_d = (\gamma_1, \gamma_2, \cdots, \gamma_{d+1})$ denote the coefficients of the polynomial $\psi_d(t)$ in (4.3). The corresponding matrix operation $Y = \rho_d(A)X$ can be implemented by Algorithm 4.1 below.

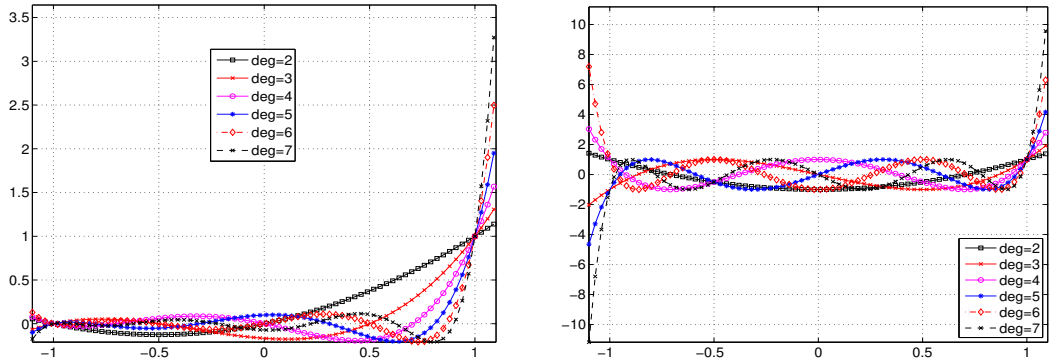
Algorithm 4.1: Polynomial function: $Y = \text{POLY}(A, X, a, b, \Gamma_d)$

1 Compute $c_0 = \frac{a+b}{a-b}$ and $c_1 = \frac{2}{b-a}$. Set $Y = \gamma_1 X$.
2 **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 interpolants of degrees 2 to 7 and the Chebyshev polynomials of degrees 2 to 7 side by side in Fig. 4.1. 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 interpolants 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 interpolants 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. 4.1. Illustration of polynomial functions.

5. 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. We first present a pseudo-code of a detailed version of the ARRABIT algorithm in Algorithm 5.1, then explain each feature in detail. As one can see, the ARRABIT algorithm uses A only in matrix multiplications. The main memory requirements are three $n \times (k+q)$ matrices for X, Y and Z in the GN method and two $n \times ((k+q)(p+1))$ matrices in the ARR step. These matrices usually become smaller as the iteration proceeds.

Algorithm 5.1: Algorithm ARRABIT (detailed version)

Input $A \in \mathbb{R}^{n\times n}$, integer $k \in (0, n)$ and tolerance tol > 0.
Choose d and $d_{\max}$, the initial and maximum polynomial degrees. 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, Σ_c) to be empty sets for deflation purposes.
Initialize a matrix $X \in \mathbb{R}^{n\times(k+q)}$ and 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 */

17 Compute $X = Z - X(Y^{\mathrm{T}}Z - I)/2$.
18 Compute $X = X - Q_c(Q_c^T X)$ if Q_c is not empty.
19 Set rcp = rc and compute rc = rcond($X^T X$).
20 **if** *the inner stop rule* (5.4) *is met*, **then** break. `/* end inner loop */`
21 Compute $Y = [X, AX, \ldots, A^p X]$. `/* augmentation */`
22 $Y = Y - Q_c(Q_c^T Y)$ if Q_c is not empty. `/* projection */`
23 Perform ARR step: $(X, \Sigma) = \mathrm{RR}(A, Y)$. `/* ARR */`
24 Extract $k + q$ leading Ritz pairs (x_i, μ_i) from (Q_c, Σ_c) and (X, Σ).
25 Overwrite (X, Σ) by the $k + q$ Ritz pairs. Compute residuals by (5.2).
26 **if** *the outer stop rule* (5.1) *is met for* tol, **then**
27 output the Ritz pairs (X, Σ) and exit. `/* output and exit */`
28 **if** *the outer stop rule* (5.1) *is met for* tol_t, **then** `/* continuation */`
29 Set $\mathrm{tol}_{t+1} = \max\left(10^{-2}\mathrm{tol}_t, \mathrm{tol}\right)$, $b = \mu_{k+q}$ and $t = t + 1$.
30 Collect converged Ritz pairs in (Q_c, Σ_c) that satisfy (5.5). `/* deflation */`
31 Overwrite (X, Σ) by the remaining not yet converged Ritz pairs.
32 **if** *rules in* (5.6) *are met*, **then** *set* $p = \min(p + 1, p_{\max})$.x `/* update p */`
33 Update the polynomial degree by rules (5.7)-(5.8). `/* update degree */`

Guard vectors (Line 3). 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 fewer 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} (Line 6 and 29). 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 can be computed by calling the ARPACK [15]). Alternatively, it can be estimated by the Gershgorin circle theorem or the Lanczos algorithm with a few iterations. 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 $X^T AX$ (which requires an RR projection). As the iterations progress, more accurate estimates of λ_{k+q} will become available after each later ARR projection. In general, ARRABIT can be slow if it is difficult to construct a suitable polynomial due to the poor spectral distribution.

Outer loop stop rule (Line 26). 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.,

$$\texttt{maxres} := \max_{i=1,\ldots,k}\{\texttt{res}_i\} \le \mathrm{tol}, \tag{5.1}$$

where

$$\texttt{res}_i := \frac{\|Ax_i - \mu_i x_i\|_2}{\max(1, |\mu_i|)}, \quad i = 1, \cdots, k. \tag{5.2}$$

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, $\texttt{maxres} < (1 + 9h/k)\text{tol}$ ($< 10 * \text{tol}$), where h is the number of Ritz pairs with residuals less than $0.1 \times$ 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 (Line 28). When a high accuracy (say, tol $\leq 10^{-8}$) is requested, we use a continuation procedure to compute Ritz-pairs satisfying a sequence of tolerances: $\text{tol}_1 > \text{tol}_2 > \cdots \geq \text{tol}$, 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

$$\text{tol}_{t+1} = \max(10^{-2}\,\text{tol}_t,\, \text{tol}), \tag{5.3}$$

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 (Line 20). 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 ARPACK 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 $\text{maxit}_1 = 10$ and $\text{maxit}_2 = 5$ Therefore, the maximum number of inner iterations allowed is $\text{maxit}_1 \times \text{maxit}_2 = 50$.

The inner loop stopping criteria are either

$$\texttt{rc} = \texttt{rcond}(X^T X) \leq \max(10 \times \text{tol}_t^2, 10^{-15}) \quad \text{or} \quad \texttt{rc}/\texttt{rcp} > 0.99, \tag{5.4}$$

where tol_t is the current tolerance (in a continuation sequence) and `rcp` is the previously computed $\text{rcond}(X^T X)$. The notation $\texttt{rcond}(X^T X)$ is the reciprocal of the 1-norm condition number of $X^T X$. We use the `dgecon` subroutine in LAPACK to estimate this condition number, which we find to be relatively cheap. The first condition in (5.4) 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 (Line 18, 22 and 30). 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 (5.2) for definition) satisfies

$$\texttt{res}_i \leq \max(10^{-14}, \text{tol}_t^2). \tag{5.5}$$

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 $Y = [X\ AX\ \cdots\ A^p X]$. 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 (Line 32). 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

$$\frac{\mu_{k+q}}{\mu_k} > 0.95 \quad \text{and} \quad \frac{\texttt{maxres}}{\texttt{maxresp}} > 0.1, \tag{5.6}$$

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 (Line 33). Under normal conditions, the higher degree is used in a polynomial accelerator, the fewer 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 (4.4). 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

$$\hat{d} = \min_{d \geq 3} \left\{ d \in \mathbb{Z} : \frac{\rho_d(\mu^*_{k+q})}{\rho_d(\mu^*_k)} < 0.9 \right\}, \tag{5.7}$$

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

$$d = \min(\hat{d}, d_{\max}) \tag{5.8}$$

where μ^*_k and μ^*_{k+q} are a pair of Ritz values corresponding to the iteration with the smallest residual "`maxres`" defined in (5.1) (therefore the most accurate so far). The value of 0.9 is of course adjustable.

