SIAM J. MATRIX ANAL. APPL.
Vol. 29, No. 3, pp. 954–971

A CHEBYSHEV–DAVIDSON ALGORITHM FOR LARGE SYMMETRIC EIGENPROBLEMS*

YUNKAI ZHOU† AND YOUSEF SAAD‡

Abstract. A polynomial filtered Davidson-type algorithm is proposed for symmetric eigenproblems, in which the correction-equation of the Davidson approach is replaced by a polynomial filtering step. The new approach has better global convergence and robustness properties when compared with standard Davidson-type methods. The typical filter used in this paper is based on Chebyshev polynomials. The goal of the polynomial filter is to amplify components of the desired eigenvectors in the subspace, which has the effect of reducing both the number of steps required for convergence and the cost in orthogonalizations and restarts. Numerical results are presented to show the effectiveness of the proposed approach.

Key words. polynomial filter, Davidson-type method, global convergence, Krylov subspace, correction-equation, eigenproblem

AMS subject classifications. 15A18, 15A23, 15A90, 65F15, 65F25, 65F50

DOI. 10.1137/050630404

1. Introduction.

We consider a Davidson-type method for the standard eigenvalue problem

$$Au = \lambda u, \tag{1.1}$$

where $A \in \mathbb{R}^{n\times n}$ is symmetric, n is large, and a large number of eigenpairs need to be computed. We assume throughout that the eigenvalues wanted are the smallest ones. There is a growing need for solving this type of problem efficiently. To cite just one example, symmetric eigenvalue problems usually constitute the most time-consuming part of electronic structure calculations [18, 12, 6].

The original Davidson method [11] was initially designed for diagonally dominant matrices, which for eigenvalue problems means matrices whose off-diagonal elements are small compared with the changes in magnitude between diagonal elements [19]. The Davidson approach sacrifices the attractive Krylov subspace structure, at the cost of having to compute eigenpairs and associated residual vectors of a projection matrix at each (outer) iteration. The trade-off is that the Davidson approach can augment the subspace by a new vector potentially much better than the one based on a strict Krylov subspace structure.

The "augmentation vector" added to the subspace at each step usually results from solving a correction-equation. The efficiency of the standard Davidson-type methods depends on the quality of the correction-equation used. Efficient (preconditioned) linear equation solvers are often utilized to solve the correction-equations. The original Davidson method uses the correction-equation $(\mathrm{diag}(A) - \mu I)t = -r$, where

*Received by the editors May 2, 2005; accepted for publication (in revised form) by D. Calvetti March 28, 2007; published electronically October 5, 2007. This work was supported by the U.S. Department of Energy under contract DE-FG02-03ER25585, by NSF grants ITR-0428774 and CMMI-0727194, and by the Minnesota Supercomputing Institute.
http://www.siam.org/journals/simax/29-3/63040.html
†Department of Mathematics, Southern Methodist University, Dallas, TX 75275 (yzhou@smu.edu).
‡Department of Computer Science and Engineering, University of Minnesota, Minneapolis, MN 55455 (saad@cs.umn.edu).

$r = Ax - \mu x$ is the residual vector corresponding to a Ritz pair (μ, x), and t is the augmentation vector to be computed. In [19] better approximations of A are used to replace the diagonal of A in the correction equation. However, it was noted in [19] that using the "exact preconditioner" leads to stagnation, since $t = (A - \mu I)^{-1} r = -x$ cannot augment the subspace. This led to the development of the more efficient Jacobi–Davidson (JD) algorithm [32, 14, 31]. Work in the literature has shown that the JD can be competitive with efficient Krylov subspace methods such as those in [33, 17, 36, 42, 45]. In [41, 43] other correction-equations for Davidson-type methods are derived.

The JD method can be related to the Newton method or the approximate Rayleigh-quotient iteration (RQI). As such, it has been observed that the method can be rather slow if the starting vector is far away from the desired eigenvector. We note that even though RQI is globally convergent for symmetric eigenproblems (see [22], [23, p. 81]), it may converge to an unwanted eigenpair. The global convergence can be slow when the approximate RQI is used in a subspace method and the method is required to converge to wanted eigenpairs. Global acceleration schemes for JD have been studied. For example, in [5] a nonlinearized JD correction-equation is proposed; however, the preconditioning may be difficult to apply for the nonlinearized correction-equation, and the approach can become much more expensive than the JD method when the number of desired eigenpairs is large. Another approach to achieving better global convergence, as suggested in [19, 14], is to apply an Arnoldi or Lanczos method to get a good initial vector and then apply the JD algorithm.

In this paper, we explore a different Davidson-type approach called the Chebyshev–Davidson method. There is no need to form or solve any correction-equations within this approach; instead, intervalwise filtering based on Chebyshev polynomials is utilized.

The Chebyshev–Davidson approach is very suitable for problems where solving (preconditioned) equations is expensive, e.g., when good preconditioners for correction-equations are either unknown or too expensive to construct.

Details of the Chebyshev–Davidson method are presented in sections 3–4. Comparisons with existing representative eigenvalue algorithms are in section 6. The Chebyshev–Davidson method (written in Fortran95) has been applied to solve a class of highly challenging problems with dimension over a few millions, where more than ten thousand eigenpairs need to be computed.

2. Advantages of polynomial filtering. The global convergence of a Davidson-type method can be improved in a natural and systematic way via polynomial filtering. We first make the following three observations. The first is on the well-known polynomial filtering argument: For a symmetric matrix A with the eigendecomposition $A = Q\Lambda Q^T$, any polynomial $\psi(s) : \mathbb{R} \to \mathbb{R}$ satisfies

$$\psi(A)v = Q\psi(\Lambda)Q^T v \quad \forall v \in \mathbb{R}^n. \tag{2.1}$$

The second observation is on the fast local convergence of JD. It is shown in [43] that the locally fast convergence of JD is mainly caused by the retention of the approximate RQI direction in the basis of the projection subspace. Assume throughout that (μ, x) denotes the current Ritz pair that best approximates a wanted eigenvalue, and the Ritz vector x is of unit length, and let $r = Ax - \mu x$ denote the residual. It was observed in [43] that the JD correction equation,

$$\text{Solve for } t \perp x \text{ from } \quad (I - xx^T)(A - \mu I)(I - xx^T)t = r, \tag{2.2}$$

can be simplified to

$$\text{Solve for } t \text{ from } \quad (I - xx^T)(A - \mu I)t = r. \tag{2.3}$$

The right projection by $(I - xx^T)$ and the final orthogonality constraint $t \perp x$ can be omitted. It is the approximate RQI direction, which is an approximation to $(A - \mu I)^{-1}x$, that leads to the success of the JD approach. The left projector $(I - xx^T)$ in (2.2) is crucial in retaining the important approximate RQI direction in the JD direction (solution t of (2.2)). This can be readily seen by writing (2.2) or (2.3) as

$$(A - \mu I)t = r + x\ \alpha, \tag{2.4}$$

where α is a nonzero scalar. The left projector also improves the conditioning of (2.2) and (2.3) on the $x^\perp$ subspace—the subspace in which a vector to augment the current projection subspace is sought, but this property is irrelevant for this paper.

Note that the exact RQI direction is $(A-\mu I)^{-1}x$, which is the current Ritz vector x filtered by the rational polynomial $\varphi(s) = \frac{1}{s-\mu}$. This polynomial significantly magnifies the direction of a possibly wanted eigenvector corresponding to the Ritz value μ (the current best approximation to a wanted eigenvalue of A).

The third observation is that one can improve global convergence by choosing a polynomial $\psi(s)$ which magnifies not only the direction corresponding to one single point, but also directions corresponding to an interval containing wanted eigenvalues, and at the same time dampens unwanted eigenvalues. With polynomial filtering, it is unlikely that wanted eigenvalues will be missed, because when the whole interval containing wanted eigenvalues is magnified, so is each wanted eigenvalue in the interval. In contrast, standard Davidson-type methods may miss some wanted eigenvalues. This is because correction-equations often resemble certain shift-invert formulations, and the shift chosen at some step may approximate larger eigenvalues before all the wanted smaller eigenvalues are computed. Chebyshev filtering offers an alternative which can improve global convergence as well as robustness (in the sense that wanted eigenvalues are not missed) of Davidson-type methods.

