Journal of Computational Physics 229 (2010) 9188–9200

Contents lists available at ScienceDirect

Journal of Computational Physics

journal homepage: www.elsevier.com/locate/jcp

A block Chebyshev–Davidson method with inner–outer restart for large eigenvalue problems [☆]

Yunkai Zhou

Department of Mathematics, Southern Methodist University, Dallas, TX 75275, USA

ARTICLE INFO

Article history:
Received 20 May 2008
Received in revised form 2 August 2010
Accepted 27 August 2010
Available online 6 September 2010

Keywords:
Chebyshev polynomial filter
Chebyshev–Davidson method
Subspace iteration
Restart
Block method
Block-wise filtering
Eigenproblem

ABSTRACT

We propose a block Davidson-type subspace iteration using Chebyshev polynomial filters for large symmetric/hermitian eigenvalue problem. The method consists of three essential components. The first is an adaptive procedure for constructing efficient block Chebyshev polynomial filters; the second is an inner–outer restart technique inside a Chebyshev–Davidson iteration that reduces the computational costs related to using a large dimension subspace; and the third is a progressive filtering technique, which can fully employ a large number of good initial vectors if they are available, without using a large block size. Numerical experiments on several Hamiltonian matrices from density functional theory calculations show the efficiency and robustness of the proposed method.

1. Introduction

One of the fundamental problems in computational science and engineering is the standard eigenproblem

$$Au = \lambda u, \tag{1}$$

where $A \in \mathbb{C}^{n\times n}$ is symmetric/hermitian, and n is large. Significant research efforts have been devoted to developing practical methods for large-scale eigenproblems in the passed several decades. There exist abundant literature in this field (see references in e.g. [1–5]). Typical methods include the Krylov subspace methods [6–9], preconditioned non-Krylov methods [10–15], and multigrid methods [16–20]. A rather comprehensive comparison of a few state-of-the-art eigensolvers was performed in [21], where the solvers were applied to compute a few interior eigenvalues encountered in electronic-structure calculations. The generalized Davidson GD + k method [14,15] was found to have better performance in terms of speed than other solvers compared in [21].

In many realistic applications, the matrices involved are of large-scale. An additional challenge from these applications is that the dimension of the wanted eigensubspace is also large; for example, one may need hundreds or thousands of eigenpairs of matrices with dimension over several millions. These applications often challenge the limit of both computer hardware and numerical algorithms.

Even though there exist many eigenvalue algorithms, not as many are specifically designed for computing a large number of eigenpairs, and relatively few are designed for effectively reusing a large number of good initial guesses when they are

☆ Work supported in part by NSF Grants CMMI-0727194 and OCI-0749074, by Department of Energy under contract DE-FG02-03ER25585, by the NSF through TeraGrid resources provided by TACC and NICS, and by SMU HPC center.

E-mail address: yzhou@smu.edu

0021-9991/$ - see front matter © 2010 Elsevier Inc. All rights reserved.
doi:10.1016/j.jcp.2010.08.032

available. This situation where a large number of good initial vectors exist is quite common in the self-consistent-field calculations in quantum chemistry and material science [22,23].

Subspace iteration (e.g. [4,1,3,2]) is the primary method that can take advantage of a large number of available initial vectors. The standard subspace iteration usually refers to the block power method or simultaneous iteration [24,25], i.e., the subspace used is of fixed dimension. Fixed dimension subspace iteration using Chebyshev filters for symmetric eigenvalue problems appeared in late 60's [24–26]. Later researches show that a Lanczos-type method (see [1, p. 330]) can be more efficient than a fixed dimension Chebyshev subspace iteration.

Block methods (e.g. [27–30]) are efficient for relatively small block sizes, but they become less efficient when the block size is over several hundreds. Moreover, a very large block size can lead to overwhelming memory demand. To reuse all available good initial vectors, most block methods use a block size that equals the number of initial vectors. This becomes impractical if there are hundreds or thousands or more good initial vectors. Therefore, standard block methods may not effectively utilize a large number of good initial vectors when they are available.

In [31] a Chebyshev–Davidson method is proposed, where the Chebyshev filtering is integrated into a Davidson-type subspace iteration. Therefore advantages related to varying dimension subspace iteration are preserved.

In this paper we develop a block version of the Chebyshev–Davidson algorithm proposed in [31]. A *two-indexes trick* is deployed, which allows flexible implementation of the restarted block method, including changing block sizes conveniently within the Chebyshev–Davidson iteration if necessary. Furthermore, we propose an *inner–outer restart* approach, which can effectively reduce the memory requirement, and a *progressive filtering* technique, which can fully exploit a large number of good initial vectors without the need to use an unnecessarily large block size.

The progressive filtering exploits an advantage a Davidson-type method has over a Lanczos-type method, namely that in a Davidson-type method one has more flexibility in augmenting the basis with new vectors. This advantage is gained by the cost of computing actual residual vectors after a Rayleigh–Ritz refinement at each iteration. Lanczos-type methods do not have to compute actual residuals, the saved computation leads to the constraint that they need to keeps a strict Krylov structure [32].

Our proposed block method can also work efficiently for the more common cases where no good initial vectors are available.

Throughout the paper, $\sigma(A)$ denotes the full spectrum of A, k_{want} denotes the number of wanted eigenvalues. For simplicity we assume that the wanted eigenvalues are the smallest k_{want} values of $\sigma(A)$.

2. Block Chebyshev polynomial filter and filter bounds

The first kind Chebyshev polynomial which is essential to our filtering method first appeared in 1854 [33]. The well-known polynomial is defined as (see e.g. [1, p. 371, 3, p. 142])

$$C_k(t) = \begin{cases} \cos(k\ \cos^{-1}(t)), & -1 \leqslant t \leqslant 1, \\ \cosh(k\ \cosh^{-1}(t)), & |t| > 1. \end{cases} \tag{2}$$

Associated with this definition is the following 3-term recurrence,

$$C_{k+1}(t) = 2tC_k(t) - C_{k-1}(t), \quad t \in \mathbb{R}, \tag{3}$$

here clearly $C_0(t) = 1$ and $C_1(t) = t$.

Note that $\cosh(kx) = (e^{kx} + e^{-kx})/2$, it is natural to expect the exponential growth of the Chebyshev polynomial outside the interval $[-1, 1]$. This useful property is well-emphasized in [1, p. 371]. In fact, under comparable conditions, $|C_k(x)|$ as defined in (2) grows fastest outside $[-1, 1]$ among all polynomials with degree $\leqslant k$ [34] [35, p. 31].

Let $\sigma(A) \subseteq [a_0, b]$. In order to approximate the lower end of $\sigma(A)$ contained in $[a_0, a]$ where $a_0 < a < b$, essentially one only needs to map $[a, b]$ into $[-1, 1]$. This is easily realized by an affine mapping defined as $\mathcal{L}(t) := (t - c)/e$, where $c = \frac{a+b}{2}$ and $e = \frac{b-a}{2}$ denote respectively the center and half-width of a given interval $[a, b]$. However, in many applications, the a can be unknown. (For example, one needs to compute a given number of smallest eigenvalues of a matrix.) This does not cause a problem for the Chebyshev–Davidson method, since essentially one only needs to dampen $[a_i, b]$ with $a_0 < a_i$, and then gradually update a_i so that all the wanted number of eigenvalues are computed. The filtering lower bound a_i at the ith iteration can be readily obtained from the Ritz values available at that iteration step. For the situation that a is known, (for example, one wants to compute all the eigenvalues no greater than a), the Chebyshev–Davidson method provides a more attractive approach since it can target directly at dampening $[a, b]$. In contrast, existing methods (e.g. [6,36,8,27–30]) usually do not provide direct means to compute eigenvalues on a given interval.

The block Chebyshev iteration, which utilizes the three-term recurrence (3) with the goal to dampen values on a given interval $[a, b]$ is presented in Algorithm 2.1. The iteration of the algorithm is equivalent to computing

$$Y = C_m(\mathcal{L}(A))X. \tag{4}$$

The iteration formula is adapted from the complex Chebyshev iteration used in [37]. However, unlike the non-block version used in [37,31], Algorithm 2.1 does not use a scaling factor normally applied to prevent overflow. This is sound in practice because the polynomial degree m used is not high, more importantly, the Chebyshev iteration is used inside the

Chebyshev–Davidson algorithm, which follows the Chebyshev iteration by a normalization step, this effectively prevents overflow, as shown by extensive usage of Algorithm 2.1 in a wide range of applications. Without using the scaling factor saves computation, and it waives the need to provide an estimate for the smallest eigenvalue.

The construction of Algorithm 2.1 appears to only target at dampening $[a, b]$ by mapping this interval into $[-1, 1]$. However, the property of "fastest growth outside $[-1, 1]$" of the Chebyshev polynomial automatically gives this filter significant power to magnify the part of spectrum to the left of $[a, b]$. This ideally corresponds to approximating the eigensubspace associated with the lower end of $\sigma(A)$.