6. Numerical Results

In this section, we evaluate the performance of ARRABIT on a set of sixteen sparse matrices and test the algorithm on a single computing node (2 processors) to determine how it performs in comparison to established solvers. We have implemented our ARRABIT algorithm, as is described by the pseudocode Algorithm 5.1, in C. 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 the sequential experiments : tol = 10^{-6} or tol = 10^{-12}. By our stopping 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.

6.1. 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[1] [15], which is behind the Matlab built-in iterative eigensolver EIGS, and will naturally serve as the benchmark solver since it is robust and has a long history in the eigenvalue computation field. Another is the Scalable Library for Eigenvalue Problem Computations [8, 21] (SLEPc, verrsion 3.5.4)[2] , which implements multiple methods based on Krylov subspace. ARPACK is written in Fortran and it can be accessed from C without much effort. SLEPc is written in C and can be called directly. In our experiments, all parameters in ARPACK and SLEPc are set to their default values, and each solver terminates with its own stopping rules using either tol $= 10^{-5}$ or tol $= 10^{-11}$. Besides, we choose the Krylov Schur solver [7] in SLEPc, which is the default option and also the fastest. The threaded version Intel Math Kernel Library (MKL) is used as the external dependency for BLAS/LAPACK/SpMV/block SPMV subroutines for all three solvers, which implies that we utilize the parallel codes as possible as we could. Consequently, no explicit code parallelization is performed outside Intel MKL.

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[3] [6], the Chebyshev-Davidson algorithm[4] [34], the Block Locally Optimal Preconditioned Eigenvalue Xolvers (BLOPEX [5]) [9] and the FEAST algorithm [20, 32]. The reason of not using FEAST is that it is a package designed to compute all eigenvalues (and their eigenvectors) in an interval and its performance is greatly affected by the quality of the estimation of this interval and the efficiency of solving the linear systems of equations involved in applying the rational filter. For the other solvers, our initial tests indicated that their overall performance could not measure up with that of the commercial-grade software packages ARPACK and SLEPc 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.

Our numerical experiments are performed 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 C standalone executable programs and submit them as batch jobs to Edison. The reported runtimes are wall-clock times.

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 [11], 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.

[1] See http://www.caam.rice.edu/software/ARPACK/
[2] Downloadable from http://slepc.upv.es/download
[3] See http://www-users.cs.umn.edu/ saad/software/filtlan
[4] See http://faculty.smu.edu/yzhou/code.htm
[5] Downloadable from http://code.google.com/p/blopex
[6] See http://www.nersc.gov/users/computational-systems/edison/
[7] See http://www.cise.ufl.edu/research/sparse/matrices

Table 6.1 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 hundreds place or to $k = 1000$ whichever is smaller.

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 for the sequential test 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 6.1: Information of Test Matrices.

matrix name	n	k	$nnz(A)$	density of A
Andrews	60000	600	760154	0.021%
C60	17576	176	407204	0.132%
cfd1	70656	707	1825580	0.037%
finance	74752	748	596992	0.011%
Ga10As10H30	113081	1000	6115633	0.048%
Ga3As3H12	61349	613	5970947	0.159%
shallow_water1s	81920	819	327680	0.005%
Si10H16	17077	171	875923	0.300%
Si5H12	19896	199	738598	0.187%
SiO	33401	334	1317655	0.118%
wathen100	30401	304	471601	0.051%
Ge87H76	112985	1000	7892195	0.062%
Ge99H100	112985	1000	8451395	0.066%
Si41Ge41H72	185639	1000	15011265	0.044%
Si87H76	240369	1000	10661631	0.018%
Ga41As41H72	268096	1000	18488476	0.026%

6.2. Comparison between RR and ARR

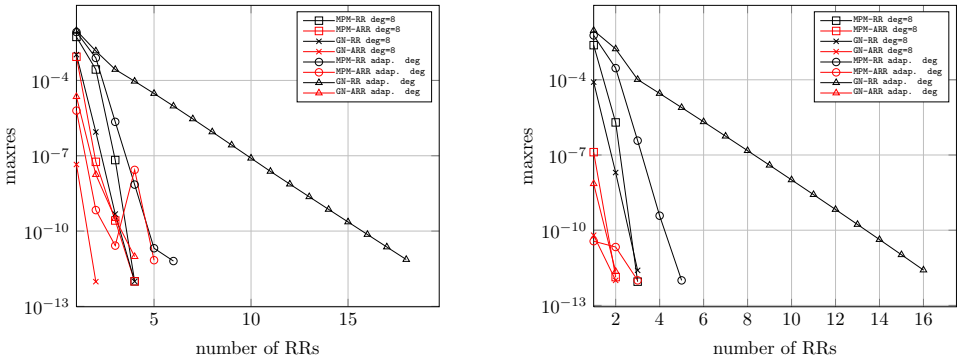
We first evaluate the performance difference between ARR and RR for both MPM and GN. Table 6.2 gives results for computing both k largest and smallest eigenpairs on the first six matrices in Table 6.1 to the accuracy of tol $= 10^{-12}$. We note that RR and ARR correspond to $p = 0$ and $p > 0$, respectively, in Algorithm 5.1. In order to differentiate the effect of changing p from that of changing the polynomial degree, we also test a variant of Algorithm 5.1 with a fixed polynomial degree at $d = 8$ (by skipping line 33). In Table 6.2, "`maxres`" denotes the maximum relative residual norm in (5.1), "time" is the wall-clock time 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 6.1 and 6.2 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 the 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 6.3.
- 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 6.1 that in several cases a single ARR is sufficient to reach the accuracy of tol=1e-6 (even of tol=1e-10 in one case).

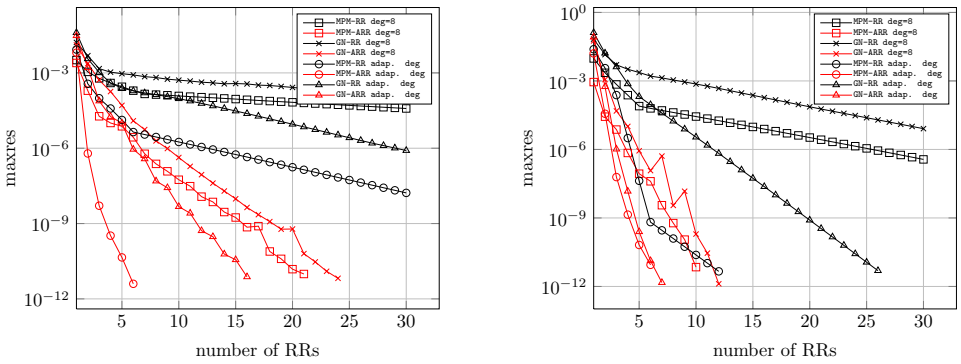
Table 6.2: Comparison results between RR and ARR with tol=1e-12.