Note that polynomial filters have long been exploited to accelerate Arnoldi/Lanczos algorithms; see, e.g., [26, 33]. Here we consider a natural application of Chebyshev polynomials within a Davidson-type (non-Krylov) framework. This approach combines the acceleration power of the Chebyshev filtering technique and the flexibility and robustness of the Davidson approach.

To further explain the third observation, we suppose that the eigenvalues of A are ordered as $\lambda_1 \le \lambda_2 \le \cdots \le \lambda_n$, and that the wanted eigenvalues are located in $[\lambda_1, \lambda_k]$. If $\psi(s)$ is chosen to approximate the step function

$$\phi(s) = \begin{cases} 1, & \lambda_1 \le s \le \lambda_k, \\ 0, & \lambda_k < s \le \lambda_n, \end{cases} \tag{2.5}$$

then (2.1) shows that $\psi(A)v \approx \sum_{i=1}^{k} \alpha_i q_i$, where q_i is the ith column of Q and $\alpha_i = q_i^T v$. That is, $\psi(A)v$ is contained in the subspace spanned by the wanted eigenvectors $Q(:, 1:k)$. If this $\psi(A)v$ is augmented into the basis, convergence to the wanted eigenvectors is expected to be much faster than augmenting the basis by a vector closer to unwanted eigenvectors. This claim can be verified by explicitly computing Q and using $\sum_{i=1}^{k} \alpha_i q_i$ ($\alpha_i = q_i^T x$, where x denotes the current Ritz vector at each iteration) as the augmentation vector in a Davidson-type method. This is equivalent to using a filter that exactly approximates (2.5). Note that this filter

leads to no gap among wanted eigenvalues, but in this ideal setting it can still lead to fast convergence in a Davidson-type method. However, a low degree polynomial cannot approximate (2.5) well. In real computations, a filter that can introduce more favorable gaps for the wanted eigenvalues is far better than others that introduce no gap.

For a subspace method applied to (1.1), the essence in obtaining fast convergence is in augmenting the subspace by vectors close to the wanted invariant subspace of A. Therefore, a convergence acceleration scheme should construct a suitable filter ψ so that the vector $\psi(A)v$ used for augmentation is contained in the wanted eigensubspace. By this filtering we obtain better global convergence.

According to observations just made, it is essential to filter the current Ritz vector x, not the residual vector r. Note that at each iteration of a Davidson-type method, r is orthogonal to the projection basis, and this basis is used to approximate the wanted eigenvectors; hence r can become orthogonal to the wanted eigenvectors during the iteration. The residual vector r is not suitable for the filtering because, when $Q(:, 1:k)^T r \approx 0$, $\psi(A)r = Q\psi(\Lambda)Q^T r$ is approximately inside the subspace spanned by unwanted eigenvectors.

We also mention that our own experiments, together with those in [13], show that the preconditioned Davidson method based on equation $(A - \mu I)t = r$ can be inefficient because of the higher possibility of stagnation if this equation is solved more accurately. An efficient correction-equation essentially should retain the approximate RQI direction in its solution. Thus, the JD method is equivalent to (2.4), but the x term in the right-hand side of (2.4) may be more important than the r term. However, in the case of rather inaccurate solves with a fixed preconditioner, [1] shows that $(A - \mu I)t = r$ can have better performance than other correction equations.

3. Chebyshev polynomial filter. The observations in section 2 suggest that polynomials which significantly magnify the lowest end of a wanted interval and dampens unwanted intervals at the same time can be used as a filter to improve global convergence. The well-known Chebyshev polynomials are a natural choice for this task. Using Chebyshev polynomials to accelerate symmetric eigenvalue computations dates back to [24, 25]. A nice discussion on Chebyshev accelerated subspace iteration can be found in [23, pp. 329–330], where it is mentioned that the Lanczos method is usually better than the Chebyshev accelerated (fixed dimension) subspace iteration algorithm. Here we integrate Chebyshev filtering into a varying dimension Davidson-type algorithm.

With Chebyshev acceleration, the subspace used in a Davidson-type method can be of much smaller dimension than that is required by a Lanczos-type method for good efficiency. Therefore the filtering approach leads to substantial savings in (re-) orthogonalization costs.

Recall that the real Chebyshev polynomials of the first kind are defined by (see, e.g., [23, p. 371], [27, p. 142])

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

Note that $C_0(t) = 1, C_1(t) = t$. Recall also the important three-term recurrence,

$$C_{k+1}(t) = 2tC_k(t) - C_{k-1}(t), \qquad t \in \mathbb{R}. \tag{3.1}$$

A remarkable property of the Chebyshev polynomial is its rapid growth outside the interval $[-1, 1]$. This property is illustrated in Figure 3.1. Here we plot only the

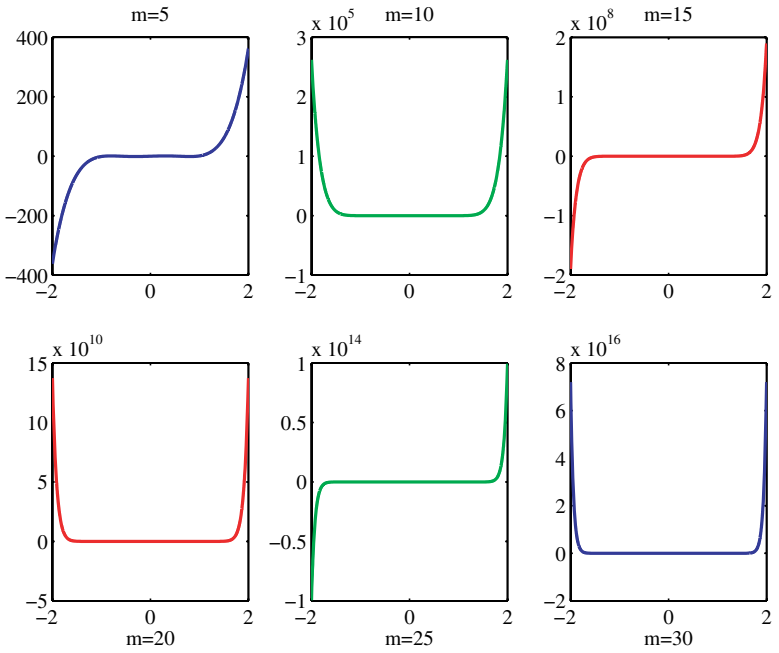

FIG. 3.1. *Rapid increase outside* $[-1, 1]$ *of Chebyshev polynomial of degree* m.

polynomial on the $[-2, 2]$ interval, but note that the farther away we are from $[-1, 1]$, the larger the magnitude of $C_k(t)$. Suppose that the spectrum of A is contained in $[a_0, b]$ and we want to dampen the interval $[a, b]$ for $a > a_0$; then we need only to map $[a, b]$ into $[-1, 1]$ by an affine mapping. This mapping will map the wanted lower end of the spectrum, i.e., the eigenvalues closer to a_0, farther away from $[-1, 1]$ than the ones closer to a. Applying the three-term Chebyshev recurrence will then magnify eigenvalues near a_0 and dampen eigenvalues in $[a, b]$, which is the desired filtering.

In practice, we need the lower bound a of the unwanted interval, which is easy to approximate during each iteration in a Davidson-type method. The upper bound b of the eigenvalues of A can by obtained by Gerschgorin's theorem. It can also be estimated by an upper-bound-estimator (Algorithm 4.3 in [47]), which applies a few steps of Lanczos iteration with a final safeguard step.

The Chebyshev iteration, which dampens values in $[a, b]$ while magnifying values in the interval to the left of $[a, b]$, is presented in Algorithm 3.1 below. Here we follow the formula derived in [26], [27, p. 223] for the complex Chebyshev iteration and adapt it to the real case. The iteration of the algorithm is equivalent to computing

$$y = p_m(A)x, \qquad \text{where} \quad p_m(t) = C_m\left(\frac{t-c}{e}\right). \tag{3.2}$$

As defined in the algorithm, c is the center of the interval $[a, b]$ and e its half-width; both depend on the bounds used.

