A CHEBYSHEV-DAVIDSON ALGORITHM FOR LARGE SYMMETRIC EIGENPROBLEMS [*]

YUNKAI ZHOU [†] AND YOUSEF SAAD [†]

Abstract.
A polynomial filtered Davidson-type algorithm is proposed for solving symmetric eigenproblems. 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. A typical filter, one that is 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 the number of steps required for convergence and the cost resulting from orthogonalizations and restarts. Comparisons with JDQR, JDCG and LOBPCG methods are presented, as well as comparisons with the well-known ARPACK package.

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

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

1. Introduction.

This paper considers solution methods for the standard symmetric eigenvalue problem

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

where $A \in \mathbb{R}^{n\times n}$ and n is large. We consider a class of methods which can be viewed as accelerated, or preconditioned, versions of Krylov subspace techniques. Another viewpoint is to consider these methods as Davidson-type methods in which the correction equation is replaced by a certain filtering step. The goal is to improve global convergence and robustness of Davidson-type methods.

The original Davidson method [8] was initially designed for handling diagonally dominant matrices. We note that for eigenvalue problems, diagonal dominance means that the off-diagonal elements are small compared with the changes in magnitude between diagonal elements [15]. The Davidson approach gives up the attractive Krylov subspace structure at the cost of having to compute eigenpairs and associated residual vectors of a projection matrix, at each step. The trade-off is that the Davidson approach can be more efficient by being able to augment the subspace by a new vector that can potentially be 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 linear equation solvers, often in the preconditioned forms, are often utilized to solve the correction-equations. In the original Davidson approach, the correction equation uses only the diagonal of A. Later this was generalized, and in [15] by using better approximations of A. However, it was noted in [15], that using the exact preconditioner $(A - \lambda I)^{-1}$ is counter-productive as it leads to stagnation. This led to the development of the more efficient Jacobi-Davidson (JD) algorithm [26, 10, 25]. Work in

*Work supported by U.S. Department of Energy under contract DE-FG02-03ER25585, and by the Minnesota Supercomputing Institute.

†Dept. of Computer Science & Eng., University of Minnesota, Minneapolis, MN 55455, USA (zhou@msi.umn.edu, saad@cs.umn.edu).

the literature has shown that the Jacobi-Davidson approach can be competitive with efficient Krylov subspace methods such as those in [27, 14, 29, 32, 34]. In [31] other Davidson-type methods are derived, where the correction equations come from applying Newton's method to some transformed formulations of (1.1).

In this paper, we explore a different approach for Davidson-type methods which does not explicitly solve a correction-equation. Though we do not need to form or solve any correction-equations, we compute a polynomial of A times an approximate eignvector. One can also solve the correction equation in the Davidson approach by using Chebyshev iteration, see, e.g., [22], or any Krylov subspace iterative method. However, we note that choice of the right hand side vector in the correction-equation is very critical to the efficiency of the procedure. The algorithm proposed here applies Chebyshev polynomial to an approximate eigenvector instead of the residual vector. Therefore, it can be viewed from the angle of accelerated Krylov-subspace methods.

The JD method can be related to the Newton's 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 [18] [19, 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 [4] a non-linearized JD correction-equation is proposed, which leads to some nice improvements in global convergence. The drawback is that the preconditioning may be difficult to apply for the non-linearized correction-equation. Moreover, the approach can become much more expensive than JD when the number of wanted eigenpairs is large. Another approach to achieve better global convergence is to apply an Arnoldi or Lanczos method to get a good initial vector and then apply JD. This is suggested in [15, 10].

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 one is on the well-known polynomial filtering argument: For a symmetric matrix A with the eigen-decomposition $A = Q\Lambda Q^T$, 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 [33] 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. A few additional details are now provided. Assume throughout that (μ, x) denote the current Ritz pair that best approximates a wanted eigenvalue, the Ritz vector x is of unit length; and let $r = Ax - \mu x$ denote the residual. It was observed in [33] that the Jacobi-Davidson 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 as

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

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 JD. 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 non-zero scaler. We mention that this 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).

Bearing in mind polynomial filtering and recalling the relation (2.1), our 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 instead 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 the correction-equation is often solved by a shift-invert approach, 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. We point out that this useful filtering idea has long been exploited to accelerate the Arnoldi and Lanczos algorithms, see e.g. [27].