Algorithm 2.1 shows that a desired Chebyshev filter can be easily adapted by changing the two endpoints of the higher end of $\sigma(A)$.

Algorithm 2.1. $[Y]$ = `Chebyshev_filter`(X, m, a, b).

```
Purpose: Filter vectors in $X$ by an $m$ degree Chebyshev polynomial that dampens on the interval $[a, b]$. Output the
   filtered vectors in $Y$.
 1. $e = (b - a)/2;\quad c = (b + a)/2;$
 2. $Y = (AX - cX)/e;$
 3. Do $i = 2$ to $m$
 4.       $Y_{new} = 2(AY - cY)/e - X;$
 5.       $X = Y;$
 6.       $Y = Y_{new};$
 7. End Do
```

The upper bound should bound $\sigma(A)$ from above in order not to magnify the higher end of the spectrum. An upper bound estimator is constructed in [38], it is based on a k-step standard Lanczos decomposition

$$AV_k = V_kT_k + f_ke_k^T,$$

where k is a small integer, V_k contains the k Lanczos basis, T_k is a size-k tridiagonal matrix, f_k is a vector, and e_k is a length-k unit vector with the first element equal to one. The bound estimator in [38] used $\max_{i\leqslant k}|\lambda_i(T_k)| + \|f_k\|_2$ as an upper bound for $\sigma(A)$. Although this upper bound estimator worked extremely well in practice, a theoretical proof that it guarantees an upper bound was not available in [38]. In [39] we provide such a theoretical proof after adding a natural condition that is usually true in practice. Here we improve the estimator by sharpening the bound as $\max_{i\leqslant k}\lambda_i(T_k) + \|f_k\|_2\|e_k^TQ_k\|_\infty$, where Q_k contains eigenvectors of T_k. Note that the absolute value $\max_{i\leqslant k}|\lambda_i(T_k)|$ is sharpened by $\max_{i\leqslant k}\lambda_i(T_k)$, and $\|f_k\|_2$ is scaled by a factor $\leqslant 1$. The improved estimator is listed in Algorithm 2.2. In practice, we found that $k = 5$ or 6 in Algorithm 2.2 is sufficient to provide a value that bounds $\sigma(A)$ from above.

Algorithm 2.2. Estimating an upper bound of $\sigma(A)$ by k-step Lanczos

```
 1. Generate a random vector $v$, set $v \leftarrow v/\|v\|_2$;
 2. Compute $f = Av$; $\alpha = f^Hv$; $f \leftarrow f - \alpha v$; $T(1,1) = \alpha$;
 3. Do $j = 2$ to $min(k,10)$
 4.      $\beta = \|f\|_2;\quad v_0 \leftarrow v;\quad v \leftarrow f/\beta;$
 5.      $f = A\,v;\quad f \leftarrow f - \beta v_0;$
 6.      $\alpha = f^Hv;\quad f \leftarrow f - \alpha v;$
 7.      $T(j, j-1) = \beta;\quad T(j-1, j) = \beta;\quad T(j, j) = \alpha;$
 8. End Do
 9. Compute eigen-decomposition of $T$: $TQ = QD$, where $Q$ is orthonormal. Return $\max_i\lambda_i(T) + \|f\|_2\|e^TQ\|_\infty$ as the upper
    bound.
```

The main advantage of Algorithm 2.2 is that it provides an effective upper bound via only a small number of matrix–vector products. This is useful since in many applications the matrices involved are not constructed explicitly but instead accessed via matrix–vector product subroutines. Moreover, the small number of matrix–vector products required implies that computing the upper bound costs very little extra computation.

The upper bound obtained from Algorithm 2.2 provides the higher (or right) endpoint, but the filter also needs a lower (or left) endpoint of the interval to be dampened. By utilizing the structure of a Davidson-type method, namely that Ritz values are computed at each iteration, we can obtain this lower endpoint quite conveniently without extra computation.

The lower endpoint for the first filtering step is easy to obtain: since at this step no eigenvalues of A have converged, the lower endpoint can be any value between $\min_i\lambda_i(A)$ and the computed upper bound. By the Courant–Fisher min–max theorem [1, p. 206] we see that any of $\lambda_i(T)$ computed in Algorithm 2.2 can be used for this purpose. We can also take a weighted value, e.g. $(\lambda_{min}(T) + \lambda_{max}(T))/2$, or $(3\,\lambda_{min}(T) + \lambda_{max}(T))/4$, such that about half or larger portion of the higher end of $\sigma(A)$ will be dampened at the first filtering. Alternatively, if several good initial vectors are available, we may compute the Rayleigh-quotients using some of the initial vectors and use the average of these Rayleigh-quotients as the lower endpoint for the first iteration.

At each step of the latter iterations, the Ritz values computed at the current Chebyshev–Davidson step readily provide a suitable lower endpoint for the filtering. The main idea is that we only need a lower endpoint that is larger than some wanted eigenvalues which have not converged. For example, we can use the largest Ritz value, or the second largest Ritz value, or a value between the two largest Ritz values. (Observed numerical results show only very minor differences among these choices.) The Courant–Fisher min–max theorem guarantees that a lower endpoint chosen this way will be larger than some wanted eigenvalues that have not converged, and at the same time smaller than the upper bound computed by Algorithm 2.2. Hence an effective block Chebyshev filter can be constructed from these two bounds by calling Algorithm 2.1.

As discussed above, the extra work associated with computing bounds for the block Chebyshev filtering is negligible. The main computations for the filtering are the matrix–vector products by the three-term recurrences in Algorithm 2.1 (*Steps 2* and *4*).

The polynomial degree m is a free parameter. It might be useful to adaptively adjust m as the iteration proceeds. Our numerical experiences show that a straightforward application of the technique in [25] (or [1, p. 330]) for adapting m is less satisfactory than using a fixed m. However, extensive numerical experiences show that an m for overall fast convergence is quite easy to choose. For example, an m within 15–25 often works well, and a slightly larger m (say, $m = 30$) may be preferred over a smaller m if the lower end of the spectrum is highly clustered. This rule of thumb agrees with the properties of Chebyshev polynomials.

3. Block Chebyshev–Davidson method with inner–outer restart

In this section, we present the block Chebyshev–Davidson subspace iteration with *inner–outer restart*. Throughout, V denotes the matrix whose columns form a basis of the current projection subspace; k_b denotes the block size; dim_{max} denotes the maximum dimension of the subspace spanned by V; and act_{max} denotes the maximum dimension of the active subspace. Here the *active subspace*, denoted as V_{act}, refers to the current subspace V deflated by the converged eigenvectors. Clearly, $act_{max} \leqslant dim_{max}$. Note that V_{act} does not consume additional memory, it is just part of the columns in V.

The *inner–outer restart* technique is introduced mainly to reduce memory requirement and reorthogonalization cost. The *outer-restart* corresponds to the standard (thick) restart, i.e., the subspace is reduced to a smaller size (denoted as k_{ro} plus the number of converged eigenvalues) when its dimension exceeds dim_{max}. The *inner-restart* corresponds to reducing the dimension of the active subspace to a smaller value (denoted as k_{ri}) when it exceeds act_{max}.

With inner–outer restart, the maximum dimension of V can be kept relatively small. This is because the *inner-restart* refines the basis vectors several times before the *outer-restart* becomes necessary. Therefore the dim_{max} can be kept small without scarifying convergence speed. Usually dim_{max} only needs to be slightly larger than k_{want}, e.g. $dim_{max} = k_{want} + act_{max}$. In contrast, Lanczos-type methods with standard restart often require the subspace to be of dimension $2 * k_{want}$ in order to compute k_{want} eigenvalues efficiently.

With act_{max} kept small and dim_{max} only slightly larger than k_{want}, the memory used by the block Chebyshev–Davidson method is about half of the memory required by Lanczos-type method with standard restart. Therefore, inner–outer restart technique can reduce memory requirement quite significantly, especially for large problems where k_{want} is over several thousands and the dimensions of matrices are over several millions.

The inner–outer restart is mainly based on two observations. The first one is that, in a Davidson-type method, a Rayleigh–Ritz refinement step is usually performed at each iteration. This refinement contains three steps:

RR.1 Computing the last column of the Rayleigh-quotient matrix H (where $H = V^H * A * V$ is symmetric/hermitian);
RR.2 Computing an eigen-decomposition of H (denoted as $HY = YD$);
RR.3 Refining the basis as $V \leftarrow V * Y$.

The last two steps can be expensive when the size of H becomes too large, which is the case when k_{want} is large and only standard restart is applied (in which case $dim_{max} \approx 2 * k_{want}$ for Lanczos-type methods).

The *inner-restart* tries to reduce the work required by the Rayleigh–Ritz refinement. We do not wait until the size of H exceeds dim_{max} to perform the standard restart. Instead, we restart much earlier than when dim_{max} is reached, and progressively refine the basis during the process.

The inner-restart is essentially a standard restart restricted to the active subspace V_{act}. That is, when act_{max} is reached, the last few columns in the window are truncated. Since the truncation is after a Rayleigh–Ritz refinement step, the columns in the window have been ordered so that the last few columns are least important, and the inner-restart keeps the current best non-convergent vectors in the active subspace.