	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 eigpairs by fix deg = 8																
Andrew.	9.2e-13	113	4	0/8	8.7e-13	47	3	1/8	7.1e-12	89	6	0/8	9.7e-13	53	2	1/8
C60	3.6e-12	17	14	0/8	4.1e-12	6	3	1/8	9.4e-12	18	18	0/8	7.6e-13	6	4	1/8
cfd1	1.0e-12	162	4	0/8	9.9e-13	92	4	1/8	9.9e-13	112	4	0/8	9.7e-13	79	2	1/8
financ.	8.9e-13	58	3	0/8	1.3e-12	54	2	1/8	2.5e-12	69	3	0/8	1.0e-12	49	2	1/8
Ga10As.	2.9e-10	800	10	0/8	3.6e-12	636	7	3/8	7.7e-09	589	16	0/8	4.3e-11	474	6	3/8
Ga3As3.	9.9e-13	251	6	0/8	9.8e-13	165	5	1/8	2.1e-07	288	30	0/8	9.7e-13	112	4	1/8
computing k largest eigpairs with adaptive polynomial degree																
Andrew.	9.2e-11	133	9	0/5	7.2e-12	49	4	1/5	2.6e-12	156	17	0/5	5.9e-12	55	3	1/5
C60	4.9e-12	18	10	0/8	8.2e-12	6	3	1/8	9.1e-12	21	22	0/8	9.7e-13	6	4	1/8
cfd1	6.4e-12	147	6	0/3	6.9e-12	79	5	2/3	7.4e-12	213	18	0/3	9.6e-12	86	4	1/3
financ.	1.0e-12	112	5	0/3	1.0e-12	34	3	2/3	2.6e-12	192	16	0/3	2.3e-12	57	2	1/3
Ga10As.	8.2e-12	669	6	0/5	5.0e-12	269	5	1/5	8.5e-12	860	19	0/5	1.0e-12	353	4	1/5
Ga3As3.	8.4e-12	283	9	0/5	1.1e-12	135	5	1/5	5.4e-12	312	19	0/5	1.3e-12	118	4	1/4
computing k smallest eigpairs by fix deg = 8																
Andrew.	4.2e-12	179	6	0/8	3.6e-13	86	5	1/8	1.0e-11	218	19	0/8	8.7e-12	84	4	1/8
C60	3.0e-12	12	6	0/8	9.1e-13	10	6	1/8	2.7e-12	15	16	0/8	1.1e-12	6	4	1/8
cfd1	3.8e-05	1412	30	0/8	9.8e-12	809	21	3/8	1.2e-04	1284	30	0/8	6.6e-12	813	24	3/8
financ.	3.7e-07	584	30	0/8	7.0e-12	277	10	3/8	8.1e-06	689	30	0/8	1.3e-12	343	12	3/8
Ga10As.	2.1e-05	1508	9	0/8	2.1e-12	935	11	3/8	2.6e-03	2431	11	0/8	6.4e-06	936	5	3/8
Ga3As3.	7.4e-12	543	8	0/8	9.9e-12	229	4	1/8	4.9e-12	523	22	0/8	9.7e-13	183	4	1/8
computing k smallest eigpairs with adaptive polynomial degree																
Andrew.	5.0e-12	203	8	0/8	6.9e-12	84	4	1/8	5.0e-12	245	21	0/8	1.0e-12	81	4	1/8
C60	4.7e-12	13	9	0/7	3.5e-12	6	8	1/6	8.1e-12	21	19	0/7	5.0e-12	6	6	1/6
cfd1	1.7e-08	1302	30	0/15	4.0e-12	391	6	2/15	8.1e-07	1207	30	0/15	7.6e-12	565	16	3/15
financ.	4.6e-12	492	12	0/15	8.9e-12	191	6	1/15	4.9e-12	572	26	0/15	1.5e-12	238	7	1/15
Ga10As.	3.8e-06	1992	12	0/8	9.3e-12	713	5	1/8	5.5e-11	2281	29	0/8	1.8e-12	628	6	1/8
Ga3As3.	1.7e-12	619	9	0/8	1.4e-12	259	4	1/8	4.1e-12	550	23	0/8	1.1e-12	214	6	1/8



(a) matrix: cfd1 (b) matrix: finance

Fig. 6.1. ARR vs RR: Iteration history of `maxres` for computing k largest eigenpairs.



(a) matrix: cfd1 (b) matrix: finance

Fig. 6.2. ARR vs RR: Iteration history of `maxres` for computing k smallest eigenpairs.

- In some rare cases (e.g. MPM-ARR Adap. deg.), "maxres" may increase occasionally but the trend is often corrected in the next outer iteration. The main reason of the increase is that a few eigenvectors with large residual may appear after ARR. However, other eigenvectors are very likely to converge better. This phenomenon is usually caused by a rank-deficient X (note that X may have lost rank before RR), which indicates that a part of the desired eigen-space is lost.

6.3. 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 6.1. We compare two schemes: the first is to use a fix degree among $\{4, 8, 15\}$ and skip line 33 of Algorithm 5.1, and the second is the adaptive scheme in Algorithm 5.1. The computational results are summarized in Table 6.3. We also plot the iteration history of `maxres`, for computing both k largest and smallest eigenpairs on the matrices cfd1 and finance in Figures 6.3 and 6.4, 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.

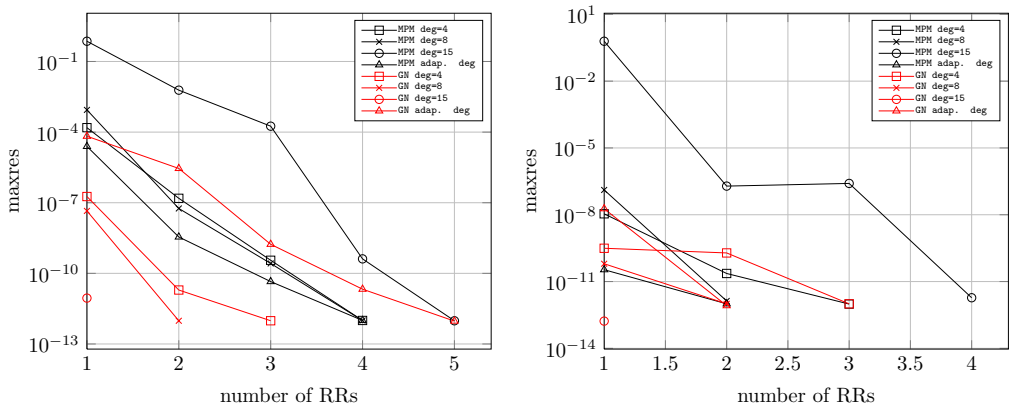
- In general, the number of outer iterations is decreased as the polynomial degree is increased, but the wall-clock time is not necessarily reduced because of the extra cost in using higher-degree polynomials. Overall, our adaptive strategy seems to have achieved a reasonable balance.

- With fixed polynomial degrees, in a small number of test case MPM and GN fail to reach the required accuracy.

Table 6.3: Comparison results of different polynomial degrees on tol=1e-12.