In Algorithm 3.1, the σ's are used for scaling purposes; the a_0 is a crude approximation of the smallest eigenvalues of A. The following discusses certain details of scaling. The three-term recurrence using $p_m(A)$ yields the iteration

$$x_{j+1} = \frac{2}{e}(A - cI)x_j - x_{j-1}, \quad j = 1, 2, \ldots, m-1,$$

with x_0 given and $x_1 = (A - cI)x_0$. This is equivalent to a power iteration of the form

$$\begin{pmatrix} x_{j+1} \\ x_j \end{pmatrix} = \underbrace{\begin{pmatrix} \frac{2}{e}(A - cI) & -I \\ I & 0 \end{pmatrix}}_{\mathcal{B}} \begin{pmatrix} x_j \\ x_{j-1} \end{pmatrix}.$$

A little analysis would show that all the eigenvalues of the nonsymmetric matrix $\mathcal{B}$ are complex and of modulus one, except that those corresponding to eigenvalues of A that are less than a are mapped to real eigenvalues larger than one in magnitude. Therefore, as for the standard power method, a scaling at each step is recommended. The simplest strategy, discussed in [27], is to consider the scaled sequence

$$\tilde{x}_j = \frac{C_j[\frac{2}{e}(A - cI)]}{C_j[\frac{2}{e}(a_0 - cI)]}\, x_0,$$

where $\rho_j = C_j[\frac{2}{e}(a_0 - cI)]$ is the scaling factor. This requires a_0, but since it is only used for scaling, a rough estimate of a_0 is sufficient. For the first Chebyshev–Davidson iteration, we can use a value $a_0 \le a$; for the latter Chebyshev–Davidson steps, the smallest Ritz value from the previous step can be used. The vector sequence is not computed as shown above, because ρ_j itself can be large and this would defeat the purpose of scaling. Instead, each $\tilde{x}_{j+1}$ is updated using the scaled vectors $\tilde{x}_j$ and $\tilde{x}_{j-1}$. This is discussed in [27], and Algorithm 3.1 implements this scaling (the tildes and vector subscripts are omitted).

ALGORITHM 3.1. *[y] =* `Chebyshev_filter`*(x, m, a, b, a_0).*
Purpose: Filter x by an m degree Chebyshev polynomial which dampens on $[a, b]$.

1. $e = (b - a)/2$; $c = (b + a)/2$;
2. $\sigma = e/(a_0 - c)$; $\sigma_1 = \sigma$;
3. $y = (Ax - cx)\sigma_1/e$;
4. *For* $i = 2 : m$
5. $\quad \sigma_{new} = \frac{1}{(2/\sigma_1 - \sigma)}$;
6. $\quad y_{new} = 2(Ay - cy)\sigma_{new}/e - \sigma\sigma_{new}x$;
7. $\quad x = y$; $\quad y = y_{new}$; $\quad \sigma = \sigma_{new}$;
8. *End For*

Note that the filter is intervalwise; hence no shifts such as Chebyshev zeros or Leja points are required. This is to be contrasted with pointwise filtering methods, e.g., [33, 17, 2, 3].

Clearly, polynomials other than Chebyshev can be used for filtering. Several polynomials are discussed in [34], with emphasis on approximating rational functions of the form $\varphi(s) = 1/(s - \mu)$. In contrast, we do not approximate the (shift-inverted) rational functions, but require polynomials to have the desired intervalwise filtering property, i.e., dampen an interval and significantly magnify other intervals. We choose Chebyshev polynomials because of their desirable filtering properties and ease of implementation. Note that the Chebyshev filtering used in [34] is different from Algorithm 3.1, since the former requires an additional parameter Δ, which is not straightforward to specify. Another difference is that in [34] Chebyshev filtering is used in a Lanczos-type algorithm, while here we integrate Chebyshev filtering into a Davidson-type framework. The Rayleigh–Ritz step in a Davidson-type method readily provides the necessary bounds for constructing efficient Chebyshev filters. The resulting Chebyshev–Davidson method compares favorably with other methods, as shown in section 6.

Algorithm 3.1 requires no inner products, and this is another appealing feature of Chebyshev acceleration, since inner products incur a global reduction which requires additional communication costs in a parallel computing context.

4. Chebyshev polynomial accelerated Davidson method. The pseudocode for the Chebyshev–Davidson method is presented in Algorithm 4.1 below. This code is very different from other Davidson-type methods in the literature (e.g., [4]). We use a natural but useful indexing scheme. The deflation of converged eigenvectors is handled by indexing the columns of the projection basis V. No extra storage for the converged eigenvectors is necessary. Moreover, restarting is simplified (as seen in step 8f) by the indexing. The implementation does not require extra basis updates or memory copies during the restart, since the updates in step 8g need to be performed even when restart is not necessary. We note that putting restart at step 8f is better than putting it at the end of the outer loop, because it saves operations in step 8g when restarting is necessary.

We make a few comments on Algorithm 4.1. Comments (v)–(vii) are related to the robust implementation of any Davidson-type methods.

(i) It is important that the bound `upperb` bounds all eigenvalues of A from above. Otherwise the interval containing largest eigenvalues may also be magnified through filtering, and this can drastically slow convergence or even lead to wrong convergence. One inexpensive way for the bound estimation at step 5 is `upperb` $= \left|\left|A\right|\right|_1$; if A is available only through a matrix-vector-product subroutine, then we can apply the upper-bound-estimator, Algorithm 4.1 in [47], to get an upper bound.

(ii) The choice of the lower bound for the unwanted interval at each iteration is one of the most critical ingredients of the method. The Chebyshev–Davidson method allows quite flexible choices for this lower bound, without any extra computations. Numerical results show that the choice at step 8l is remarkably efficient. Other choices, such as the maximum of the current Ritz values, can also be used as `lowerb`.

(iii) For the orthogonalization step 8b, we use the iterated Gram–Schmidt algorithm, often referred to as the DGKS method [10].

(iv) The refinement at step 8g is performed at each step. One can avoid this step until some eigenpair converges. But according to [23, p. 325], this refinement is necessary in order to have faster convergence for the eigenvectors.

(v) The swap at step 8j may be performed by the following pseudocode:
set $v_{tmp} = V(:, k_c)$;
For $(i = k_c - 1 : -1 : 1)$ Do
  If $(\mu \geq eval(i))$, exit the For loop; End If
  set $eval(i+1) = eval(i)$, $eval(i) = \mu$; set $V(:, i+1) = V(:, i)$, $V(:, i) = v_{tmp}$;
End For.
(Note that unnecessary memory copies in the above can be avoided with some more involved programming.)

(vi) The *noswap* flag at steps 8i–k is used to improve robustness. This flag decreases the possibility of counting converged unwanted eigenvalues as wanted ones.

(vii) At step 8j, a convergence test is performed only on the first Ritz pair among the $k_{sub} - k_c$ Ritz pairs available at each iteration. A simple loop can be added to check the convergence of more than one Ritz pair. We note that for almost all Davidson-type subspace methods, if all the $k_{sub} - k_c$ Ritz pairs are checked for convergence at each iteration step and no swap procedure is included, then there is a high possibility of missing wanted eigenvalues.

(viii) Algorithm 4.1 essentially contains a framework for Davidson-type methods based on filtering. The Chebyshev filter at step 8a can be replaced by other suitable filters. However, we mention that among the filters tried, including a least square polynomial and a different implementation of Chebyshev polynomials in [34], the filter as implemented in Algorithm 3.1 has the best numerical behavior.

5. Analysis. The analysis given here serves to give a preliminary understanding of the convergence for Algorithm 4.1. Therefore several simplifications of the algorithm are made.