Since we always work with the active block V_{act}, the Rayleigh–Ritz steps only need to be performed on Rayleigh-quotient matrices of size not exceeding act_{max}. This reduces the cost both in eigen-decomposition (step RR.2) and in the basis refinement (step RR.3).

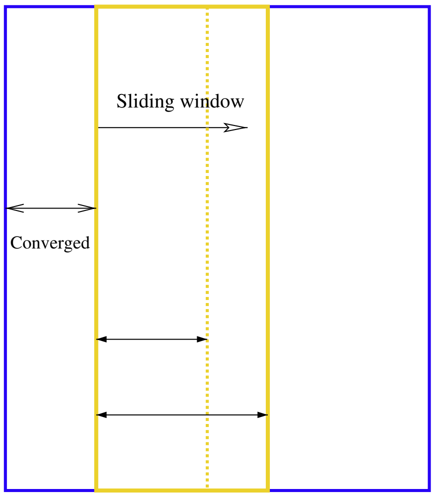

Fig. 1. Illustration of the inner–outer restart process: the "sliding window" corresponds to the active subspace. The inner-restart means truncating the active subspace into a smaller subspace after a Rayleigh–Ritz refinement step, whenever the maximum dimension of the active subspace is reached. This is done by keeping the left part of the active subspace until the dotted line.

Fig. 1 illustrates the inner–outer restart process. We work with the active block V_{act}, the maximum size of V_{act} is controlled by act_{max} that is usually much smaller than k_{want}. The active block acts like a window sliding through V: during the process the size of the window is reduced when it reaches act_{max}, and the window slides forward only when one or more new basis vectors converge. The converged eigenvectors are always deflated to the beginning columns of V, and the window contains only the non-converged vectors in V. The basis vectors inside the window are progressively refined during this inner-restart process until it reaches the end of V, after which we apply outer-restart if necessary.

The second observation is that a larger dim_{max} may incur more reorthogonalization cost at each iteration. That is, if we perform restart only when dim_{max} is reached, then more non-converged basis vectors are involved in the reorthogonalization per iteration. But if we perform inner-restart, then the number of non-converged vectors involved in the reorthogonalization per iteration is less than act_{max}. Note that orthogonalization cost can be quite significant since orthogonalizing k vectors is of $O(nk^2)$ complexity, it can easily be of $O(n^2)$ if $k \approx \sqrt{n}$.

The price paid for the reduced refinement and reorthogonalization cost per iteration is that the inner–outer restart normally requires more iterations to reach convergence. However, the total cputime for the inner–outer restart approach can be much smaller than that of only using standard restart.

The inner–outer restart technique mainly works within a low dimensional subspace V_{act}. Therefore this technique may be inefficient inside a standard Lanczos-type method, because low subspace dimension corresponds to low degree of the (implicit) Lanczos polynomial filters, which can lead to slow convergence. It is the suitably constructed Chebyshev filters inside a Davidson-type subspace method that makes this technique efficient.

We point out that the inner–outer restart is closely related to the locking associated with Davidson-type subspace iterative methods such as [12,40], since all these approaches perform restarts within an active subspace with restricted dimension. More precisely, implementations of existing locking and restart methods use two separate matrices, one to store the deflated converged eigenvectors (denoted as Q, which is of size k_{want}), the other to store the basis of the active subspace (denoted as V, of size act_{max}). Therefore the main memory used is $dim_{max} = k_{want} + act_{max}$.

The inner–outer restart may be considered as an extension to the existing locking + restart methods in two ways: Firstly, we do not separate Q from V, which waives memory copy from V to Q each time a basis vector is converged. Moreover, separating Q and V implies that the memory occupied by the size-k_{want} matrix Q does not participate in the subspace refinement and truncation, all essential calculations are performed on V, therefore the dimension of V usually cannot be small. In our approach as seen in Fig. 1, we can use all vectors (except the converged ones) for subspace iteration purpose. For example, at the early stage of iteration we can use a suitably large window size for inner iteration (and this size can be independent of act_{max}), at the end of iteration the total number of vectors used for iteration is $act_{max} + k_{want} - n_c$ where n_c denotes the number of vectors that have converged. Hence in theory inner–outer restart may converge fairly rapidly with $dim_{max} = k_{want} + 20$, while existing locking + restart implementations could have difficulty converging hundreds or thousands of eigenpairs with an active subspace of dimension only 20. Secondly, the inner–outer restart approach works for the general case $dim_{max} = k_{want} + c$, where c is a positive integer, e.g. $c = \lceil \frac{act_{max}}{2} \rceil$ if act_{max} is relatively large.

Now we address the situation where a large number of good initial vectors are available. This scenario arises in a number of specific applications. It appears that very few existing block methods were designed for efficiently exploiting this situation. The standard way for existing block methods to employ all available initial vectors is to use a block size large enough to include all initials. This approach usually is not practical if thousands or more good initial vectors are available, since it is well-known that too large block sizes lead to huge memory demand but need not necessarily improve cache efficiency.

Here we introduce a *progressive filtering* technique that can conveniently utilize any number of available initial vectors. We exploit an advantage of a Davidson-type method over Lanczos-type methods, namely that no specific Krylov structures need to be kept during the iteration. In Chebyshev–Davidson method, we can include new appropriate vectors into the projection basis whenever necessary. A distinctive feature of the progressive filtering is that a large block size is not needed, but we can still deploy all available initial vectors. The choice of block size can be the same as what has been practiced in standard block methods, such as choosing a block size between 3 and 30.

Arguably, when the number of available initial vectors is as large as k_{want}, the fixed dimension subspace iteration method may be a best choice. The idea of our progressive filtering is a natural extension of this subspace iteration: instead of doing subspace iteration on the full available initial vectors at once, we progressively iterate over blocks of the initials vectors. Essentially we perform a Chebyshev accelerated variable-dimension subspace iteration, for which all available initial vectors will be progressively utilized. One advantage of this approach is that it has freedom to augment potentially better new basis vectors during the iteration process, instead of only resorting to the initials.

The implementation of the progressive filtering is quite simple. Assume that the initial vectors are ordered so that their corresponding Rayleigh-quotients are in non-decreasing order (this is easy to realize in practice), we progressively filter all initial vectors in the natural order, i.e., from first to the last. First, we start the Chebyshev–Davidson iteration by filtering the first k_b number of the initial vectors, where k_b denotes a block size suitable for any standard block methods (e.g. $3 \leqslant k_b \leqslant 30$). Then we progressively filter the remaining initial vectors and include the filtered vectors in the projection basis as follows: Denote the number of eigenvalues that are converged at the current iteration step as e_c, (here e_c does not count the converged eigenvalues from previous step), at the next iteration we filter e_c number of the leftmost unused initial vectors together with $k_b - e_c$ number of the current best non-converged Ritz vectors, then augment the k_b filtered vectors into the projection basis. That is, we replace the just converged and deflated basis vectors by the same number of unused initial vectors. Clearly, if no eigenvalues are converged at an iteration step, then we filter k_b current best non-converged Ritz vectors, same as in the standard block Chebyshev–Davidson with no good initial vectors available. The process continues until all the initial vectors are progressively filtered, after this stage, we apply the standard block Chebyshev–Davidson iteration until all k_{want} eigenpairs are converged.

Algorithm 3.1. contains the pseudo-code for the block Chebyshev–Davidson subspace iteration with inner–outer restart for computing k_{want} smallest eigenpairs. Matlab syntax is used to denote matrices and function calls. For simplicity, we only present the main structure of the algorithm and neglect several less essential details.

In Algorithm 3.1, k_b denotes the block size; k_{sub} counts the dimension of the projection subspace V, where $k_{sub} \leqslant dim_{max}$; k_c counts the number of converged eigenpairs, and $V(:, 1{:}k_c)$ always contains the converged and deflated eigenvectors; the active projection subspace is contained in $V(:, k_c + 1{:}k_{sub})$.

The design of Algorithm 3.1 uses a systematic indexing for the basis vectors in V. The key component of the indexing is what we called the *two-indexes trick*: We use two indexes, k_{sub} and the dimension of the active subspace k_{act}, which are required to satisfy $k_{act} + k_c = k_{sub}$ throughout the iterations. By this design, deflation and restarts can be achieved simply by adjusting indexes. For example, deflation (also called "locking") is conveniently realized by not using any $V(:, 1{:}k_c)$ for projection when computing H, and by keeping all the basis vectors in $V(:, 1{:}k_{sub})$ orthonormal; and restarts are conveniently realized as in *Step iv.6* and *Step iv.12*. Both deflation and restarts require no additional memory copies.