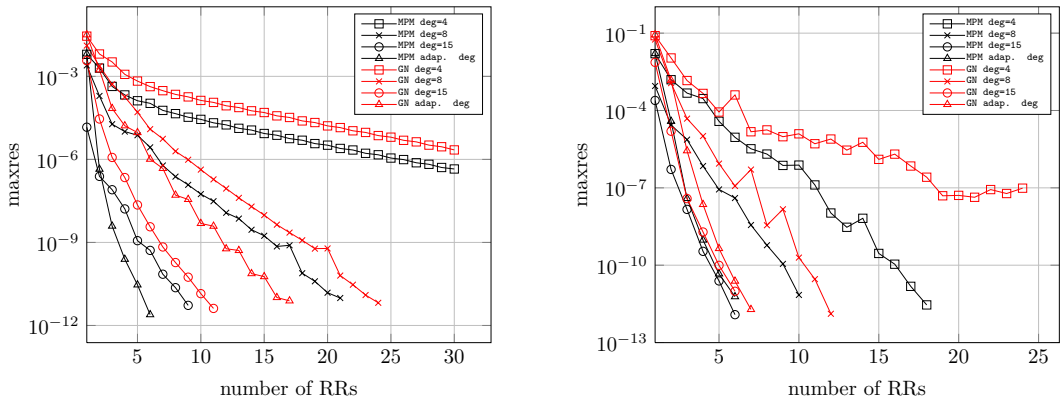
	deg=4				deg=8				deg=15				adaptive deg			
matrix	maxres	time	RR	p/d	maxres	time	RR	p/d	maxres	time	RR	p/d	maxres	time	RR	p/d
MPM for k largest eigpairs																
Andrew.	1.2e-12	44	4	1/4	8.7e-13	47	3	1/8	9.5e-13	126	5	1/15	7.2e-12	49	4	1/5
C60	8.0e-12	5	6	3/4	4.1e-12	6	3	1/8	7.4e-12	8	3	1/15	8.2e-12	6	3	1/8
cfd1	1.0e-12	69	4	1/4	9.9e-13	92	4	1/8	9.7e-13	162	5	1/15	6.9e-12	79	5	2/3
financ.	9.7e-13	40	3	1/4	1.3e-12	54	2	1/8	1.9e-12	85	4	1/15	1.0e-12	34	3	2/3
Ga10As.	7.6e-12	284	4	1/4	3.6e-12	636	7	3/8	2.8e-01	1683	4	1/15	5.0e-12	269	5	1/5
Ga3As3.	1.3e-12	109	4	1/4	9.8e-13	165	5	1/8	2.8e-01	512	4	1/15	1.1e-12	135	5	1/5
GN for k largest eigpairs																
Andrew.	1.5e-12	66	5	1/4	9.7e-13	53	2	1/8	8.7e-13	54	2	1/15	5.9e-12	55	3	1/5
C60	2.4e-13	10	11	3/4	7.6e-13	6	4	1/8	9.9e-13	4	2	1/15	9.7e-13	6	4	1/8
cfd1	9.7e-13	67	3	1/4	9.7e-13	79	2	1/8	8.9e-12	73	1	1/15	9.6e-12	86	4	1/3
financ.	1.0e-12	51	3	2/4	1.0e-12	49	2	1/8	1.7e-13	52	1	1/15	2.3e-12	57	2	1/3
Ga10As.	2.7e-11	823	18	3/4	4.3e-11	474	6	3/8	0.0e+00	2578	30	1/15	1.0e-12	353	4	1/5
Ga3As3.	1.1e-12	151	7	2/4	9.7e-13	112	4	1/8	1.1e-01	1123	30	1/15	1.3e-12	118	4	1/4
MPM for k smallest eigpairs																
Andrew.	7.2e-13	107	9	3/4	3.6e-13	86	5	1/8	9.4e-13	109	4	1/15	6.9e-12	84	4	1/8
C60	7.1e-12	6	8	3/4	9.1e-13	10	6	1/8	8.3e-13	7	5	1/15	3.5e-12	6	8	1/6
cfd1	4.5e-07	876	30	3/4	9.8e-12	809	21	3/8	5.3e-12	416	9	3/15	4.0e-12	391	6	2/15
financ.	2.9e-12	573	18	3/4	7.0e-12	277	10	3/8	1.2e-12	158	6	1/15	8.9e-12	191	6	1/15
Ga10As.	2.1e-12	1028	10	3/4	2.1e-12	935	11	3/8	6.2e-10	1664	8	2/15	9.3e-12	713	5	1/8
Ga3As3.	1.8e-12	312	9	3/4	9.9e-12	229	4	1/8	9.7e-13	416	4	1/15	1.4e-12	259	4	1/8
GN for k smallest eigpairs																
Andrew.	1.2e-12	153	10	3/4	8.7e-12	84	4	1/8	9.7e-13	71	2	1/15	1.0e-12	81	4	1/8
C60	7.6e-12	8	9	3/4	1.1e-12	6	4	1/8	4.4e-13	10	8	3/15	5.0e-12	6	6	1/6
cfd1	2.2e-06	1085	30	3/4	6.6e-12	813	24	3/8	4.1e-12	443	11	3/15	7.6e-12	565	16	3/15
financ.	4.3e-08	646	24	3/4	1.3e-12	343	12	3/8	9.8e-12	182	6	1/15	1.5e-12	238	7	1/15
Ga10As.	7.1e-12	1162	13	3/4	6.4e-06	936	5	3/8	1.3e-12	515	3	1/15	1.8e-12	628	6	1/8
Ga3As3.	4.9e-12	317	10	3/4	9.7e-13	183	4	1/8	9.9e-13	210	3	1/15	1.1e-12	214	6	1/8

Finally, we compare the performance of Algorithm 5.1 either using Chebyshev interpolants defined in (4.3) or the Chebyshev polynomials defined in (2.7) on the first six matrices in Table 6.1. The comparison results are given in Table 6.4. Even though both types of polynomials work well on these six problems, some performance differences are still observable in favor of our polynomials.



(a) matrix: cfd1 (b) matrix: finance

Fig. 6.3. ARR: Iteration history of `maxres` for computing k largest eigenpairs using different polynomial degrees.



(a) matrix: cfd1 (b) matrix: finance

Fig. 6.4. ARR: Iteration history of `maxres` for computing k smallest eigenpairs using different polynomial degrees.

6.4. Comparison with ARPACK and SLEPc

We now compare MPM and GN with ARPACK and SLEPc for computing both k largest and smallest eigenpairs for the sixteen test matrices presented in Table 6.1 (which also lists the k values). Computational results are summarized in Tables 6.5 and 6.6, 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 ARPACK is measured by the quantity $\log_2(\text{time}_{\text{ARPACK}}/\text{time})$, as shown in Figures 6.5 and 6.6 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 ARPACK; that is, when moving from the left towards the right, problems become progressively more and more time-consuming for ARPACK to solve. A quick glance at the figures tells us that MPM and GN provide clear speedups over ARPACK on most problems, especially on the more time-consuming problems towards the

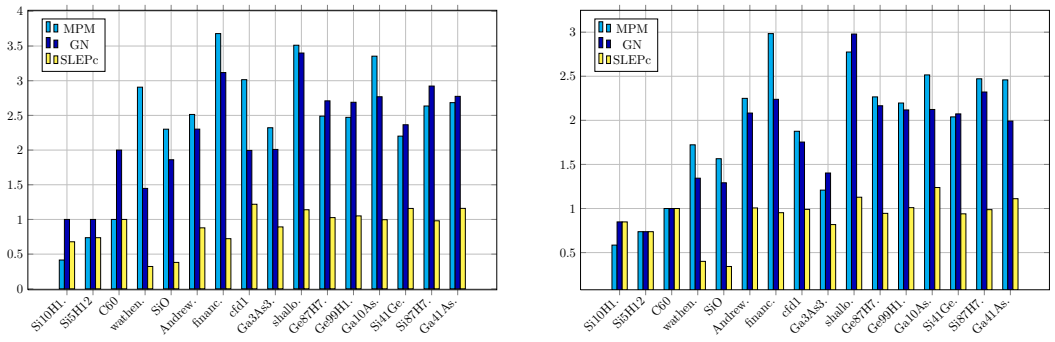
Table 6.4: Comparison results on Chebyshev interpolants in (4.3) and Chebyshev polynomials in (2.7).