Assume that the eigenvalues of A are $\lambda_1 < \lambda_2 \leq \cdots \leq \lambda_n$, and denote the associated unit eigenvectors by $q_1, \ldots, q_n$. According to (3.2) and the fact that the

ALGORITHM 4.1. Chebyshev–Davidson method.
(computing k_{want} number of smallest eigenpairs)
Input: *x—initial vector; m—polynomial degree; k_{keep}—# of vectors to keep during restart; dim_{max}—maximum subspace dimension; τ—convergence tolerance.*
Output: *converged eigenvalues $eval(1:k_c)$ (in nonincreasing order) and their corresponding eigenvectors $V(:, 1:k_c)$, where k_c denotes # of converged eigenpairs.*

1. *Start with the unit vector x, $V = [x]$.*
2. *Compute $W = [Ax], H = [\mu]$, where $\mu = x^T w$.*
3. *Compute the residual vector: $r = W(:,1) - \mu x$.*
4. *If $||r|| <= \tau$, set $eval(1) = \mu$, $k_c = 1$, $H = [\,]$; Else, set $k_c = 0$.*
5. *Estimate the upper bound (`upperb`) of eigenvalues.*
6. *Set `lowerb` $= (\texttt{upperb} + \mu)/2$, $a_0 =$ `lowerb`.*
7. *Set $k_{sub} = 1$ (k_{sub} stores the current subspace dimension).*
8. *Outer Loop:* **Do while** *($iter \leq iter_{max}$)*
   a. *Call the Chebyshev polynomial filter:*
      $[t] =$ `Chebyshev_filter`$(x, m,$ `lowerb`, `upperb`, $a_0)$.
   b. *Orthonormalize t against $V(:, 1:k_{sub})$ to get a unit vector $V(:, k_{sub}+1)$;*
      set $k_{sub} \leftarrow k_{sub} + 1$; set $k_{old} \leftarrow k_{sub}$.
   c. *Compute $W(:, k_{sub}) = AV(:, k_{sub})$.*
   d. *Compute the last column of the symmetric Rayleigh-quotient matrix H:*
      $H(1:k_{sub}-k_c, k_{sub}-k_c) = V(:, k_c+1:k_{sub})^T W(:, k_{sub})$.
   e. *Compute the eigendecomposition of H: $HY = YD$,*
      where $\mathrm{diag}(D)$ is in nonincreasing order. Set $\mu = D(1,1)$.
   f. *If ($k_{sub} \geq dim_{max}$), then restart: set $k_{sub} = k_c + k_{keep}$.*
   g. *Update basis: $V(:, k_c+1:k_{sub}) \leftarrow V(:, k_c+1:k_{old})Y(:, 1:k_{sub}-k_c)$;*
      update W: $W(:, k_c+1:k_{sub}) \leftarrow W(:, k_c+1:k_{old})Y(:, 1:k_{sub}-k_c)$.
   h. *Compute the residual vector: $r = W(:, k_c+1) - \mu V(:, k_c+1)$.*
   i. *Set noswap $= 0$, iter $\leftarrow$ iter $+ 1$.*
   j. *Test for convergence: If $||r|| <= \tau \ \max(\mathrm{diag}(D))$, set $k_c = k_c + 1$,*
      set $eval(k_c) = \mu$; also swap eigenpairs if necessary (see comment (v))
      so that converged eigenvalues are in nonincreasing order;
      set noswap $= 1$ if any swap happens.
   k. *If ($k_c \geq k_{want}$ and noswap $== 0$),* **Return** *$eval(1:k_c)$ and $V(:, 1:k_c)$ as the converged wanted eigenpairs.* **Exit**.
   l. *Update lower bounds:* `lowerb` $= median(\mathrm{diag}(D))$;
      If $a_0 > \min(\mathrm{diag}(D))$, set $a_0 \leftarrow \min(\mathrm{diag}(D))$.
   m. *Set the next Ritz vector for filtering: $x = V(:, k_c+1)$.*
   n. *Update H: $H = D(k_c+1:k_{sub}, k_c+1:k_{sub})$.*

interval of the eigenvalues to be dampened at each step is adaptively changing, we see that the matrix applied at the jth step is

$$p_m^{(j)}(A) = C_m^{(j)}((A - c_j I)/e_j). \tag{5.1}$$

The first simplification assumes that the interval of the eigenvalues to be dampened at each filtering step of Algorithm 4.1 is fixed. That is, the matrix involved is fixed as $p_m(A) \equiv C_m((A - cI)/e)$. The second simplification assumes that no restart is used in the algorithm.

We further assume that Algorithm 4.1 is a *one-dimensional* version. That is, in step 8b we keep $k_{sub} \equiv 1$ and set $V(:,1) = t/\big|\big|t\big|\big|$, and in step 8m we set $x = V(:,1)$. Then the algorithm becomes a standard power method with the matrix $p_m(A)$. As a result, the convergence will be governed by the ratio of the two dominant eigenvalues. Note that the interval $[a,b]$ of the eigenvalues to be dampened satisfies $\lambda_1 < a$. The (unique) dominant eigenvalue of the matrix $p_m(A)$ is $C_m((\lambda_1 - c)/e)$. So, in the one-dimensional version of the algorithm, $V(:,1)$ converges to q_1 with the convergence factor

$$\rho = \frac{\max_{j>1} |C_m((\lambda_j - c)/e)|}{|C_m((\lambda_1 - c)/e)|} < 1.$$

Consider now the situation in which k_{sub} can be increased. The simplified method turns out to have a simple Krylov interpretation. Assume that we perform two steps of the algorithm, i.e., that the dimension of the subspace is two. The first vector of the basis is $p_m(A)x$. The second is obtained as $p_m(A)x_1$, where x_1 is an approximate eigenvector from the one-dimensional space spanned by the first vector, which is simply a multiple of $p_m(A)x$. The subspace used in this case is

$$K_2 = \text{span}\{p_m(A)x, p_m(A)x_1\} = \text{span}\{p_m(A)x, p_m^2(A)x\} \ ,$$

which is the Krylov subspace of dimension two usually denoted by $K_2(p_m(A), x)$. Consider now the third step. The process will inject to the subspace a vector of the form

$$p_m(A)x_2 \quad \text{with} \quad x_2 \in K_2 \ .$$

The vector x_2 is a Ritz eigenvector computed from projecting A onto the subspace K_2, and it is a linear combination of vectors from K_2, so we can write $x_2 = \alpha_1 p_m(A)x + \alpha_2 p_m^2(A)x$. The new subspace K_3 is again a Krylov subspace. Indeed,

$$K_3 = \text{span}\{p_m(A)x, p_m^2(A)x, p_m(A)x_2\} \equiv \text{span}\{p_m(A)x, p_m^2(A)x, p_m^3(A)x\} \ .$$

The result can be easily extended to an arbitrary step j for the simplified method.

PROPOSITION 5.1. *Assuming that the filtering interval is fixed and no restart is applied, then step j of Algorithm* 4.1 *is mathematically equivalent to a Rayleigh–Ritz process applied to A using the Krylov subspace*

$$K_j\left(p_m(A), x\right).$$

In particular, this means that if one generated an orthogonal basis V_j of the Krylov subspace $K_j(p_m(A), x)$ and computed the eigenvalues of $V_j^T A V_j$, these eigenvalues would be identical with those of the simplified Algorithm 4.1. This is not quite a

Krylov subspace method, because the projection uses A instead of the transformed matrix $p_m(A)$. However, this simple result permits one to analyze the simplified algorithm in a complete way by considering eigenvectors. Indeed, eigenvectors of A and $p_m(A)$ are identical, and there are results which establish upper bounds for the angle between the exact eigenvector and the Krylov subspace. This will be omitted, and the reader is referred to [23] for details.

Although the simplified algorithm can be viewed from the angle of Krylov subspaces, Algorithm 4.1 is not a Krylov method. There are a number of distinguishing features, related to implementation and other practical aspects. In fact, Proposition 5.1 assumes that the polynomial is fixed, but the actual Chebyshev–Davidson method adapts the filters by dynamically adjusting the bounds of the interval of eigenvalues to be dampened at each iteration. This in practice leads to significantly more efficient filters. The following presents a conservative analysis: Taking (5.1) into account, we see that

$$K_2 = \operatorname{span}\{p_m^{(1)}(A)x, p_m^{(2)}(A)p_m^{(1)}(A)x\},$$

$$K_3 = \operatorname{span}\{p_m^{(1)}(A)x,\ p_m^{(2)}(A)p_m^{(1)}(A)x,\ \alpha_1 p_m^{(3)}(A)p_m^{(1)}(A)x + \alpha_2 p_m^{(3)}(A)p_m^{(2)}(A)p_m^{(1)}(A)x\},$$

where $\alpha_2 \neq 0$ by the designed filtering (if the previous subspace can be augmented). This list easily extends to K_j for any j. Letting

$$\Phi_k(t) = \prod_{l=1}^{k} p_m^{(l)}(t), \tag{5.2}$$

then the last term in K_j contains $\Phi_j(A)x$ with a nonzero coefficient. The filtering is designed such that the subspace contains a significant direction in $\Phi_j(A)x$. Note that $\Phi_j(A)x$ corresponds to an accelerated power method applied to x. Because of the Rayleigh–Ritz refinement, there is a Ritz vector from K_j which converges to q_1 at least as fast as $\Phi_j(A)x$ does, where the convergence rate for $\Phi_j(A)x$ to q_1 is $\frac{\max_{l>1}|\Phi_j(\lambda_l)|}{|\Phi_j(\lambda_1)|}$ under standard conditions [39, 16]. This rate can be considerably faster than the one obtained using a fixed filtering interval. The convergence for the latter eigenvectors follows from deflation; e.g., the second eigenvalue becomes dominant for the matrix A restricted to the subspace orthogonal to q_1.