Each iteration adds k_b vectors to the projection basis at *Step iv.2*. Although we use a fixed block size, the design of this algorithm, which is mainly controlled by two indexes k_{sub} and k_{act}, easily allows one to use varying block sizes if necessary. (That is, the k_b at *Steps iv.1* and *iv.2* can be different from the k_b at previous steps, which provides freedom in choosing how many vectors to filter at *Step iv.1*. For example, one can use the last two size-k_b blocks from Algorithm 2.1 to augment the subspace instead of using only the last size-k_b block. This may be useful when m is relatively large and the starting k_b is too small, but we do not pursue this further in this paper.)

Input variables to Algorithm 3.1 include k_{want} and a matrix or a subroutine for matrix–vector products. Optional parameters include m the Chebyshev polynomial degree, dim_{max} the largest dimension of the subspace, k_b the block size, act_{max} the maximum dimension of the active subspace, k_{ro} and k_{ri} two integers controlling respectively number of vectors to keep during the outer and the inner-restarts, τ the convergence tolerance, and V_{init} which contains the size-k_{init} non-increasingly ordered (according to Rayleigh-quotients) initial vectors. The output variables are the converged eigenvalues $eval(1{:}k_c)$ and corresponding converged eigenvectors $V(:, 1{:}k_c)$, where k_c is the number of converged eigenpairs.

Algorithm 3.1. Block Chebyshev–Davidson algorithm with inner–outer restart.

```
(i) Compute upperb via Algorithm 2.2, set lowb = $(3\lambda_{min}(T) + \lambda_{max}(T))/4$.
(ii) Set $V_{tmp} = V_{init}(:, 1{:}k_b)$, (construct $k_b - k_{init}$ random vectors if $k_{init} < k_b$); set $k_i = k_b$ ($k_i$ counts # of used vectors in $V_{init}$).
(iii) Set $k_c = 0$, $k_{sub} = 0$ ($k_{sub}$ counts the dimension of the current subspace); set $k_{act} = 0$ ($k_{act}$ counts the dimension of the
      active subspace).
(iv) Loop: while itmax is not exceeded
    1. $[V(:, k_{sub} + 1{:}k_{sub} + k_b)]$ = Chebyshev_filter($V_{tmp}$, $m$, lowb, upperb).
    2. Orthonormalize $V(:, k_{sub} + 1{:}k_{sub} + k_b)$ against $V(:, 1{:}k_{sub})$.
    3. Compute $W(:, k_{act} + 1{:}k_{act} + k_b) = AV(:, k_{sub} + 1{:}k_{sub} + k_b)$; set $k_{act} = k_{act} + k_b$; set $k_{sub} = k_{sub} + k_b$.
```

```
 4. Compute the last $k_b$ columns of the Rayleigh-quotient matrix $H$:
    $H(1{:}k_{act}, k_{act}-k_b+1{:}k_{act}) = V(:, k_c+1{:}k_{sub})'W(:, k_{act}-k_b+1{:}k_{act})$, then symmetrize $H$.
 5. Compute eigen-decomposition of $H(1{:}k_{act}, 1{:}k_{act})$ as $HY = YD$, where $diag(D)$ is in non-increasing order. Set $k_{old} = k_{act}$.
 6. If $k_{act} + k_b > act_{max}$, then do inner restart as: $k_{act} = k_{ri}$, $k_{sub} = k_{act} + k_c$.
 7. Do subspace rotation (final step of Rayleigh–Ritz refinement) as: $V(:, k_c+1{:}k_c+k_{act}) = V(:, k_c+1{:}k_{old})Y(1{:}k_{old}, 1{:}k_{act})$,
    $W(:, 1{:}k_{act}) = W(:, 1{:}k_{old})Y(1{:}k_{old}, 1{:}k_{act})$.
 8. Test for convergence of the $k_b$ vectors in $V(:, k_c+1, k_c+k_b)$, denote number of newly converged Ritz pairs at this step as $e_c$. If $e_c > 0$, then update $k_c = k_c + e_c$, save converged Ritz values in $eval$ (sort $eval(:)$ in non-increasing order), and deflate/lock converged Ritz vectors (only need to sort $V(:, 1{:}k_c)$ according to $eval(1{:}k_c)$).
 9. If $k_c \geqslant k_{want}$, then return $eval(1{:}k_c)$ and $V(:, 1{:}k_c)$, exit.
10. If $e_c > 0$, set $W(:, 1{:}k_{act}-e_c) = W(:, e_c+1{:}k_{act})$, $k_{act} = k_{act} - e_c$.
11. Update $H$ as the diagonal matrix containing non-converged Ritz values $H = D(e_c+1{:}e_c+k_{act}, e_c+1{:}e_c+k_{act})$.
12. If $k_{sub} + k_b > dim_{max}$, do outer restart as: $k_{sub} = k_c + k_{ro}$, $k_{act} = k_{ro}$.
13. Get new vectors for the next filtering:
    Set $V_{tmp} = [V_{init}(k_i+1{:}k_i+e_c), V(:, k_c+1{:}k_c+k_b-e_c)]$; $k_i = k_i + e_c$;
14. Set lowb as the median of non-converged Ritz value in $D$.
```

Remarks on Algorithm 3.1: (i) the convergence test at *step 8* should quit testing whenever the first non-convergence is detected; (ii) the default values of the optional parameters act_{max}, k_{ro} and k_{ri} can be readily determined by k_{want}, k_b and the matrix dimension; e.g. the k_{ro} and k_{ri} usually need not be inputed, they default to values related to other parameters such as $k_{ri} = \max(\lfloor \frac{act_{max}}{2} \rfloor, \; act_{max} - 3k_b)$ and $k_{ro} = dim_{max} - 2k_b - k_c$; (iii) the orthogonalization at *step 2* is performed by the DGKS method [41]; random vectors are used to replace any vectors in $V(:, k_{sub}+1{:}k_{sub}+k_b)$ that may become numerically linearly dependent to the current basis vectors in V; (iv) When no initial vectors are available, simply set $k_{init} = 0$. In this case the algorithm will start from random initial vectors; Algorithm 3.1 can also conveniently address $0 \leqslant k_{init} \leqslant k_{want}$, where k_{init} can be 0 or as large as k_{want}, using a standard block size k_b.

4. Analysis

Denote the center and half-width of the interval to be dampened at the jth step of Algorithm 3.1 as c_j and e_j, respectively, then the Chebyshev polynomial applied at the jth step can be expressed as

$$p_m^{(j)}(A) = C_m^{(j)}((A - c_j I)/e_j).$$

In the following we discuss convergence property of Algorithm 3.1 under some simplified assumptions. Assume that dim_{max} and act_{max} are large enough so that no restart is necessary in Algorithm 3.1. Denote the linearly independent k_b starting vectors as V_1. Then, the subspaces generated by Algorithm 3.1 at the first three steps can be written respectively as

$$K_1 = \mathrm{span}\{p_m^{(1)}(A)V_1\}, \quad K_2 = \mathrm{span}\{K_1, p_m^{(2)}(A)K_1\},$$
$$K_3 = \mathrm{span}\{K_2, \alpha p_m^{(3)}(A)K_1 + \beta p_m^{(3)}(A)p_m^{(2)}(A)p_m^{(1)}(A)V_1\},$$

where the filter is designed to make $\beta \neq 0$ if the previous subspace can be augmented. Extending the above notation to the jth step, we see that the last term in the subspace K_j generated by the simplified block Chebyshev–Davidson method contains

$$\Phi_j(A)V_1, \tag{5}$$

where

$$\Phi_j(A) = \prod_{l=1}^{j} p_m^{(l)}(A).$$

The expression (5) shows that the subspace generated by the simplified block Chebyshev–Davidson method always includes the vectors that are generated by an accelerated fixed dimension subspace iteration applied to V_1. Therefore, applying the results in [42,43] for subspace iteration, and by the property of Rayleigh–Ritz refinement from subspace acceleration, we see the convergence rate to the wanted eigensubspace $\mathcal{V}_1$ (corresponding to $\lambda_1, \ldots, \lambda_{k_b}$) can be bounded as

$$dist(\Phi_j(A)V_1, \mathcal{V}_1) \leqslant C \frac{\max_{l>k_b} |\Phi_j(\lambda_l)|}{\min_{l \leqslant k_b} |\Phi_j(\lambda_l)|}, \tag{6}$$

where $\sigma(A)$ is assumed to be in non-increasing order, and we also assume the standard gap condition ($\lambda_{k_b} < \lambda_{k_b+1}$). The constant C is proportional to

$$\frac{dist(V_1, \mathcal{V}_1)}{\sqrt{1 - dist(V_1, \mathcal{V}_1)^2}}.$$

The bound (6) can be pessimistic since it mainly uses the last term associated with a fixed dimension subspace iteration, but (6) serves to explain two cases where the convergence can be fast. The first case is a small C, i.e., when the canonical angle between V_1 and $\mathcal{V}_1$ is small; this provides the reason why good initial vectors should be used when they are available. The second case is a small $\frac{\max_{l>k_b}|\Phi_j(\lambda_l)|}{\min_{l\leqslant k_b}|\Phi_j(\lambda_l)|}$, the Chebyshev filtering is designed to quickly reduce the value of this term.