	MPM				MPM, Cheb. poly.				GN				GN, Cheb. poly.			
matrix	maxres	time	RR	p/d	maxres	time	RR	p/d	maxres	time	RR	p/d	maxres	time	RR	p/d
computing k largest eigpairs, tol=1e-6																
Andrew.	8.8e-08	38	2	1/5	1.0e-06	31	2	1/3	4.3e-08	44	2	1/5	1.8e-07	43	2	1/3
C60	1.6e-08	6	2	1/8	2.2e-08	3	2	1/5	1.3e-06	3	2	1/7	2.6e-06	4	3	1/5
cfd1	6.2e-06	36	1	1/3	9.2e-08	47	2	1/2	4.9e-08	73	2	1/3	1.7e-07	43	1	1/3
financ.	3.7e-11	21	1	1/3	6.3e-07	18	1	1/3	7.5e-09	31	1	1/3	2.2e-07	32	1	1/3
Ga10As.	6.6e-08	152	2	1/5	9.1e-07	197	3	1/3	3.3e-06	228	2	1/5	3.1e-07	299	3	1/3
Ga3As3.	5.8e-08	62	2	1/5	9.7e-08	69	3	1/2	1.4e-06	77	2	1/5	6.9e-06	70	2	1/3
computing k largest eigpairs, tol=1e-12																
Andrew.	7.2e-12	49	4	1/5	5.9e-10	65	7	3/3	5.9e-12	55	3	1/5	3.1e-12	82	7	3/3
C60	8.2e-12	6	3	1/8	1.1e-12	5	5	2/5	9.7e-13	6	4	1/8	4.8e-12	10	13	3/2
cfd1	6.9e-12	79	5	2/3	3.9e-12	86	7	2/2	9.6e-12	86	4	1/3	9.4e-12	88	4	2/2
financ.	1.0e-12	34	3	2/3	3.7e-12	37	3	1/2	2.3e-12	57	2	1/3	6.1e-12	76	4	2/2
Ga10As.	5.0e-12	269	5	1/5	3.4e-12	439	10	3/3	1.0e-12	353	4	1/5	1.4e-11	777	15	3/3
Ga3As3.	1.1e-12	135	5	1/5	9.7e-12	131	8	3/2	1.3e-12	118	4	1/4	2.7e-12	151	9	2/3
computing k smallest eigpairs, tol=1e-6																
Andrew.	5.2e-07	54	2	1/8	1.1e-07	49	2	1/5	2.0e-06	55	2	1/8	2.4e-06	76	4	1/5
C60	3.5e-08	4	4	1/6	2.7e-06	4	3	1/4	2.1e-08	4	3	1/6	3.0e-07	5	5	2/4
cfd1	5.1e-07	234	2	1/15	1.7e-06	188	2	1/15	1.0e-05	249	5	2/15	4.5e-06	353	6	3/15
financ.	3.1e-06	106	2	1/15	6.9e-06	92	2	1/11	6.0e-06	114	3	1/15	9.6e-06	110	3	1/11
Ga10As.	3.0e-06	370	2	1/8	7.3e-07	417	3	1/5	1.1e-07	403	3	1/8	2.9e-06	637	4	1/5
Ga3As3.	1.4e-07	175	2	1/8	3.7e-07	155	2	1/5	4.7e-08	145	3	1/8	4.6e-06	194	5	2/5
computing k smallest eigpairs, tol=1e-12																
Andrew.	6.9e-12	84	4	1/8	5.6e-12	94	9	3/5	1.0e-12	81	4	1/8	4.5e-12	138	11	3/5
C60	3.5e-12	6	8	1/6	1.2e-12	8	10	3/4	5.0e-12	6	6	1/6	1.6e-12	9	12	3/4
cfd1	4.0e-12	391	6	2/15	6.3e-12	440	15	3/15	7.6e-12	565	16	3/15	9.3e-12	697	26	3/15
financ.	8.9e-12	191	6	1/15	5.8e-12	207	9	3/11	1.5e-12	238	7	1/15	3.5e-12	286	13	3/11
Ga10As.	9.3e-12	713	5	1/8	7.7e-12	850	9	3/2	1.8e-12	628	6	1/8	3.4e-06	714	7	1/2
Ga3As3.	1.4e-12	259	4	1/8	1.6e-12	320	9	3/5	1.1e-12	214	6	1/8	7.0e-10	397	30	3/2

right. For example, MPM and GN deliver a speedup of about 4 times on each of the seven most time-consuming problems in Fig. 6.5(a), and a speedup of about 10 times on the most time-consuming problem Ga41As41H72 in Fig. 6.6(a). On the other hand, SLEPc also provides good acceleration over ARPACK. Its number of SpMV is at the same level as ARPACK. For small matrices such as C60 and Si5H12, SLEPc is as fast as MPM and GN. On the other hand, for larger matrices, it takes SLEPc about twice the time of MPM and GN to solve the problems.

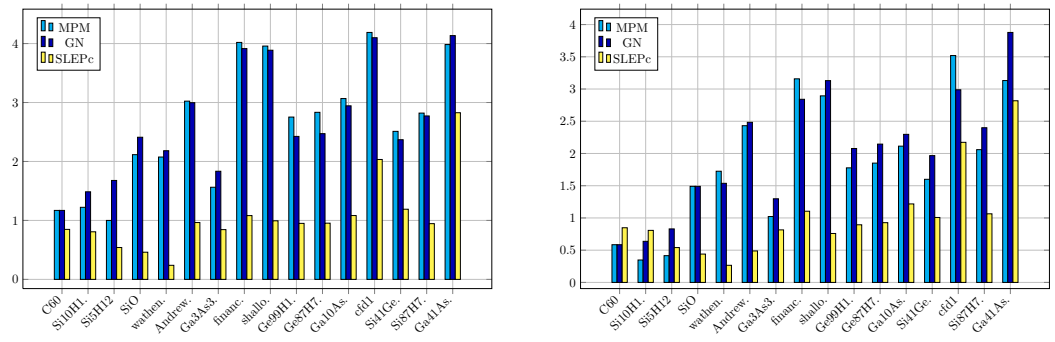
The benchmark solver ARPACK usually, though not always, returns solutions more accurate than what is requested by the tolerance value. In particular, for tol $= 10^{-6}$ the accuracy of ARPACK solutions often reach the order of $O(10^{-12})$. This is due to the fact that ARPACK needs to maintain a high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

As is observed previously, it is often more time-consuming for ARPACK, MPM and GN to compute k smallest eigenpairs than k largest ones on many test matrices. By examining the



(a) tol $= 10^{-6}$ (b) tol $= 10^{-12}$

Fig. 6.5. Speedup to ARPACK: $\log_2(\text{time}_{\text{ARPACK}}/\text{time})$ on computing k largest eigenpairs.



(a) tol $= 10^{-6}$ (b) tol $= 10^{-12}$

Fig. 6.6. Speedup to ARPACK: $\log_2(\text{time}_{\text{ARPACK}}/\text{time})$ on computing k smallest eigenpairs.

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.

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 an overall speed advantage over ARPACK, especially on those problems more time-consuming for ARPACK (towards the right end of Figures 6.5 and 6.6). They are faster in spite of taking considerably more matrix-vector multiplications than ARPACK, as can be seen from Tables 6.5 and 6.6, thanks to the benefits of relying on high-concurrency operations on many-core computers.

- MPM and GN generally require a small 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

Table 6.5: Comparison results on computing k largest eigenpairs.