To further explain the third observation, we suppose that the eigenvalues in $diag(\Lambda)$ 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 i-th 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. Even though a low degree polynomial approximating (2.5) is hard to get, the above claim can be verified by explicitly computing Q and using $\sum_{i=1}^k \alpha_i q_i$, $(\alpha_i = q_i^T x)$ as the augmentation vector in a Davidson-type method, where x denotes the current Ritz vector at each iteration. This experiment is equivalent to using a filter that exactly approximates (2.5). It is surprising to note that even though this filter leads to no gap among wanted eigenvalues, it can still achieve fast convergence in a Davidson-type method. This experiment confirms that in order to obtain fast convergence, the basis should be augmented by vectors close to the wanted invariant subspace.

We note that the initial vector v need not be a good approximation to a wanted eigenvector in order to achieve fast convergence. Instead, the essence is to construct a suitable filter ψ so that $\psi(A)v$ 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 the ones in [9], 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, JD 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 a fixed preconditioner and rather inaccurate solves, [1] shows that equation $(A - \mu I)t = r$ can have better performance than other correction equations.

3. Chebyshev Polynomial Filter.

Throughout this paper we assume that the wanted eigenvalues are the smallest ones. 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 polynomial are a natural choice for this task. Using Chebyshev polynomial to accelerate symmetric eigenvalue computations dates back to [20, 21]. A nice discussion on Chebyshev accelerated subspace iteration can be found in [19, pp.329-330], where it is mentioned that the Lanczos method will usually be better than the Chebyshev accelerated (fixed dimension) subspace iteration algorithm. Here we integrate Chebyshev filtering into a Davidson-type algorithm.

Note that in most implementations of the Chebyshev-based subspace iteration [20, 21] the computational problem addressed is that of computing eigenvalue with largest modulus. In this case, a symmetric interval of the form $[-e, e]$ is used (requiring one parameter for the interval instead of two). The technique can easily be adapted to the problem of computing eigenvalues with largest or smallest real part (see e.g., [22] for the nonsymmetric case). The Chebyshev polynomials to be used in a Davidson-type method are based on an interval $[a, b]$ (two parameters) where the eigenvalues to be damped are located.

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. Hence the filtering approach is likely to lead to substantial savings in (re-)orthogonalization costs.

Recall that the real Chebyshev polynomials of the first kind are defined by (see e.g. [19, p.371] [23, 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 3-term recurrence,

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

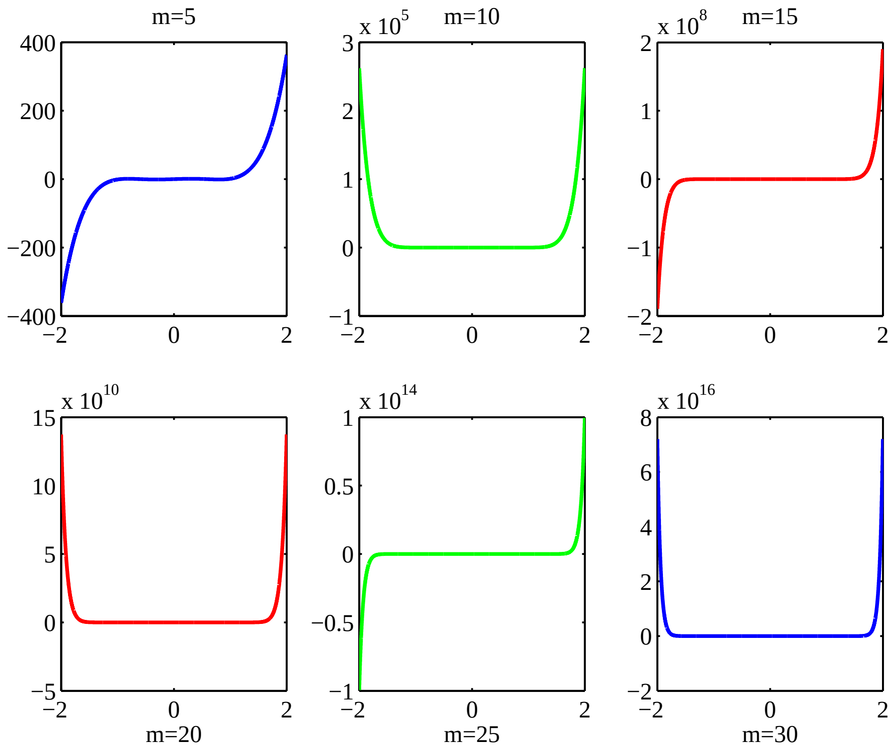

FIG. 3.1. *Rapid increase outside* $[-1, 1]$ *of Chebyshev polynomial of degree* m.