Convergence rate for the restarted method is more tedious to establish. Intuitively, since Rayleigh–Ritz refinement combined with restart always keeps the current best Ritz vectors in the subspace, and the filters are designed to constantly improve the current subspace towards the wanted eigenspace, convergence is in practice not a problem. However, it is well-known that restart may slow down the convergence speed, this is one side effect of using limited memory.

5. Numerical results

We present numerical results of Algorithm 3.1 executed without any simplification. Four Hamiltonian matrices from density functional theory (DFT) electronic-structure calculations are used for the numerical experiments.[1] These matrices are symmetric and indefinite. Table 1 lists some statistics of each matrix.

These matrices are constructed from solving the Kohn–Sham equation

$$A\Psi = \left(\frac{-\hbar^2}{2m_e}\nabla^2 + V_{eff}(x,y,z)\right)\Psi = \lambda_E\Psi, \tag{7}$$

in density functional theory [45,46]. We use real-space pseudopotential method via a high order finite difference scheme [47,48]. In (7), λ_E is the *energy eigenvalue*, $\Psi(x, y, z)$ is the *wave function* also called the *eigenfunction*, V_{eff} is the effective system *potential*, m_e is the electron mass, and $\hbar$ is the *Planck's constant*.

The problem is set up inside a spherical region of radius ϑ centered at the origin. The eigenfunctions satisfy Dirichlet boundary conditions on the sphere and they vanish outside the spherical region. That is, $\Psi(x, y, z) = 0$ for any $\sqrt{x^2+y^2+z^2} \geqslant \vartheta$. An uniform grid is laid on a cubical region containing the sphere. Fig. 2 illustrates this grid, where $\Theta = \{(x, y, z) : |x|, |y|, |z| \leqslant \vartheta\}$ is the region containing the sphere. A 12th order centered finite difference scheme on the uniform grid laid on the cubical region is used to discretize the Laplacian.

The potential term V_{eff} contains a local component (diagonal matrix) added to the discretized Laplacian. In addition, there are nonlocal components around each atom which correspond to adding low-rank matrices to the Laplacian [47,48]. Since the potential V_{eff} depends on the wave functions Ψ, the eigenproblem (7) is nonlinear and is often solved via a *self-consistent* loop. Each matrix in Table 1 corresponds to the final Hamiltonian for which self-consistency has been reached.

In DFT calculations the number of wanted eigenpairs corresponds to the number of occupied states, which is proportional to the number of valence electrons in a material system. Therefore, the more valence electrons, the more smallest eigenvalues one need to compute. And the corresponding eigenvectors are used to compute charge density which is needed for updating V_{eff}. Here for simplicity of numerical comparisons in Matlab, we do not restrict k_{want} to be the number of occupied states.

We implement Algorithm 3.1 (denoted as "chebdav") in Matlab and compare with two representative codes: JDCG[2] [40] and LOBPCG[3] [49]. JDCG is a Davidson-type method based on JDQR [50]. It takes symmetry into account and therefore is more efficient than JDQR for symmetric eigenproblems. LOBPCG is a block preconditioned eigensolver. A block Krylov-type method IRBL [51,52] was also used for comparison but was found to be less competitive when k_{want} becomes relatively large. Therefore we only report comparisons of chebdav with JDCG and LOBPCG.

Fig. 3 shows CPU time comparisons computing $k_{want} = 150$ smallest eigenvalues and corresponding eigenvectors for the four matrices listed in Table 1. We compare the common case where no good initial vectors are available. To address this case, we simply set $k_{init} = 0$ for chebdav. For simplicity, we use a fixed polynomial degree $m = 25$, a fixed block size $k_b = 6$, and a fixed $act_{max} = 60$, for all computations by chebdav in Fig. 3. The convergence tolerance is set to 10^{-10} for all methods. The memory usage of chebdav and JDCG is comparable, with the maximum subspace dimension set to 210 and the active subspace dimension set to 60 for both methods. These choices of dimensions give close to best performance for JDCG for the targeted problems in Fig. 3, much better than using the default values of JDCG which are mainly for small k_{want} problems. LOBPCG uses about three times larger memory, since it uses a block size equal to k_{want} and the active subspace contains three such blocks. Note that JDCG is not a block method, we set $ones(n, 1)$ as its initial; the initials for chebdav are $ones(n, 1)$ augmented by $k_b - 1 = 5$ random vectors; while the initials of LOBPCG are $ones(n, 1)$ augmented by $k_{want} - 1 = 149$ random vectors. (We note that repeated calling to LOBPCG with a smaller block size and with locking took much longer to converge than the reported time in Fig. 3.) The speedup rate of chebdav over JDCG and LOBPCG as shown in Fig. 3 is representative for $k_{want} \geqslant 100$, as observed from rather extensive test runs on various type of hermitian matrices.

To test the efficiency of the progressive filtering of available initial vectors, we use the 150 eigenvectors computed by JDCG (as in Fig. 3) and perturbed them by $10^{-\gamma} * rand(n, 150)$ as initials to chebdav and LOBPCG. Table 2 shows comparison

[1] Matrices are available from the University of Florida Sparse Matrix Collection [44] under group name PARSEC.
[2] http://mntek3.ulb.ac.be/pub/docs/jdcg/.
[3] http://www.mathworks.com/matlabcentral/fileexchange/48-lobpcg-m.

Table 1
Dimension and number of non-zeros (nnz) of four test matrices. The materials correspond to a water molecule, a silicon dioxide and two hydrogen passivated germanium and silicon quantum dots. The subindex of each atom name shows the number of the corresponding atom.

Matrix name	H_2O	SiO_2	$Ge_{99}H_{100}$	$Si_{87}H_{76}$
Dimension	67,024	155,331	112,985	240,369
nnz	2,216,736	11,283,503	8,451,395	10,661,631

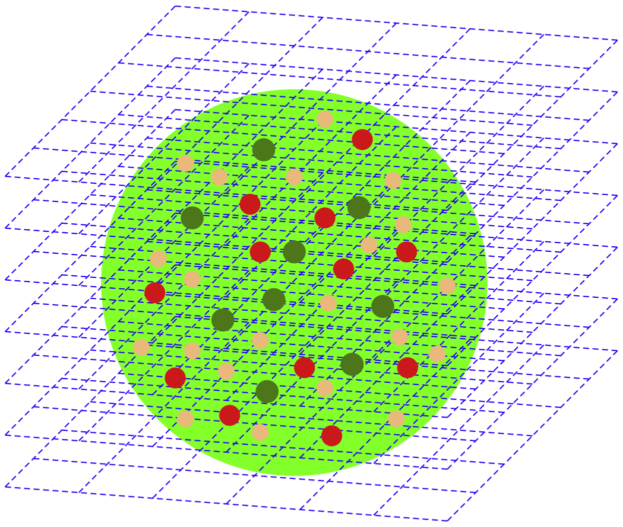

Fig. 2. A typical configuration using an uniform grid for electronic structure calculation of a localized system. The region inside the outer sphere represents the domain where the wave functions are allowed to be nonzero. The small dots inside the sphere represent atoms [48]. The actual grid is much finer than the one shown in this figure.

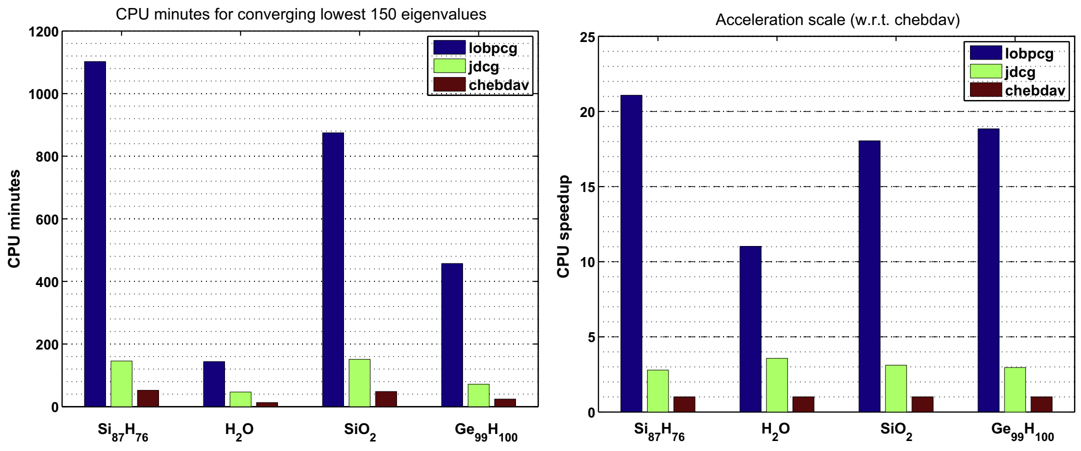

Fig. 3. CPU time comparisons of chebdav, JDCG and LOBPCG for converging 150 eigenpairs of the matrices in Table 1. The second figure plots the CPU time of LOBPCG and JDCG over that of chebdav, which shows that chebdav is an order of magnitude faster than LOBPCG, and about three times faster than JDCG.