	ARPACK			SLEPc			MPM				GN			
name	maxres	time	SpMV	maxres	time	SpMV	maxres	time	SpMV	RR/p/d	maxres	time	SpMV	RR/p/d
						tol=1e-6								
Andrew.	3.9e-08	217	3e+03	8.5e-06	118	5e+03	8.8e-08	38	8e+04	2/1/5	4.3e-08	44	7e+04	2/1/5
C60	3.6e-08	12	2e+03	6.9e-06	6	3e+03	1.6e-08	6	7e+04	2/1/8	1.3e-06	3	3e+04	2/1/7
cfd1	3.0e-14	291	3e+03	9.7e-06	125	5e+03	6.2e-06	36	3e+04	1/1/3	4.9e-08	73	6e+04	2/1/3
financ.	1.3e-14	269	3e+03	9.5e-06	163	5e+03	3.7e-11	21	3e+04	1/1/3	7.5e-09	31	3e+04	1/1/3
Ga10As.	5.7e-14	1554	8e+03	9.7e-06	779	1e+04	6.6e-08	152	1e+05	2/1/5	3.3e-06	228	1e+05	2/1/5
Ga3As3.	8.0e-07	310	5e+03	8.6e-06	167	6e+03	5.8e-08	62	7e+04	2/1/5	1.4e-06	77	6e+04	2/1/5
shallo.	7.7e-07	844	8e+03	9.9e-06	383	9e+03	7.8e-09	74	2e+05	2/1/7	4.8e-08	80	1e+05	2/1/7
Si10H1.	2.0e-09	8	2e+03	3.9e-06	5	2e+03	4.0e-10	6	4e+04	2/1/6	3.8e-08	4	3e+04	2/1/6
Si5H12	3.5e-11	10	2e+03	3.1e-06	6	2e+03	4.6e-10	6	4e+04	2/1/6	8.1e-09	5	3e+04	2/1/6
SiO	1.1e-10	69	3e+03	1.6e-06	53	4e+03	6.3e-09	14	6e+04	2/1/5	5.3e-08	19	6e+04	2/1/5
wathen.	2.1e-06	30	2e+03	7.6e-06	24	3e+03	9.5e-06	4	2e+04	1/1/3	1.7e-08	11	4e+04	2/1/5
Ge87H7.	3.7e-14	1544	8e+03	9.9e-06	758	1e+04	9.6e-10	275	2e+05	2/1/6	3.6e-08	236	1e+05	2/1/6
Ge99H1.	3.3e-14	1548	8e+03	9.8e-06	747	1e+04	6.2e-10	279	2e+05	2/1/6	2.5e-08	240	1e+05	2/1/6
Si41Ge.	1.7e-08	2612	9e+03	9.9e-06	1170	1e+04	1.2e-09	568	3e+05	2/1/7	2.1e-08	507	2e+05	2/1/7
Si87H7.	2.6e-11	3921	1e+04	9.4e-06	1987	1e+04	1.0e-09	631	3e+05	2/1/7	5.8e-08	517	2e+05	2/1/7
Ga41As.	1.1e-13	4106	1e+04	9.1e-06	1838	1e+04	1.1e-09	639	2e+05	3/1/5	2.0e-06	600	1e+05	2/1/6
name	maxres	time	SpMV	maxres	time	SpMV	maxres	time	SpMV	RR/p/d	maxres	time	SpMV	RR/p/d
						tol=1e-12								
Andrew.	2.8e-14	233	4e+03	8.4e-12	116	5e+03	7.2e-12	49	1e+05	4/1/5	5.9e-12	55	8e+04	3/1/5
C60	2.1e-14	12	3e+03	9.2e-12	6	3e+03	8.2e-12	6	7e+04	3/1/8	9.7e-13	6	5e+04	4/1/8
cfd1	2.4e-14	290	3e+03	9.9e-12	146	5e+03	6.9e-12	79	9e+04	5/2/3	9.6e-12	86	8e+04	4/1/3
financ.	2.0e-14	269	3e+03	9.0e-12	139	5e+03	1.0e-12	34	5e+04	3/2/3	2.3e-12	57	6e+04	2/1/3
Ga10As.	5.4e-14	1538	8e+03	8.9e-12	652	1e+04	5.0e-12	269	2e+05	5/1/5	1.0e-12	353	2e+05	4/1/5
Ga3As3.	8.0e-14	312	5e+03	9.6e-12	177	6e+03	1.1e-12	135	1e+05	5/1/5	1.3e-12	118	9e+04	4/1/4
shallo.	3.8e-14	883	8e+03	9.6e-12	404	1e+04	9.9e-13	129	3e+05	4/1/7	1.3e-12	112	1e+05	3/1/7
Si10H1.	2.3e-14	9	2e+03	8.2e-12	5	2e+03	1.5e-12	6	4e+04	3/1/6	9.9e-13	5	3e+04	3/1/6
Si5H12	1.1e-14	10	2e+03	8.5e-12	6	2e+03	9.6e-13	6	4e+04	3/1/6	1.7e-12	6	3e+04	3/1/6
SiO	1.8e-14	71	3e+03	6.5e-12	56	4e+03	9.2e-13	24	7e+04	4/1/5	9.9e-13	29	7e+04	4/1/5
wathen.	1.7e-14	33	2e+03	5.7e-12	25	3e+03	9.3e-13	10	4e+04	3/1/5	1.0e-12	13	4e+04	3/1/5
Ge87H7.	2.6e-13	1467	8e+03	9.4e-12	762	1e+04	2.3e-12	305	2e+05	3/1/6	9.5e-13	327	2e+05	3/1/6
Ge99H1.	4.0e-13	1476	8e+03	9.4e-12	733	1e+04	1.3e-12	322	2e+05	3/1/6	9.9e-13	340	2e+05	3/1/6
Si41Ge.	3.2e-14	2647	9e+03	9.5e-12	1380	1e+04	4.6e-12	644	2e+05	3/1/7	9.8e-13	629	2e+05	3/1/7
Si87H7.	4.1e-14	3649	1e+04	9.6e-12	1841	1e+04	8.2e-12	658	3e+05	3/1/7	1.0e-12	730	2e+05	3/1/7
Ga41As.	9.2e-13	3915	9e+03	7.1e-12	1812	1e+04	9.5e-13	712	2e+05	3/1/7	9.0e-13	984	2e+05	4/1/7

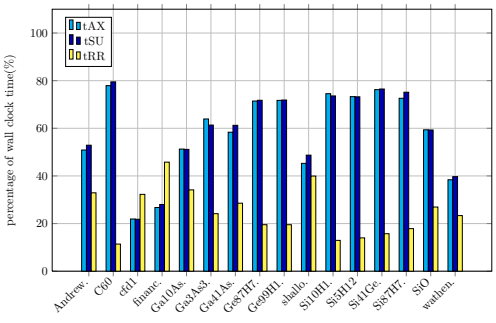
polynomial degree never reaches the maximum degree 15 except on cfd1, finance and wathen100 when computing k smallest eigenpairs.

In Fig. 6.7, we plot runtimes of three categories: SpMV (i.e., AX), SU (lines 10 to 22 of Algorithm 5.1) and ARR (lines 23 to 27 of Algorithm 5.1). 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, mostly done in batch of $k+q$, dominates the entire computation in

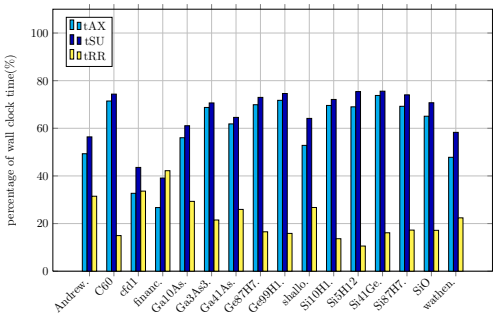
Table 6.6: Comparison results on computing k smallest eigenpairs.