A remarkable property of the Chebyshev polynomial is its rapid growth outside the $[-1, 1]$ interval. This property is illustrated in Figure 3.1. We only plot the $[-2, 2]$ interval, but note that the farther away from -1 or 1, the larger the absolute value of the Chebyshev polynomial. 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 only need to map $[a, b]$ into $[-1, 1]$ by a linear 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. We do not need the global lower bound a_0 which may be hard to estimate. Hence using Chebyshev filtering within a Davidson-type method is practical. The upper bound b of the eigenvalues of A can by obtained by Gershgorin's theorem. It can also be estimated by an initial application of the Lanczos algorithm, though care must be exercised because b must be an upper bound of the full spectrum of A.

The Chebyshev iteration, whose goal is to dampen values in $[a, b]$ while magnifying values in the interval to the left of $[a, b]$ is presented in Algorithm 3.1. Here we follow the formula derived in [23, p. 223] for the complex Chebyshev iteration and adapt it to the real case. The iteration of the algorithm is equivalent to computing

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

As defined in the algorithm, c is the center of the interval and e its half-width.

```
ALGORITHM 3.1. [y] = Chebyshev_filter(x, m, a, b).
Filter x by an m degree Chebyshev polynomial that dampens on [a, b].
Input: vector x, degree m and interval end points a, b.
Output: the filtered vector y.
  1.   $e = (b - a)/2$;
  2.   $c = (b + a)/2$;
  3.   $\sigma = e/(a - c)$;
  4.   $\sigma_1 = \sigma$;
  5.   $y = (Ax - cx)\sigma_1/e$;
  6.   For $i = 2 : m$
  7.       $\sigma_{new} = \frac{1}{(2/\sigma_1 - \sigma)}$;
  8.       $y_{new} = 2(Ay - cy)\sigma_{new}/e - \sigma\sigma_{new}x$;
  9.       $x = y$;
 10.       $y = y_{new}$;
 11.       $\sigma = \sigma_{new}$;
 12.   End For
```

It is clear that polynomials other than Chebyshev which have the property to dampen one interval and significantly magnify another interval can be used for filtering. Several polynomials are discussed in [28], where the concern is on approximating rational functions like $\varphi(s) = 1/(s - \mu)$. Chebyshev polynomials were selected because of their suitable filtering properties, and their ease of implementation. Chebyshev acceleration for nonsymmetric eigenproblems was discussed in [22]. Note that the techniques in [22] may also be adapted into a Davidson-type method. 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.

We first present the Chebyshev polynomial filtered Davidson method in Algorithm 4.1. We point out that the pseudo code for Davidson-type methods is very different from those in the literature (e.g. [2]). 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 *(h).6*) by the indexing. Our implementation does not require extra basis updates or memory copies during the restart, since the updates in step *(h).7* need to be performed even when restart is not necessary. We note that putting restart at step *(h).6* is better than putting it at the end of the outer loop, because it saves operations in step *(h).7* when restarting is necessary.

We make a few comments on Algorithm 4.1. Comments V–VII are related to the robust implementations of any Davidson-type methods.

I. It is important that the upper bound `upperb` bounds all eigenvalues of A. Otherwise the interval containing largest eigenvalues may also be magnified through filtering and this can drastically slow convergence. Step *(e)* provides one choice that is not expensive, but we note that any `upperb` $\geq \max_i(\lambda_i(A))$ should work well.

II. The choice of the lower bound for the unwanted interval at each iteration is by a heuristic. This is actually one of the most critical ingredients of the procedure.