Table 2
CPU minutes comparison using the same 150 initial vectors. Chebdav uses $k_b = 10$, with degree $m = 20$. The block size for LOBPCG is 150. The total memory used by LOBPCG is about three times more than that of chebdav.

Matrix	$Si_{87}H_{76}$	H_2O	SiO_2	$Ge_{99}H_{100}$
Chebdav (10^{-6})	45.08	9.54	37.25	18.50
LOBPCG (10^{-6})	196.33	92.07	266.50	136.47
Chebdav (10^{-8})	26.18	5.91	10.94	10.98
LOBPCG (10^{-8})	116.53	57.68	146.47	52.00

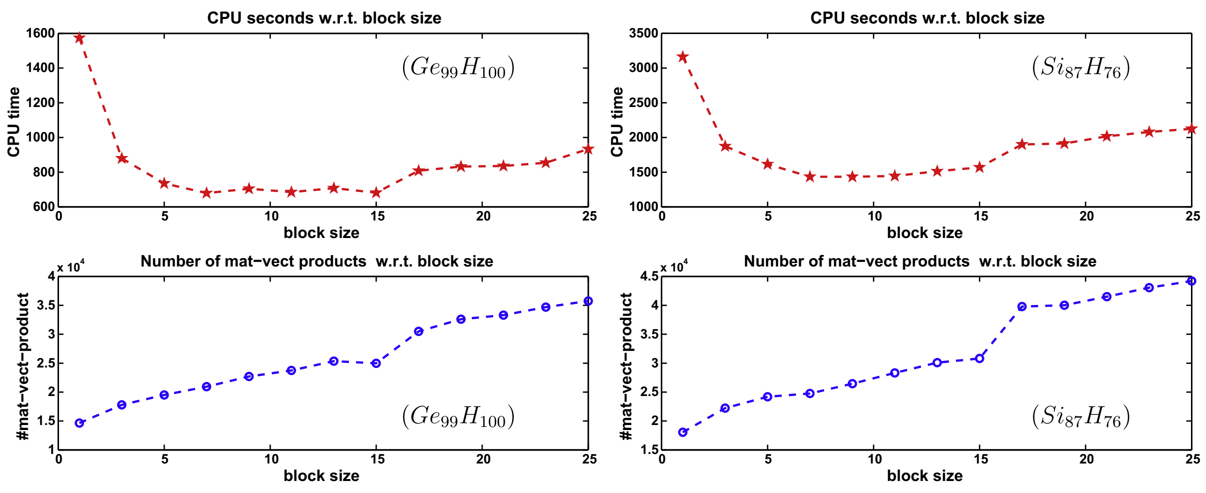

Fig. 4. The effect of block sizes on converging 100 smallest eigenpairs for $Ge_{99}H_{100}$ (left) and $Si_{87}H_{76}$ (right), with other parameters fixed ($m = 25$, $act_{max} = 60$, $dim_{max} = 150$).

converging the smallest 150 eigenpairs for $\gamma = 6$ and 8. Since JDCG is non-block, we only compare with LOBPCG in Table 2. With LOBPCG using a block size 150, we are essentially comparing with an accelerated subspace iteration. The convergence tolerance is set to 10^{-10}. Table 2 shows that the progressive filtering can effectively utilize all available initial vectors without using a large block size. However, comparing the data in Table 2 with Fig. 3 we also notice that LOBPCG achieves faster acceleration utilizing good initial vectors: Table 2 shows that chebdav is on average only around five times faster than LOBPCG. This may be attributed to the fact that subspace iteration type methods are particularly good for utilizing many initial vectors.

To examining the effect of block sizes, we input different k_b to Algorithm 3.1 but fix all other parameters. Fig. 4 shows two representative results with k_b varying from 1 to 25 with a stride length 2. Clearly, the block version is consistently more efficient than the non-block version ($k_b = 1$). We also see that the number of matrix–vector products does not determine the total CPU time for convergence. From Fig. 4 we see that $k_b > 15$ is less efficient than k_b around 10. This is mainly because that we use a relatively small $act_{max} = 60$, which leads to more frequent inner-restarts for $k_b > 15$ than for $k_b \approx 10$. (*Remark*: While not reported here, we observed better efficiency for $k_b = 25$ if we use larger act_{max}, e.g. $act_{max} = 120$.)

For the comparisons in Fig. 4 we simply set the polynomial degree as $m = 25$. To test the effect of different polynomial degrees, we fix all other parameters but input m (where m varies from 15 to 40 with a stride length 2) to Algorithm 3.1. Fig. 5 lists two representative examples for the Hamiltonians from DFT calculations. As seen from Fig. 5, the main trend is that increasing m appears to improve the efficiency. This is not too surprising since the Hamiltonians have quite clustered eigenvalues. But it is reasonable to expect that when m becomes too large it will be counterproductive.

Another important parameter related to inner-restart is act_{max}. As mentioned earlier, our intuition is that if the dimension of the active subspace is too large, then it can incur more cost related to orthogonalization and basis refinement. In this case, although the iteration can converge in less steps costing less matrix–vector products, it can take longer CPU time than using an active subspace with suitable dimension. Fig. 6 confirms this intuition. In Fig. 6 we intensionally set dim_{max} at a high value so that outer-restart becomes not necessary (since convergence is achieved before dim_{max} is reached). The observed difference is mainly related to act_{max}, which takes value from 20 to 215 with a stride length 15. The k_b is intensionally set to a small 3 so that $act_{max} > 140$ may be considered too large relative to this block size. Fig. 6 also shows that if act_{max} is too small, then it causes too frequent inner-restarts and takes far more iteration steps (hence longer CPU time) to converge.

The convergence tolerance used in Figs. 4–6 is 10^{-10}. The results in these figures demonstrate that Algorithm 3.1 is robust and works conveniently with different parameters. Generally, users only need to specify $5 \leqslant k_b \leqslant 15$, $15 \leqslant m \leqslant 35$, and act_{max} around $10k_b$. These parameters combined with the default values set in our code for the remaining parameters should generally lead to quite decent efficiency for Algorithm 3.1.

All the calculations in Matlab were performed on a Dell-R710 computer with two quad-core Intel X5550 Xeon CPU (clock speed 2.66 GHz) and 144 Gb RAM at the SMU HPC center.

We also implemented Algorithm 3.1 in Fortran95 and integrated it into the DFT package called PARSEC.[4] PARSEC, evolved since early 90's, is now one of the major real-space pseudopotential DFT packages. Previously PARSEC has three diagonalization solvers: A preconditioned block Davidson method called Diagla [53,54], the symmetric eigensolver from ARPACK [7,36] and the Thick-Restart Lanczos method (TRLanc) [55,8]. All of the four solvers utilize MPI for parallel functionality.

[4] http://parsec.ices.utexas.edu/.

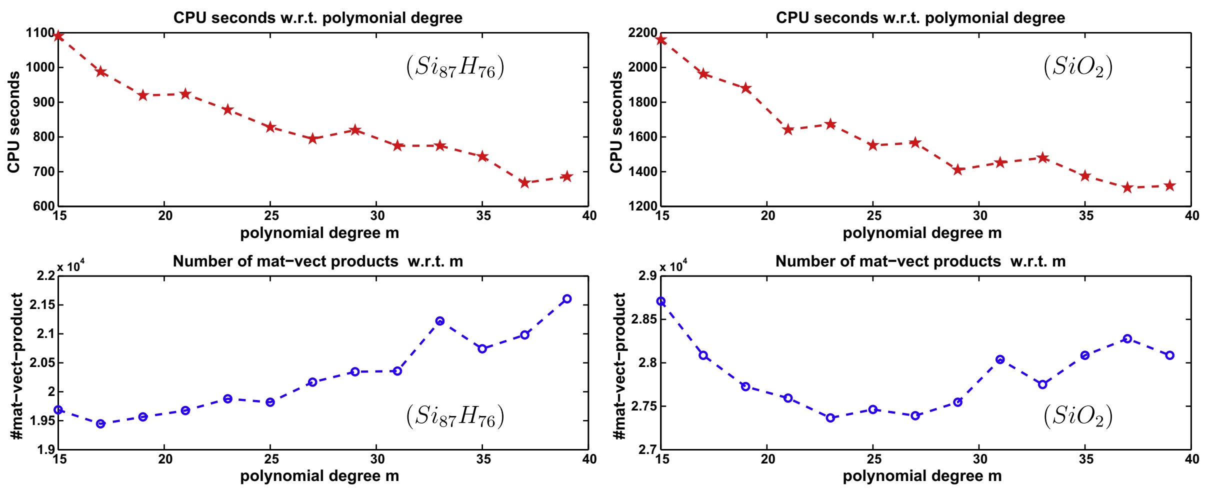

Fig. 5. The effect of polynomial degree on converging 100 smallest eigenpairs for $Si_{87}H_{76}$ and (left) and SiO_2 (right), with other parameters fixed ($k_b = 6$, $act_{max} = 60$, $dim_{max} = 150$).