6. Numerical results and discussion. We compare the Chebyshev–Davidson method (denoted as ChebyD) with several other Davidson-type and Lanczos-type methods.

The first part of the comparison is done using Matlab. We compare our algorithm with the well-known JD method implemented in the publicly available Matlab code JDQR [14][1] and JDCG [20].[2] The JDCG code is for symmetric eigenproblems; the linear solver used in JDCG is (preconditioned) CG. The JDQR code can solve both symmetric and nonsymmetric problems; GMRES [28] is the default linear solver in JDQR, and it is used for the numerical tests. Since we solve symmetric eigenproblems, it is less costly to use MINRES [21] in the JD method. Moreover, since CG is usually intended for positive definite problems, JDCG does not work as efficiently for indefinite A as for positive definite A. So for further comparisons, we implemented the JD method using the Matlab built-in MINRES as the linear solver. This code is denoted

[1] Code available at http://www.math.uu.nl/people/sleijpen/JD_software/.

[2] Code available at http://mntek3.ulb.ac.be/pub/docs/jdcg/.

TABLE 6.1
Silicon quantum dot model $Si_{34}H_{36}$, indefinite. dim = 97569, $k_{want} = 100$. For ChebyD $m = 20, k_{keep} = 60$; for JDminres #max_le_solve = 20.

Method	CPU (sec.)	#iter.	#mvp	$\|\|AV - VD\|\|/\|\|A\|\|_1$
ChebyD	1204	706	14806	4.16e-11
JDminres	1968	536	12306	6.44e-11
JDQR	3734	2183(-)	11850	2.73e-13
JDCG	3597	927	37899	2.90e-12
LOBPCG	24190	5289	528900	2.98e-10

TABLE 6.2
`bcsstk32` *from the NIST Matrix Market, indefinite. dim = 44609, $k_{want} = 100$. For ChebyD $m = 20, k_{keep} = 60$; for JDminres #max_le_solve = 20.*

Method	CPU (sec.)	#iter.	#mvp	$\|\|AV - VD\|\|/\|\|A\|\|_1$
ChebyD	418	587	12307	2.38e-11
JDminres	850	577	10360	3.55e-11
JDQR	1186	1441(-)	7639	5.80e-14
JDCG	1250	695	29810	8.86e-13
LOBPCG	1974	686	68600	9.96e-11

TABLE 6.3
Silicon quantum dot model $Si_{87}H_{76}$, indefinite. dim = 240369. $k_{want} = 80$. For ChebyD $m = 20, k_{keep} = 80$; for JDminres #max_le_solve = 20.

Method	CPU (sec.)	#iter.	#mvp	$\|\|AV - VD\|\|/\|\|A\|\|_1$
ChebyD	2915	493	10333	3.15e-11
JDminres	3725	497	11409	2.69e-11
JDQR	7879	2390(-)	13246	2.86e-13
JDCG	7220	720	37520	2.48e-12
LOBPCG	13850	667	53360	1.19e-10

as JDminres. JDminres is mainly based on Algorithm 4.1, except that step 8a is replaced by a linear equation solve using MINRES. The LOBPCG [15] code[3] is also used for comparison because it is a representative preconditioned eigensolver. We also compared with IRBL [2, 3], but noticed that the IRBL code becomes less competitive than other codes when k_{want} becomes large. Here we report only comparisons with JDQR, JDCG, JDminres, and LOBPCG.

For the test examples listed in Tables 6.1–6.3, we compute the k_{want} smallest eigenvalues and eigenvectors. The maximum subspace dimension is fixed at $2 * k_{want}$ for all methods, except that for LOBPCG it is $3 * k_{want}$. The silicon quantum dot models are available from the University of Florida Sparse Matrix Collection.[4] Figure 6.1 shows the sparsity structure of two test matrices used in Tables 6.1 and 6.2.

The accuracy is reported as $\|AV - VD\|/\|A\|_1$, where the diagonal of D contains the k_{want} converged eigenvalues, and V contains the corresponding eigenvectors. The relative convergence tolerance is set to 10^{-10} for all methods. For each test problem, the computed eigenvalues are cross validated; i.e., we compute the maximum difference of the eigenvalues computed by different methods. All the differences are found to be less than order 10^{-10}.

[3] Code available at http://www-math.cudenver.edu/~aknyazev/software/CG/toward/lobpcg.m.
[4] http://www.cise.ufl.edu/research/sparse/matrices/.

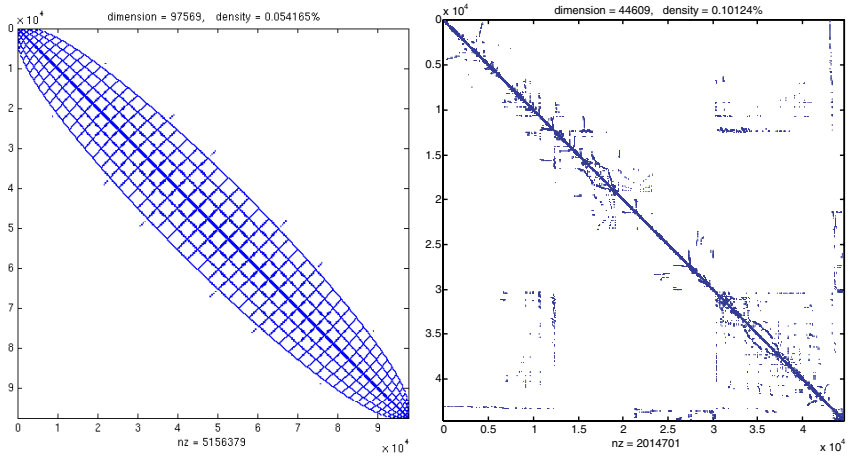

FIG. 6.1. *Structure plots of two test matrices* $Si_{34}H_{36}$ *and* `bcsstk32`.

Initial vector is set as $ones(n, 1)$ for all methods, so that the initial direction is not biased for a certain method. The LOBPCG requires additional $k_{want} - 1$ vectors for the initial block, for which random vectors are used.

In each table, "#iter" counts the total number of the outer loop, "#mvp" is the number of matrix-vector products, and "#max_le_solve" is the maximum inner iteration number for the linear equation solve by MINRES in JDminres. For JDQR, `help jdqr` indicates where #iter and #mvp are stored, but the observed output values from history(:,2) for #iter seem incompatible with the expected values. It is possible that history(:,2) stores both the outer iteration count as well as the inner iteration count, since the resulting number is often much larger than that of JDminres and JDCG. We report #iter for JDQR only for reference, and put a (-) sign to signal the difference. The #mvp for LOBPCG is reported by #iter times the block size (which is k_{want} in LOBPCG).

All the Matlab numerical experiments were performed on an AMD PC with dual Opteron 2.6GHz CPU and 8GB RAM. One of the CPU was dedicated to the computation. The OS used was Red Hat EL4 Linux with kernel version 2.6.9. We used Matlab version 7.2 (R2006a) for the computations.

Note that JDCG and LOBPCG both have preconditioned CG solvers included. However, for the symmetric indefinite problems in Tables 6.1–6.3 (and other test problems not reported here), our experiments showed that both methods work better than their "preconditioned" counterparts with a standard preconditioner such as the incomplete LU. Moreover, for these indefinite problems, it is not clear what preconditioners can be used to accelerate the preconditioned CG solves. Therefore we report comparisons with JDCG and LOBPCG without preconditioned solves. The examples show that in situations where preconditioners are hard to obtain, approaches not relying on solving correction-equations have a clear advantage and can provide effective alternatives.