	ARPACK			SLEPc			MPM				GN			
name	maxres	time	SpMV	maxres	time	SpMV	maxres	time	SpMV	RR/p/d	maxres	time	SpMV	RR/p/d
						tol=1e-6								
Andrew.	5.1e-07	439	7e+03	9.2e-06	225	8e+03	5.2e-07	54	1e+05	2/1/8	2.0e-06	55	1e+05	2/1/8
C60	7.9e-08	9	2e+03	7.2e-07	5	2e+03	3.5e-08	4	4e+04	4/1/6	2.1e-08	4	3e+04	3/1/6
cfd1	1.9e-08	4267	6e+04	6.5e-08	1042	3e+04	5.1e-07	234	8e+05	2/1/15	1.0e-05	249	6e+05	5/2/15
financ.	2.4e-09	1719	2e+04	9.8e-06	812	2e+04	3.1e-06	106	4e+05	2/1/15	6.0e-06	114	3e+05	3/1/15
Ga10As.	2.3e-12	3101	2e+04	9.7e-06	1464	2e+04	3.0e-06	370	4e+05	2/1/8	1.1e-07	403	2e+05	3/1/8
Ga3As3.	5.4e-07	517	8e+03	9.6e-06	288	1e+04	1.4e-07	175	2e+05	2/1/8	4.7e-08	145	1e+05	3/1/8
shallo.	1.7e-13	1864	2e+04	9.8e-06	936	2e+04	6.8e-08	120	4e+05	2/1/13	2.0e-06	126	2e+05	3/2/8
Si10H1.	5.0e-10	14	3e+03	8.6e-06	8	3e+03	4.8e-07	6	6e+04	2/1/7	9.0e-06	5	3e+04	2/1/7
Si5H12	2.1e-06	16	3e+03	8.4e-06	11	3e+03	7.2e-08	8	7e+04	2/1/7	1.6e-06	5	3e+04	2/1/8
SiO	2.8e-09	117	5e+03	5.2e-06	85	6e+03	3.6e-07	27	1e+05	2/1/7	5.7e-07	22	7e+04	2/1/8
wathen.	2.4e-06	118	8e+03	9.9e-06	100	8e+03	7.6e-08	28	2e+05	3/1/15	6.4e-06	26	1e+05	3/1/15
Ge87H7.	2.7e-06	2324	1e+04	6.3e-06	1200	2e+04	1.0e-06	326	3e+05	2/1/9	3.8e-10	419	3e+05	3/1/9
Ge99H1.	3.8e-10	2299	1e+04	9.9e-06	1190	2e+04	8.0e-07	341	3e+05	2/1/9	2.8e-10	428	3e+05	3/1/9
Si41Ge.	8.6e-10	4303	1e+04	9.9e-06	1886	2e+04	3.2e-06	755	4e+05	2/1/11	8.5e-10	833	3e+05	3/1/11
Si87H7.	3.2e-06	5800	2e+04	7.8e-06	3012	2e+04	5.3e-09	821	4e+05	3/1/11	1.6e-08	849	3e+05	3/1/11
Ga41As.	3.4e-06	34060	8e+04	7.4e-06	4800	3e+04	2.3e-06	2154	7e+05	3/1/11	1.3e-08	1939	5e+05	4/1/11
name	maxres	time	SpMV	maxres	time	SpMV	maxres	time	SpMV	RR/p/d	maxres	time	SpMV	RR/p/d
						tol=1e-12								
Andrew.	1.2e-13	453	7e+03	7.0e-12	323	1e+04	6.9e-12	84	2e+05	4/1/8	1.0e-12	81	2e+05	4/1/8
C60	7.6e-14	9	2e+03	1.5e-12	5	2e+03	3.5e-12	6	7e+04	8/1/6	5.0e-12	6	4e+04	6/1/6
cfd1	3.0e-14	4483	6e+04	6.6e-14	994	3e+04	4.0e-12	391	1e+06	6/2/15	7.6e-12	565	1e+06	16/3/15
financ.	3.7e-13	1705	2e+04	9.9e-12	793	2e+04	8.9e-12	191	7e+05	6/1/15	1.5e-12	238	6e+05	7/1/15
Ga10As.	2.4e-12	3086	2e+04	8.0e-12	1327	2e+04	9.3e-12	713	7e+05	5/1/8	1.8e-12	628	4e+05	6/1/8
Ga3As3.	1.1e-12	526	8e+03	7.7e-12	299	1e+04	1.4e-12	259	3e+05	4/1/8	1.1e-12	214	2e+05	6/1/8
shallo.	1.9e-13	1821	2e+04	9.4e-12	1076	2e+04	1.8e-12	245	7e+05	6/2/14	9.8e-13	208	4e+05	5/1/14
Si10H1.	7.8e-14	14	3e+03	7.0e-12	8	3e+03	8.6e-13	11	7e+04	5/1/7	1.9e-12	9	5e+04	5/1/7
Si5H12	1.3e-13	16	3e+03	7.7e-12	11	3e+03	2.8e-12	12	8e+04	5/1/7	1.1e-12	9	5e+04	4/1/7
SiO	2.2e-13	118	5e+03	8.9e-12	87	6e+03	2.7e-12	42	2e+05	5/1/7	4.5e-13	42	1e+05	5/1/7
wathen.	8.2e-13	119	8e+03	8.7e-12	99	9e+03	8.0e-12	36	3e+05	6/2/15	7.2e-12	41	2e+05	7/2/15
Ge87H7.	1.8e-13	2420	1e+04	6.4e-12	1273	2e+04	4.3e-12	671	5e+05	4/1/9	9.9e-13	547	3e+05	5/1/9
Ge99H1.	1.7e-13	2334	1e+04	5.5e-12	1256	2e+04	2.4e-12	681	5e+05	4/1/9	9.7e-13	553	3e+05	5/1/9
Si41Ge.	2.8e-13	4288	2e+04	5.4e-12	2133	2e+04	1.4e-12	1415	6e+05	5/1/11	2.8e-12	1097	4e+05	5/1/11
Si87H7.	3.0e-13	6094	2e+04	6.1e-12	2914	2e+04	2.4e-12	1462	7e+05	6/1/11	7.4e-12	1155	4e+05	5/1/11
Ga41As.	5.0e-12	33694	8e+04	8.7e-12	4778	3e+04	3.3e-12	3845	1e+06	7/3/11	1.2e-12	2292	6e+05	6/1/11

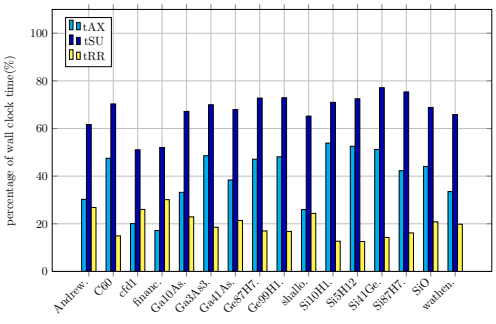
almost all cases. These trends are much more pronounced (a) for MPM than for GN (recall that GN requires to solve $k \times k$ linear systems); and (b) for computing k smallest eigenpairs than for computing k largest ones (recall that the former is generally more difficult). These runtime profiles are favorable to parallel scalability since AX operations possess high concurrency for relatively large k.