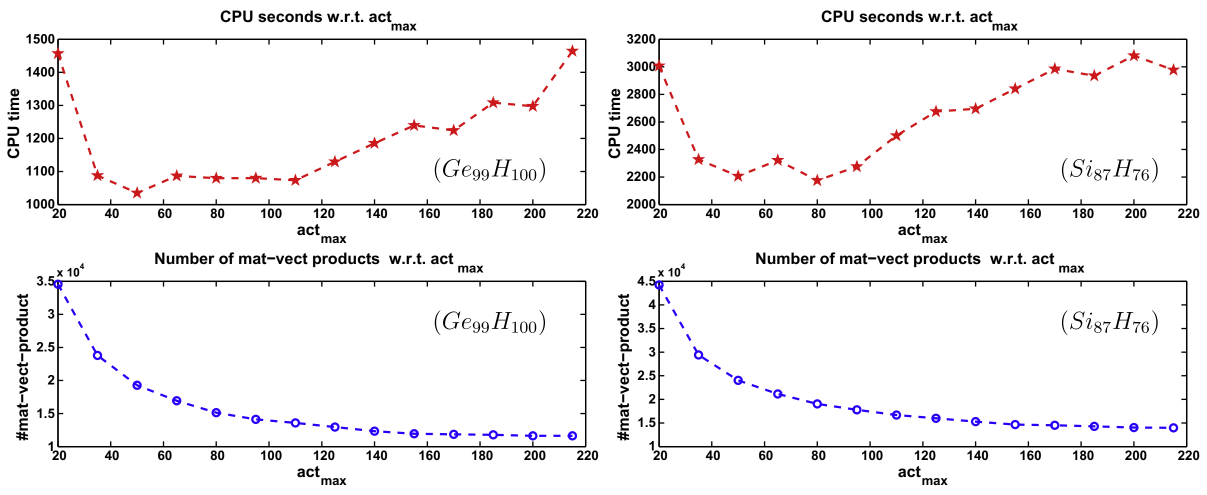

Fig. 6. The effect of the act_{max} size on converging 100 smallest eigenpairs for $Ge_{99}H_{100}$ (left) and $Si_{87}H_{76}$ (right), with other parameters fixed ($k_b = 3$, $m = 25$, $dim_{max} = 250$).

ARPACK is one of the most robust and efficient general purpose eigensolvers, especially for nonsymmetric eigenproblems. It also serves as a standard benchmarking tool for sparse eigenvalue calculations (e.g. [56]). TRLanc is specifically for sparse symmetric eigenproblems, for which more efficient restarting schemes can be employed.

Table 3 shows comparison of Algorithm 3.1 with ARPACK and TRLanc on a relatively large silicon quantum dot $Si_{4001}H_{1012}$. This quantum dot contains 4001 silicon atoms with surface passivation by 1012 hydrogen atoms. The Hamiltonian size is 1,472,440, which reduces to 368,110 by exploiting 4 symmetric group operations. The number of eigenpairs needed is 8524. Diagla is not listed in Table 3 since it did not finish computation within the given 20-h CPU limit. All source codes

Table 3
Comparison of three eigensolvers on a silicon quantum dot $Si_{4001}H_{1012}$, for which $k_{want} = 8524$. Each solver runs paralelly using 128 SunBlade x6420 cores (CPU clock at 2.3 GHz) on Ranger. Major parameters used for chebdav are $k_b = 5$, $m = 20$, $act_{max} = 400$ and $dim_{max} = k_{want} + act_{max} = 8924$. While for ARPACK and TRLanc, $dim_{max} = 2k_{want} = 17{,}048$.

Solver	Chebdav	TRLanc	ARPACK
#Mat-Vect products	830,530	109,664	108,942
CPU seconds	9361.89	27,266.28	47,872.40

are compiled using PGI compiler with same optimization flags. For BLAS/LAPACK calls we use the system optimized AMD Core Math library (ACML). The calculations were performed on the TeraGrid supercomputer named Ranger at the Texas Advanced Computing Center, using 128 SunBlade x6420 (AMD Opteron) cores.

For the results in Table 3, chebdav used none good initial vectors, it starts with an initial block containing four random vectors and the same initial vector as for ARPACK and TRLanc. From the Matlab experiments we see that the parameters used for chebdav in Table 3 are likely suboptimal. However, even with these parameters, chebdav is several times faster than ARPACK and TRLanc, and it uses only about half the memory. We again see that the number of matrix–vector products is not the only factor that determines CPU time. TRLanc costs slightly more matrix–vector products than ARPACK, but it is close to twice faster than ARPACK due to its improved restarting scheme. While chebdav costs the most matrix–vector products, the efficiency gained through these products by the targeted Chebyshev filtering significantly offset the cost elsewhere, such as reorthogonalization and subspace refinement. Moreover, chebdav mainly depends on matrix–vector products, therefore it scales better than ARPACK and TRLanc which involve more inner-product operations.

Due to its robustness and efficiency, our block Chebyshev–Davidson is adopted as the default eigensolver for the first step diagonalization in PARSEC. Using Algorithm 3.1 as the first step diagonalization solver, PARSEC has successfully solved a series of challenging problems in density functional calculations, including Hamiltonians with dimension about three million, where more than 19,000 eigenpairs need to be computed. Interested readers may refer to [57–59].

6. Conclusions

We propose a block Chebyshev–Davidson algorithm with inner–outer restart for large hermitian/symmetric eigenvalue problem. The essential component is an adaptive procedure for constructing efficient block Chebyshev filters. This procedure is conveniently realized by adapting the filtering bounds at each iteration to properly magnify the wanted portion of the spectrum. An inner–outer restart technique is also employed to reduce the subspace dimension, thus effectively reduces the computational cost related to using an unnecessarily large subspace in a Davidson-type subspace method. We also employ a progressive filtering technique which enables the full employment of a large number of good initial vectors if they are available, without using a too large block size. Numerical experiments are performed on several Hamiltonian matrices from density functional theory calculations in material science. Comparisons with several well-known eigenvalue packages confirm the remarkable efficiency and robustness of the block Chebyshev–Davidson algorithm.

Acknowledgments

The author thanks Yousef Saad for helpful comments and suggestions on this paper. He thanks Ren-cang Li and Andreas Stathopoulos for useful discussions. He is very grateful to one referee who provided many constructive suggestions and comments which improved the paper.

References

[1] B.N. Parlett, The Symmetric Eigenvalue Problem, Classics in Applied Mathematics, vol. 20, SIAM, Philadelphia, PA, 1998.
[2] G.W. Stewart, Matrix Algorithms II: Eigensystems, SIAM, Philadelphia, 2001.
[3] Y. Saad, Numerical Methods for Large Eigenvalue Problems, John Wiley, New York, 1992. <http://www.cs.umn.edu/~saad/books.html> .
[4] G.H. Golub, C.F.V. Loan, Matrix Computations, third ed., Johns Hopkins University Press, Baltimore, MD, 1996.
[5] Z. Bai, J. Demmel, J. Dongarra, A. Ruhe, H. van der Vorst (Eds.), Templates for the Solution of Algebraic Eigenvalue Problems, SIAM, Philadelphia, PA, 2000.
[6] C. Lanczos, An iteration method for the solution of the eigenvalue problem of linear differential and integral operators, J. Res. Nat. Bur. Standards 45 (1950) 255–282.
[7] D.C. Sorensen, Implicit application of polynomial filters in a *k*-step Arnoldi method, SIAM J. Matrix Anal. Appl. 13 (1992) 357–385.
[8] K. Wu, H. Simon, Thick-restart Lanczos method for large symmetric eigenvalue problems, SIAM J. Matrix Anal. Appl. 22 (2000) 602–616.
[9] G.W. Stewart, A Krylov–Schur algorithm for large eigenproblems, SIAM J. Matrix Anal. Appl. 23 (2001) 601–614.
[10] E.R. Davidson, The iterative calculation of a few of the lowest eigenvalues and corresponding eigenvectors of large real-symmetric matrices, J. Comput. Phys. 17 (1975) 87–94.
[11] R.B. Morgan, D.S. Scott, Generalizations of Davidson's method for computing eigenvalues of sparse symmetric matrices, SIAM J. Sci. Stat. Comput. 7 (1986) 817–825.
[12] G.L.G. Sleijpen, H.A. van der Vorst, A Jacobi–Davidson iteration method for linear eigenvalue problems, SIAM J. Matrix Anal. Appl. 17 (1996) 401–425.
[13] D.C. Sorensen, C. Yang, A truncated rq iteration for large scale eigenvalue calculations, SIAM J. Matrix Anal. Appl. 19 (1998) 1045–1073.
[14] A. Stathopoulos, Nearly optimal preconditioned methods for hermitian eigenproblems under limited memory. Part I: Seeking one eigenvalue, SIAM J. Sci. Comput. 29 (2) (2007) 481–514.
[15] A. Stathopoulos, J.R. McCombs, Nearly optimal preconditioned methods for hermitian eigenproblems under limited memory. Part II: Seeking many eigenvalues, SIAM J. Sci. Comput. 29 (5) (2007) 2162–2188.
[16] A. Brandt, S. McCormick, J. Ruge, Multi-grid methods for differential eigenproblems, SIAM J. Sci. Stat. Comput. 4 (1983) 244–260.
[17] J. Mandel, S. McCormick, A multilevel variational method for $Au = \lambda Bu$ on composite grids, J. Comput. Phys. 80 (1989) 442–452.
[18] E.L. Briggs, D.J. Sullivan, J. Bernholc, Large-scale electronic-structure calculations with multigrid acceleration, Phys. Rev. B 52 (1995) R5471–5474.
[19] S. Costiner, S. Táasan, Adaptive multigrid techniques for large-scale eigenvalue problems: solutions of the Schrödinger problem in two and three dimensions, Phys. Rev. E 51 (1995) 3704–3717.
[20] O. Livne, A. Brandt, Multiscale eigenbasis calculations: N eigenfunctions in $O(N\log N)$, in: T. Barth, T. Chan, R. Haimes (Eds.), Multiscale and multiresolution methods, Lecture Notes in Computer Science Engineering, vol. 20, Springer-Verlag, 2002, pp. 347–357.
[21] C. Vömel, S.Z. Tomov, O.A. Marques, A. Canning, L.-W. Wang, J.J. Dongarra, State-of-the-art eigensolvers for electronic structure calculations of large scale nano-systems, J. Comput. Phys. 227 (15) (2008) 7113–7124.
[22] A. Szabo, N.S. Ostlund, Modern Quantum Chemistry: Introduction to Advanced Electronic Structure Theory, Dover Publications, New York, 1996.