We recall that the "preconditioning" concept for eigenproblems is quite different from preconditioning for linear equations. The latter tries to reduce the eigenvalue gaps to make the condition number close to 1, while the former tries to introduce more favorable gaps for wanted eigenvalues. This is why in eigenvalue problems, the

preconditioned linear solvers are often applied to correction equations, which leads to techniques that exploit shift-and-invert. In essence, a natural "preconditioner" for an eigenvalue problem is a filter that can transform the spectrum in a desired way so as to increase eigenvalue gaps. This "preconditioning" may not need a preconditioned linear solver. In the Chebyshev–Davidson method, we realize the "preconditioning" by dynamically constructing Chebyshev filters to filter the spectrum so that gaps among wanted eigenvalues are properly magnified.

The reported results of ChebyD are typical for the Chebyshev–Davidson method. For all the test runs, it is rather straightforward to select the polynomial degree m. The effect of a varying degree m is illustrated in Figure 6.2. As seen from this figure, with m increasing (before it becomes unnecessarily large), the number of iterations decreases, the number of matrix-vector products increases, and the CPU time decreases. Note that the CPU time difference for m from 26 to 47 is not large. The test matrices are the $Si_{10}H_{16}$ and $Si_{34}H_{36}$ silicon quantum dots, but we note that similar behavior is observed for a large number of other test models. In Tables 6.1–6.3 we used $m = 20$ to see how the algorithm performs with a low degree polynomial. A better CPU time for Chebyshev–Davidson can be obtained with a larger m. The results in Tables 6.1–6.3 show that even without a fine-tuned m, the Chebyshev–Davidson method outperforms other Davidson-type methods.

The numerical results in Tables 6.1–6.3 and Figure 6.2 also show that a smaller #mvp count does not necessarily imply smaller CPU time. Eigenvalue algorithms may require substantial amounts of work not related to matrix-vector products. For example, in Figure 6.2, #mvp increases with m increasing, but because #iter decreases, there is less reorthogonalization cost involved; this explains why the CPU time decreases as m increases. As pointed out in [30], for large sparse eigenproblems where a large number of eigenpairs need to be computed, the total cost can be dominated by the reorthogonalization cost.

Regarding global convergence, Figure 6.3 shows one example where convergence of the Chebyshev–Davidson method is much faster than that of the standard JD approach. However, we would like to mention that for symmetric eigenproblems, a JD method often has good global convergence. For the same example as in Figure 6.3, a fine-tuned value of #max_le_solve for JDminres can make the global convergence of ChebyD and JDminres become similar.

The Chebyshev–Davidson algorithm was also implemented in Fortran95; its parallel version has been integrated into an electronic structure calculation package called PARSEC (pseudopotetial algorithm for real-space electronic calculations). PARSEC uses real-space pseudopotential implementation of density functional theory methods. The original ideas behind PARSEC date back to the early 1990s [8, 9]. Originally, PARSEC had three diagonalization methods: a preconditioned Davidson method [29, 35] called Diagla, the symmetric eigensolver from ARPACK [33, 17], and the thick-restart Lanczos method (TRLan) [40, 42]. The Chebyshev–Davidson algorithm was subsequently integrated into PARSEC (around October, 2005). Due to its efficiency and robustness, it was quickly adopted as the default eigensolver by our collaborators in material science. In the latest version of PARSEC, a true diagonalization is performed only at the first step of the self-consistent loop, after which diagonalizations are replaced by a nonlinear Chebyshev-filtered-subspace (CheFS) method [47, 46]. Nevertheless, the first diagonalization step can still be highly challenging. This is because a relatively complex material system can contain several thousand atoms, in which case the dimension of the discretized Hamiltonians can easily exceed several millions. Even more challenging is the fact that the number of eigenpairs needed

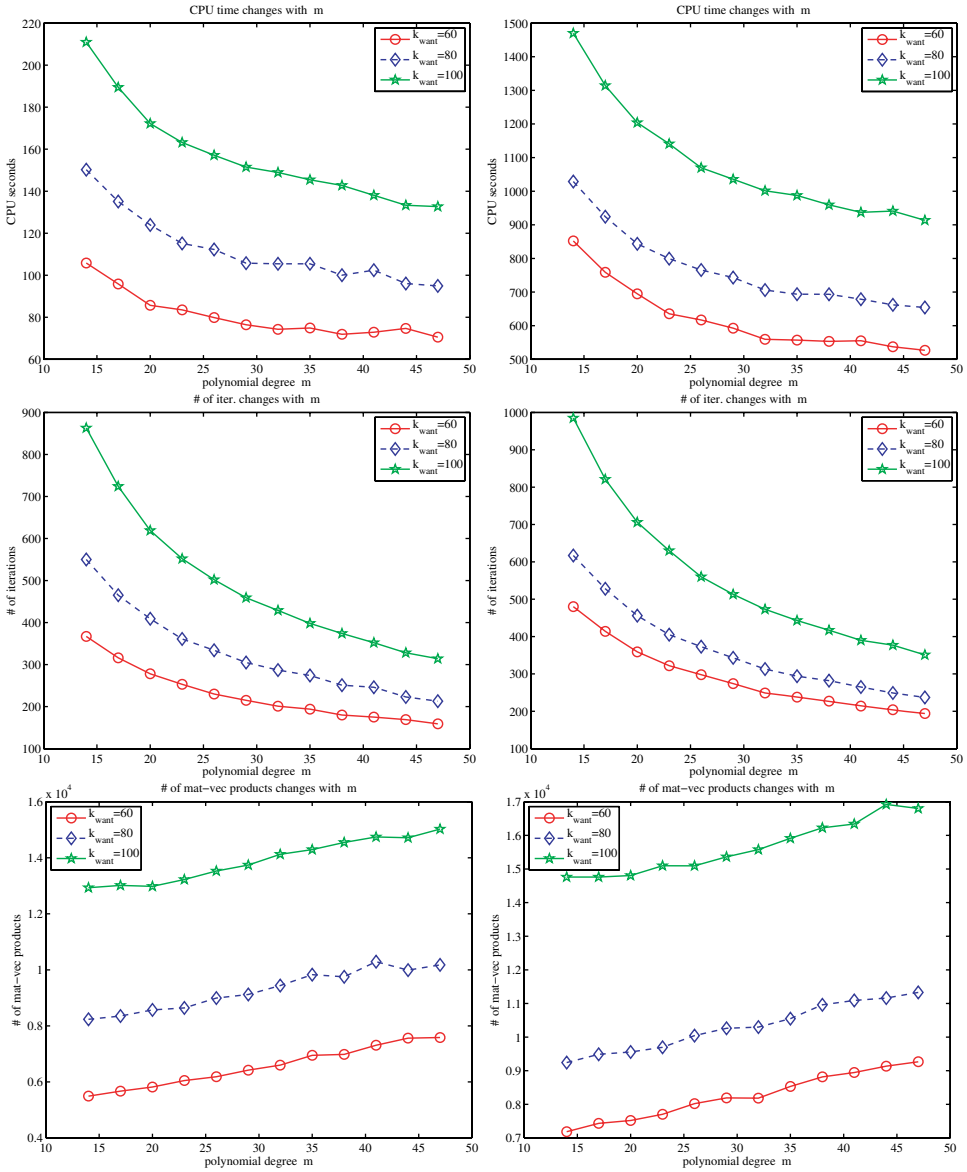

FIG. 6.2. *Changes in CPU time, number of iterations, and number of matrix-vector products with a varying polynomial degree m. Figures on the left are for quantum dot $Si_{10}H_{16}$, with $n = 17077$. Figures on the right are for quantum dot $Si_{34}H_{36}$, with $n = 97569$. The m is varied as $m = 14 : 3 : 47$ in Matlab notation. For each model, the same initial vector $ones(n, 1)$ is used for each m. The number of vectors to keep during restart is simply set as $k_{keep} = 60$ for all these tests. Three cases where $k_{want} = 60, 80, 100$ are demonstrated.*

is proportional to the number of valence electrons in the atoms, which commonly exceed several thousand. In these cases, high memory demand is clearly a concern. Moreover, eigenvalue algorithms that are efficient for exterior eigenvalues can have problems converging for interior eigenvalues.