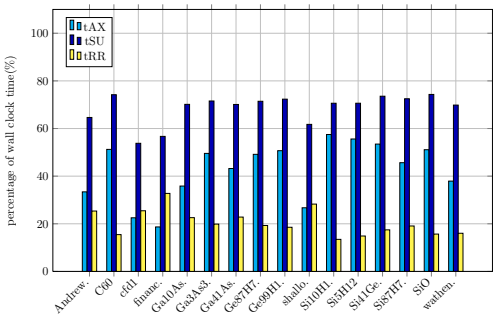
(a) tol $= 10^{-6}$, MPM, k largest eigenpairs



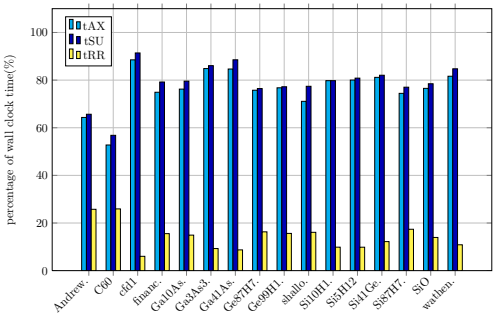
(b) tol $= 10^{-12}$, MPM, k largest eigenpairs



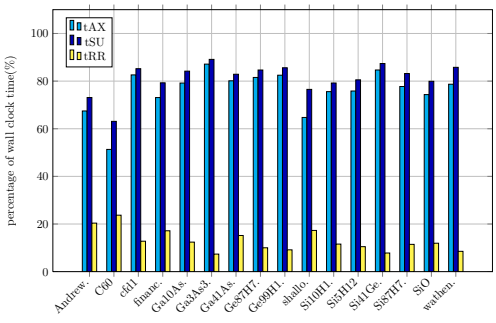
(c) tol $= 10^{-6}$, GN, k largest eigenpairs



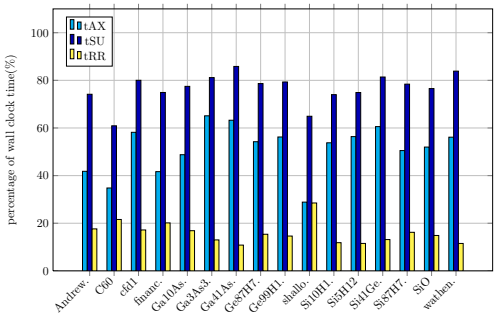
(d) tol $= 10^{-12}$, GN, k largest eigenpairs



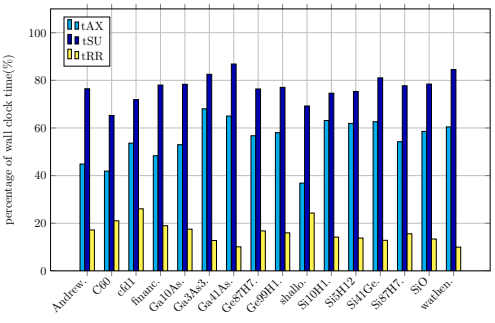
(e) tol $= 10^{-6}$, MPM, k smallest eigenpairs



(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. 6.7. A comparison of timing profile among SpMV, SU and ARR.

7. 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 stopping 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 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 ARPACK, without explicit code parallelization or 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 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.

Acknowledgments. The research of Z. Wen was supported in part by the NSFC grants 11831002 and 91730302 and by Beijing Academy of Artificial Intelligence (BAAI). 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), 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), 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), A2220–A2246.
[7] V. Hernandez, J.E. Roman, A. Tomas, and V. Vidal, *A survey of software for sparse eigenvalue problems*, Tech. Rep. STR-6, Universitat Politècnica de València, 2009. Available at http://slepc.upv.es.
[8] V. Hernandez, J. E. Roman, and V. Vidal, SLEPc: A scalable and flexible toolkit for the solution of eigenvalue problems, *ACM Trans. Math. Software*, **31** (2005), 351–362.
[9] A. Knyazev, M. Argentati, I. Lashuk, and E. Ovtchinnikov, Block locally optimal preconditioned eigenvalue xolvers (BLOPEX) in Hypre and PETSc, *SIAM Journal on Scientific Computing*, **29** (2007), 2224–2239.
[10] A.V. Knyazev, Toward the optimal preconditioned eigensolver: locally optimal block preconditioned conjugate gradient method, *SIAM J. Sci. Comput.*, **23** (2001), 517–541.
[11] L. Kronik, A. Makmal, M. Tiago, M.M.G. Alemany, X. Huang, Y. Saad, and J.R. Chelikowsky, PARSEC – the pseudopotential algorithm for real-space electronic structure calculations: recent advances and novel applications to nanostructures, *Phys. Stat. Solidi. (b)*, **243** (2006), 1063–1079.
[12] 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), 225–282.
[13] R.M. Larsen, *Lanczos bidiagonalization with partial reorthogonalization*, Aarhus University, Technical report, DAIMI PB-357, September 1998.
[14] R.B. Lehoucq, Implicitly restarted Arnoldi methods and subspace iteration, *SIAM J. Matrix Anal. Appl.*, **23** (2001), 551–562.
[15] 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.
[16] 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), A1641–A1668.
[17] X. Liu, Z. Wen, and Y. Zhang, An efficient Gauss-Newton algorithm for symmetric low-rank product matrix approximations, *SIAM J. Optim.*, **25** (2015), 1571–1608.
[18] G. Mastroianni and J. Szabados, Jackson order of approximation by lagrange interpolation. ii, *Acta Mathematica Hungarica*, **69** (1995), 73–82.
[19] B. Parlett, *The Symmetric Eigenvalue Problem*, Prentice-Hall, 1980.
[20] E. Polizzi, Density-matrix-based algorithm for solving eigenvalue problems, *Phys. Rev. B*, **79** (2009), 115112.
[21] J.E. Roman, C. Campos, E. Romero, and A. Tomas, *SLEPc users manual*, Tech. Rep. DSIC-II/24/02 - Revision 3.7, D. Sistemes Informàtics i Computació, Universitat Politècnica de València, 2016.
[22] H. Rutishauser, Computational aspects of F. L. Bauer's simultaneous iteration method, *Numer. Math.*, **13** (1969), 4–13.
[23] H. Rutishauser, Simultaneous iteration method for symmetric matrices, *Numer. Math.*, **16** (1970), 205–223.
[24] Y. Saad, Chebyshev acceleration techniques for solving nonsymmetric eigenvalue problems, *Mathematics of Computation*, **42** (1984), 567–588.

[25] Y. Saad, *Numerical Methods for Large Eigenvalue Problems*, Manchester University Press, 1992.
[26] A.H. Sameh and J. A. Wisniewski, A trace minimization algorithm for the generalized eigenvalue problem, *SIAM Journal on Numerical Analysis*, **19** (1982), 1243–1259.
[27] 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, 119–165.
[28] 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), 268–290.
[29] G.W. Stewart, Simultaneous iteration for computing invariant subspaces of non-Hermitian matrices, *Numer. Math.*, **25** (1975/76), 123–136.
[30] G.W. Stewart, *Matrix algorithms volume 2: eigensystems*, Siam, **2** (2001).
[31] W.J. Stewart and A. Jennings, A simultaneous iteration algorithm for real matrices, *ACM Trans. Math. Software*, **7** (1981), 184–198.
[32] 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), 354–390.
[33] Z. Wen and Y. Zhang, Accelerating convergence by augmented rayleigh–ritz projections for large-scale eigenpair computation, *SIAM Journal on Matrix Analysis and Applications*, **38** (2017), 273–296.
[34] Y. Zhou and Y. Saad, A Chebyshev–Davidson algorithm for large symmetric eigenproblems, *SIAM J. Matrix Anal. and Appl.*, **29** (2007), 954–971.