[23] R. Parr, W. Yang, Density Functional Theory of Atoms and Molecules, Oxford Univ. Press, 1989.
[24] H. Rutishauser, Computational aspects of F.L. Bauer's simultaneous iteration method, Numer. Math. 13 (1969) 4–13.
[25] H. Rutishauser, Simultaneous iteration method for symmetric matrices, Numer. Math. 16 (1970) 205–223.
[26] H. Rutishauser, Simultaneous iteration method for symmetric matrices, in: J.H. Wilkinson, C. Reinsh (Eds.), Handbook for Automatic Computation (Linear Algebra), vol. II, Springer-Verlag, 1971, pp. 284–302.
[27] A. Ruhe, Implementation aspects of band Lanczos algorithms for computation of eigenvalues of large sparse symmetric matrices, Math. Comput. 33 (1979) 680–687.
[28] R.G. Grimes, J.G. Lewis, H.D. Simon, A shifted block Lanczos algorithm for solving sparse symmetric generalized eigenproblems, SIAM J. Matrix Anal. Appl. 15 (1994) 228–272.
[29] A.V. Knyazev, K. Neymeyr, Efficient solution of symmetric eigenvalue problems using multigrid preconditioners in the locally optimal block conjugate gradient method, Electron. Trans. Numer. Anal. 15 (2003) 38–55.
[30] Y. Zhou, Y. Saad, Block Krylov–Schur method for large symmetric eigenvalue problems, Numer. Alg. 47 (4) (2008) 341–359.
[31] Y. Zhou, Y. Saad, A Chebyshev–Davidson algorithm for large symmetric eigenvalue problems, SIAM J. Matrix Anal. Appl. 29 (3) (2007) 954–971.
[32] Y. Zhou, Studies on Jacobi–Davidson, Rayleigh quotient iteration, inverse iteration generalized Davidson and Newton updates, Numer. Linear Algebra Appl. 13 (8) (2006) 621–642.
[33] P.L. Chebyshev, Théorie des mécanismes connus sous le nom de parallélogrammes, Mém. Acad. Sci. Pétersb 7 (1854) 539–568.
[34] W.W. Rogosinski, Some elementary inequalities for polynomials, Math. Gaz. 39 (1955) 7–12.
[35] T.J. Rivlin, An Introduction to the Approximation of Functions, 1969, first ed., Dover, 2003.
[36] R.B. Lehoucq, D.C. Sorensen, C. Yang, ARPACK User's Guide: Solution of Large Scale Eigenvalue Problems with Implicitly Restarted Arnoldi Methods, SIAM, Philadelphia, 1998.
[37] Y. Saad, Chebyshev acceleration techniques for solving nonsymmetric eigenvalue problems, Math. Comput. 42 (166) (1984) 567–588.
[38] Y. Zhou, Y. Saad, M.L. Tiago, J.R. Chelikowsky, Self-consistent-field calculation using Chebyshev-filtered subspace iteration, J. Comput. Phys. 219 (1) (2006) 172–184.
[39] Y. Zhou, R.-C. Li, Bounding the spectrum of large hermitian matrices, Linear Algebra Appl. (2010), doi:10.1016/j.laa.2010.06.034.
[40] Y. Notay, Combination of Jacobi–Davidson and conjugate gradients for the partial symmetric eigenproblem, Numer. Linear Algebra Appl. 9 (1) (2002) 21–44.
[41] J. Daniel, W.B. Gragg, L. Kaufman, G.W. Stewart, Reorthogonalization and stable algorithms for updating the Gram–Schmidt QR factorization, Math. Comput. 30 (1976) 772–795.
[42] D.S. Watkins, L. Elsner, Convergence of algorithms of decomposition type for the eigenvalue problem, Linear Algebra Appl. 41 (1991) 19–47.
[43] R.B. Lehoucq, Implicitly restarted Arnoldi methods and subspace iteration, SIAM J. Matrix Anal. Appl. 23 (2001) 551–562.
[44] T. Davis, University of Florida Sparse Matrix Collection, Technical Report, University of Florida, 2009 (NA Digest, vol. 97, no. 23, June 7, 1997).
[45] W. Kohn, L.J. Sham, Self-consistent equations including exchange and correlation effects, Phys. Rev. 140 (1965) A1133–A1138.
[46] W. Kohn, Nobel lecture: electronic structure of matter-wave functions and density functional, Rev. Mod. Phys. 71 (5) (1999) 1253–1266.
[47] J.R. Chelikowsky, N. Troullier, Y. Saad, Finite-difference-pseudopotential method: electronic structure calculations without a basis, Phys. Rev. Lett. 72 (1994) 1240–1243.
[48] J. Chelikowsky, The pseudopotential-density functional method applied to nanostructures, J. Phys. D: Appl. Phys. 33 (2000) R33–R50.
[49] A.V. Knyazev, Toward the optimal preconditioned eigensolver: locally optimal block preconditioned conjugate gradient method, SIAM J. Sci. Comput. 23 (2) (2001) 517–541.
[50] D.R. Fokkema, G.L.G. Sleijpen, H.A. van der Vorst, Jacobi–Davidson style *QR* and *QZ* algorithms for the reduction of matrix pencils, SIAM J. Sci. Comput. 20 (1) (1998) 94–125.
[51] J. Baglama, D. Calvetti, L. Reichel, IRBL: an implicitly restarted block-Lanczos method for large-scale Hermitian eigenproblems, SIAM J. Sci. Comput. 24 (2003) 1650–1677.
[52] J. Baglama, D. Calvetti, L. Reichel, irbleigs: a MATLAB program for computing a few eigenpairs of a large sparse Hermitian matrix, ACM Trans. Math. Software 5 (2003) 337–348.
[53] Y. Saad, A. Stathopoulos, J. Chelikowsky, K. Wu, S. Öğüt, Solution of large eigenvalue problems in electronic structure calculations, BIT 36 (3) (1996) 563–578.
[54] A. Stathopoulos, S. Öğüt, Y. Saad, J. Chelikowsky, H. Kim, Parallel methods and tools for predicting materials properties, Comput. Sci. Eng. 2 (2000) 19–32.
[55] K. Wu, A. Canning, H.D. Simon, L.-W. Wang, Thick-restart Lanczos method for electronic structure calculations, J. Comput. Phys. 154 (1999) 156–173.
[56] P. Arbenz, U.L. Hetmaniuk, R.B. Lehoucq, R.S. Tuminara, A comparison of eigensolvers for large-scale 3d model analysis using AMG-preconditioned iterative methods, Int. J. Numer. Methods Eng. 64 (2) (2005) 204–236.
[57] Y. Zhou, Y. Saad, M.L. Tiago, J.R. Chelikowsky, Parallel self-consistent-field calculations using Chebyshev-filtered subspace acceleration, Phys. Rev. E 74 (6) (2006) 066704-1–066704-8.
[58] M.L. Tiago, Y. Zhou, M. Alemany, Y. Saad, J.R. Chelikowsky, Evolution of magnetism in iron from the atom to the bulk, Phys. Rev. Lett. 97 (2006) 147201.
[59] J.R. Chelikowsky, Y. Saad, T.-L. Chan, M.L. Tiago, A.T. Zayak, Y. Zhou, Pseudopotentials on grids: application to the electronic, optical, and vibrational properties of silicon nanocrystals, J. Comput. Theor. Nanosci. 6 (6) (2009) 1247–1261.