Table 6.4 shows the dimension of the discretized Hamiltonians and the number of needed eigenpairs for four silicon nanocrystals and two metallic (iron) clusters. The

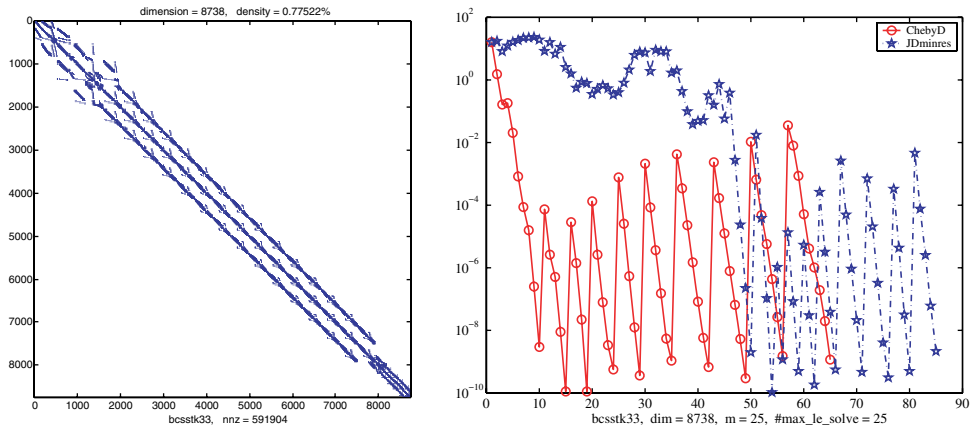

FIG. 6.3. *The matrix* `bcsstk33` *is from the NIST Matrix Market. Structure plot is on the left. On the right is the residual norm plot for the first* 10 *smallest eigenvalues. This shows that ChebyD can have much better global convergence than JDminres. Here* $m = 25$ *for ChebyD and* $\#max\_le\_solve = 25$ *for JDminres. The initial vector used is* $ones(n, 1)$ *for both methods.*

TABLE 6.4

The Chebyshev–Davidson method applied to compute k_{want} *number of eigenvalues and eigenvectors. For the silicon nanocrystals, the polynomial degree is* $m = 17$*; for the iron clusters,* $m = 20$*. The computations are performed on the SGI Altix cluster (*1.6*GHz per processor) at the Minnesota Supercomputing Institute.*

Material	Matrix dimension n	k_{want}	# Processors	CPU hours
$Si_{2713}H_{828}$	1074080	5843	16	7.83
$Si_{4001}H_{1012}$	1472440	8511	16	18.63
$Si_{6047}H_{1308}$	2144432	12751	32	45.11
$Si_{9041}H_{1860}$	2992832	19015	48	102.12
Fe_{326}	2985992	3912	24	11.62
Fe_{360}	3262312	4320	24	16.55

reported CPU time is what the Chebyshev–Davidson method used to finish the first step diagonalization in the self-consistent loop.

Physical significance of the numerical results are discussed in [37, 38]. In [37] we report the largest iron-cluster first principle DFT simulations that have been published. The results are used to clarify a decade-old controversy regarding the dependence of magnetic moment on the size of iron clusters. As to first principles DFT calculations on silicon nanocrystals, previously reported results seem not have gone beyond 2000 atoms; in contrast, we were able to do first principle calculations on a sequence of silicon nanocrystals with up to 10,000 atoms [46, 7].

Although success in these challenging DFT calculations depends more on the nonlinear CheFS method, we must mention that the Chebyshev–Davidson method plays a crucial role in the computations since it provides the CheFS method with a desired initial subspace. A suitable initial subspace can substantially reduce the number of iterations required for the CheFS method to reach self-consistency (convergence).

The other three eigensolvers (Diagla, ARPACK, and TRLan) in PARSEC were also used for computing initial subspaces, but we noticed that they became impractical

for the largest material systems in Table 6.4, in terms of both memory requirement and convergence speed. In comparison, Diagla is quite efficient when n and k_{want} are moderate, but it becomes slower than ARPACK and TRLan when n and k_{want} become large. TRLan is the fastest among these three solvers; it is observed in [47] to be about twice as fast as ARPACK because of the reduced reorthogonalization. But both TRLan and ARPACK require too much memory because of the requirement that the maximum subspace dimension be around $2k_{want}$.

To address the huge memory demand related to standard restart when n and k_{want} are large, we introduced an *inner-outer restart* technique into the Chebyshev–Davidson algorithm. The *outer restart* is the same as the standard restart, but the *inner restart* corresponds to a standard restart restricted to an inner subspace with dimension much smaller than k_{want}. This reduces the maximum dimension of the outer subspace from $2k_{want}$ to k_{want}. Therefore the Chebyshev–Davidson algorithm requires about half the memory required by a method with only standard restart. It did not have a memory requirement problem for all the materials reported in Table 6.4. More details about the *inner-outer restart* may be found in [44].

As to CPU time, we compared the Chebyshev–Davidson method with TRLan on the two smallest nanocrystals in Table 6.4. Using the same number of CPU nodes, TRLan spent 8.65 CPU hours on $Si_{2713}H_{828}$ and 34.99 CPU hours on $Si_{4001}H_{1012}$ for the first step diagonalization. The comparison is not completely fair since we employed an additional trick in the Chebyshev–Davidson routine, which corresponds to a subspace filtering step so that the last few basis vectors are only approximate eigenvectors. The number of these vectors not converged to full accuracy is bounded above by the dimension of the inner subspace used for inner restart. It is rather straightforward to add this subspace filtering step inside a Davidson-type iteration. Both this trick and the *inner-outer restart* are due to the remarkable flexibility of a Davidson-type method in adjusting basis vectors. A Lanczos-type method does not have this flexibility because of the need to keep a Krylov structure. In TRLan all the basis vectors are converged to the same full accuracy, which can be too costly since high accuracy is often not necessary for the last few vectors in the subspace, especially when the diagonalization is performed at the first step of the self-consistent loop to provide an initial subspace.

We also mention that the adaptive Chebyshev filter (based on [26, 27]) and the choice of bounds to achieve efficient filtering, as presented in this paper, are essential to the development of the nonlinear CheFSI method in [47, 46].

7. Conclusion. A Chebyshev–Davidson algorithm has been presented for solving large symmetric eigenvalue problems. It essentially consists of filtering out the unwanted portion of the spectrum by using adaptive Chebyshev polynomials of the matrix. Comparisons with existing Davidson- and Lanczos-type methods show that the Chebyshev–Davidson method is efficient and robust.

Advantages of the Chebyshev filtering approach include not requiring correction-equations (hence no preconditioned linear solves are necessary), and robust global convergence because of the intervalwise filtering. The Chebyshev filters are easily controllable within the Davidson-type framework, and thus they can be conveniently tuned to filter the full spectrum in the desired way to accelerate global convergence.

Acknowledgments. We thank Professor Calvetti for her constructive comments, especially the suggestion on testing the effect of the polynomial degree in the Chebyshev–Davidson algorithm, which improved our understanding of this parameter.

REFERENCES

[1] P. Arbenz, U. L. Hetmaniuk, R. B. Lehoucq, and R. S. Tuminara, *A comparison of eigensolvers for large-scale 3d model analysis using AMG-preconditioned iterative methods*, Int. J. Numer. Methods Engrg., 64 (2005), pp. 204–236.

[2] J. Baglama, D. Calvetti, and L. Reichel, *IRBL: An implicitly restarted block-Lanczos method for large-scale Hermitian eigenproblems*, SIAM J. Sci. Comput., 24 (2003), pp. 1650–1677.

[3] J. Baglama, D. Calvetti, and L. Reichel, *irbleigs: A MATLAB program for computing a few eigenpairs of a large sparse Hermitian matrix*, ACM Trans. Math. Softw., 5 (2003), pp. 337–348.

[4] Z. Bai, J. Demmel, J. Dongarra, A. Ruhe, and H. van der Vorst, eds., *Templates for the Solution of Algebraic Eigenvalue Problems: A Practical Guide*, Software Environ. Tools 11, SIAM, Philadelphia, PA, 2000.

[5] J. H. Brandts, *Solving eigenproblems: From Arnoldi via Jacobi-Davidson to the Riccati method*, in Numerical Methods and Applications, Lecture Notes in Comput. Sci. 2542, Comput. Sci., Springer, New York, 2003, pp. 167–173.

[6] J. Chelikowsky and Y. Saad, *Electronic structure of clusters and nanocrystals*, in Handbook of Theoretical and Computational Nanotechnology, M. Rieth and W. Schommers, eds., American Scientific Publishers, Stevenson Ranch, CA, to appear.

[7] J. R. Chelikowsky, M. L. Tiago, Y. Saad, and Y. Zhou, *Algorithms for the evolution of electronic properties in nanocrystals*, Comp. Phys. Comm., 177 (2007), pp. 1–5.

[8] J. R. Chelikowsky, N. Troullier, and Y. Saad, *Finite-difference-pseudopotential method: Electronic structure calculations without a basis*, Phys. Rev. Lett., 72 (1994), pp. 1240–1243.

[9] J. R. Chelikowsky, N. Troullier, K. Wu, and Y. Saad, *Higher-order finite-difference pseudopotential method: An application to diatomic molecules*, Phys. Rev. B, 50 (1994), pp. 11355–11364.

[10] J. Daniel, W. B. Gragg, L. Kaufman, and G. W. Stewart, *Reorthogonalization and stable algorithms for updating the Gram–Schmidt QR factorization*, Math. Comp., 30 (1976), pp. 772–795.

[11] 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), pp. 87–94.

[12] R. M. Dreizler and E. K. U. Gross, *Density Functional Theory: An Approach to the Quantum Many-Body Problem*, Springer-Verlag, Berlin, 1990.

[13] Y. T. Feng, *An integrated multigrid and Davidson method for very large scale symmetric eigenvalue problems*, Comput. Methods Appl. Mech. Engrg., 190 (2001), pp. 3543–3563.

[14] D. R. Fokkema, G. L. G. Sleijpen, and H. A. van der Vorst, *Jacobi–Davidson style QR and QZ algorithms for the reduction of matrix pencils*, SIAM J. Sci. Comput., 20 (1998), pp. 94–125.

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

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

[17] R. B. Lehoucq, D. C. Sorensen, and C. Yang, *ARPACK User's Guide: Solution of Large-Scale Eigenvalue Problems with Implicitly Restarted Arnoldi Methods*, Software Environ. Tools 6, SIAM, Philadelphia, 1998; software available online at: http://www.caam.rice.edu/software/ARPACK/.

[18] R. M. Martin, *Electronic Structure : Basic Theory and Practical Methods*, Cambridge University Press, Cambridge, UK, 2004.

[19] R. B. Morgan and D. S. Scott, *Generalizations of Davidson's method for computing eigenvalues of sparse symmetric matrices*, SIAM J. Sci. Stat. Comput., 7 (1986), pp. 817–825.

[20] Y. Notay, *Combination of Jacobi-Davidson and conjugate gradients for the partial symmetric eigenproblem*, Numer. Linear Algebra Appl., 9 (2002), pp. 21–44.

[21] C. C. Paige and M. A. Saunders, *Solution of sparse indefinite systems of linear equations*, SIAM J. Numer. Anal., 12 (1975), pp. 617–629.

[22] B. N. Parlett, *The Rayleigh quotient iteration and some generalizations for nonnormal matrices*, Math. Comp., 28 (1974), pp. 679–693.

[23] B. N. Parlett, *The Symmetric Eigenvalue Problem*, Classics in Appl. Math. 20, SIAM, Philadelphia, PA, 1997.

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

[25] H. Rutishauser, *Simultaneous iteration method for symmetric matrices*, in Handbook for Automatic Computation (Linear Algebra), J. H. Wilkinson and C. Reinsh, eds., Springer-Verlag, 1971, vol. II, pp. 284–302.
[26] Y. Saad, *Chebyshev acceleration techniques for solving nonsymmetric eigenvalue problems*, Math. Comp., 42 (1984), pp. 567–588.
[27] Y. Saad, *Numerical Methods for Large Eigenvalue Problems*, John Wiley, New York, 1992.
[28] Y. Saad and M. H. Schultz, *GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems*, SIAM J. Sci. Stat. Comput., 7 (1986), pp. 856–869.
[29] Y. Saad, A. Stathopoulos, J. Chelikowsky, K. Wu, and S. Öğüt, *Solution of large eigenvalue problems in electronic structure calculations*, BIT, 36 (1996), pp. 563–578.
[30] Y. Saad, Y. Zhou, C. Bekas, M. Tiago, and J. Chelikowsky, *Diagonalization methods in PARSEC*, Phys. Status Solidi (B), 243 (2006), pp. 2188–2197.
[31] G. L. G. Sleijpen, A. G. L. Booten, D. R. Fokkema, and H. A. van der Vorst, *Jacobi-Davidson type methods for generalized eigenproblems and polynomial eigenproblems*, BIT, 36 (1996), pp. 595–633.
[32] G. L. G. Sleijpen and H. A. van der Vorst, *A Jacobi–Davidson iteration method for linear eigenvalue problems*, SIAM J. Matrix Anal. Appl., 17 (1996), pp. 401–425.
[33] D. C. Sorensen, *Implicit application of polynomial filters in a k-step Arnoldi method*, SIAM J. Matrix Anal. Appl., 13 (1992), pp. 357–385.
[34] D. C. Sorensen and C. Yang, *Accelerating the Lanczos Algorithm via Polynomial Spectral Transformations*, Technical Report TR97-29, Department of Computational and Applied Mathematics, Rice University, Houston, TX, 1997.
[35] A. Stathopoulos, S. Öğüt, Y. Saad, J. Chelikowsky, and H. Kim, *Parallel methods and tools for predicting materials properties*, Comput. Sci. Eng., 2 (2000), pp. 19–32.
[36] G. W. Stewart, *A Krylov–Schur algorithm for large eigenproblems*, SIAM J. Matrix Anal. Appl., 23 (2001), pp. 601–614.
[37] M. L. Tiago, Y. Zhou, M. Alemany, Y. Saad, and J. R. Chelikowsky, *Evolution of magnetism in iron from the atom to the bulk*, Phys. Rev. Lett., 97 (2006), paper 147201.
[38] M. L. Tiago, Y. Zhou, Y. Saad, and J. R. Chelikowsky, *Electronic Properties and Energetics of Nanometer-size Silicon Nanocrystals*, Technical report, ICES, University of Texas/Austin, Austin, TX, in preparation.
[39] D. S. Watkins and L. Elsner, *Convergence of algorithms of decomposition type for the eigenvalue problem*, Linear Algebra Appl., 41 (1991), pp. 19–47.
[40] K. Wu, A. Canning, H. D. Simon, and L.-W. Wang, *Thick-restart Lanczos method for electronic structure calculations*, J. Comput. Phys., 154 (1999), pp. 156–173.
[41] K. Wu, Y. Saad, and A. Stathopoulos, *Inexact Newton preconditioning techniques for large symmetric eigenvalue problems*, Electron. Trans. Numer. Anal., 7 (1998), pp. 202–214.
[42] K. Wu and H. Simon, *Thick-restart Lanczos method for large symmetric eigenvalue problems*, SIAM J. Matrix Anal. Appl., 22 (2000), pp. 602–616.
[43] Y. Zhou, *Studies on Jacobi-Davidson, Rayleigh quotient iteration, inverse iteration generalized Davidson and Newton updates*, Numer. Linear Algebra Appl., 13 (2006), pp. 621–642.
[44] Y. Zhou, *A Block Chebyshev–Davidson Method with Inner-Outer Restart for Large Eigenvalue Problems*, Technical report, Department of Mathematics, Southern Methodist University, Dallas, TX, in preparation.
[45] Y. Zhou and Y. Saad, *Block Krylov-Schur Method for Large Symmetric Eigenvalue Problems*, Technical report 2004/215, Minnesota Supercomputing Institute, University of Minnesota, 2004.
[46] Y. Zhou, Y. Saad, M. L. Tiago, and J. R. Chelikowsky, *Parallel self-consistent-field calculations using Chebyshev-filtered subspace acceleration*, Phys. Rev. E, 74 (2006), paper 066704.
[47] Y. Zhou, Y. Saad, M. L. Tiago, and J. R. Chelikowsky, *Self-consistent-field calculation using Chebyshev-filtered subspace iteration*, J. Comput. Phys., 219 (2006), pp. 172–184.