Algorithm 4.1. *Chebyshev filtering Davidson-type method for the eigenvalue problem (1.1), computing the k_{want} 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 non-increasing order, and their corresponding eigenvectors $V(:,1:k_c)$, where k_c denotes # of converged eigenpairs.*

```
(a) Start with the unit vector $x$, $V = [x]$.
(b) Compute $W = [Ax]$, $H = [\mu]$ where $\mu = x^T w$.
(c) Compute the residual vector: $r = W(:,1) - \mu x$.
(d) If $\|r\| <= \tau$, set $eval(1) = \mu$, $k_c = 1$, $H = [\ ]$; Else, set $k_c = 0$.
(e) Estimate the upper bound of eigenvalues as: $\texttt{upperb} = \|A\|_1$.
(f) Set $\texttt{lowerb} = (\texttt{upperb} - \mu)/2$.
(g) Set $k_{sub} = 1$ ($k_{sub}$ stores the current subspace dimension).
(h) Outer Loop: Do while ( $iter \le iter_{max}$ )
     1. Call the Chebyshev polynomial filter:
        $[t] = \texttt{Chebyshev\_filter}(x, m, \texttt{lowerb}, \texttt{upperb})$.
     2. Orthonormalize $t$ against $V(:,1:k_{sub})$ to get a unit vector $V(:,k_{sub}+1)$;
        set $k_{sub} \leftarrow k_{sub} + 1$; set $k_{old} = k_{sub}$.
     3. Compute $W(:,k_{sub}) = AV(:,k_{sub})$.
     4. Compute the last column of the symmetric Rayleigh-Quotient matrix $H$:
        $H(1:k_{sub}-k_c, k_{sub}-k_c) = V(:,k_c+1:k_{sub})^T W(:,k_{sub})$.
     5. Compute the eigen-decomposition of $H$: $HY = YD$,
        where $diag(D)$ is in non-increasing order. Set $\mu = D(1,1)$.
     6. If ($k_{sub} \ge dim_{max}$) then restart: set $k_{sub} = k_c + k_{keep}$.
     7. Update basis: $V(:,k_c+1:k_{sub}) \leftarrow V(:,k_c+1:k_{old})Y(:,1:k_{sub}-k_c)$;
        update $W$: $W(:,k_c+1:k_{sub}) \leftarrow W(:,k_c+1:k_{old})Y(:,1:k_{sub}-k_c)$.
     8. Compute the residual vector: $r = W(:,k_c+1) - \mu V(:,k_c+1)$.
     9. Set $noswap = 0$, $iter \leftarrow iter + 1$.
    10. Test for convergence: If $\|r\| <= \tau\ max(diag(D))$, 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 non-increasing order;
        set $noswap = 1$ if any swap happens.
    11. If ($k_c \ge k_{want}$ and $noswap == 0$), Return $eval(1:k_c)$ and $V(:,1:k_c)$
        as the converged wanted eigenpairs. Exit.
    12. Update the lower bound: $\texttt{lowerb} = median(diag(D))$.
    13. Set the next Ritz vector for filtering: $x = V(:,k_c+1)$.
    14. Update $H$: $H = D(k_c+1:k_{sub}, k_c+1:k_{sub})$.
    End Do
```

Numerical results show that the current heuristic at step *(h).12*, i.e. "$\texttt{lowerb} =$ median of the current Ritz values", is efficient. Better choice based on more sophisticated analysis may lead to faster convergence.

III. For the orthogonalization step *(h).2*, we use the iterated Gram-Schmidt method [7] often referred to as DGKS.

IV. The refinement at step *(h).7* is performed at each step. One can avoid this step until some eigenpair converges. But according to [19, p. 325], this refinement is necessary in order to have faster convergence for the eigenvectors.

V. The swap at step *(h).10* may be performed by the following pseudo code:

```
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 more careful programming can save unnecessary memory copies in the above.)

VI. The *noswap* flag at steps *(h).9–11* is used to improve robustness. This flag lowers the possibility of counting converged unwanted eigenvalues as wanted ones.

VII. At step *(h).10*, 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 convergence of more than one Ritz pairs. 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 very high possibility missing wanted eigenvalues.

`Remark:` The Algorithm 4.1 essentially contains the framework for Davidson-type methods based on polynomial filtering. One only needs to replace the Chebyshev polynomial filter at step *(h).1* by other suitable polynomials having the desired filtering property as described in Section 2.

5. Analysis.

It is generally difficult to establish the convergence of a restarted method, especially in a non-Krylov subspace setting. Convergence analysis for restarted Krylov methods may be found in [27, 5, 13, 3].

Suppose the eigenvalues of A are labeled increasingly, i.e., $\lambda_1 < \lambda_2 \leq ... \leq \lambda_n$, and let their associated unit eigenvectors be $q_1, \ldots, q_n$. Assume that Algorithm 4.1 is carried out in a *single vector* version. That is, in step *(h).2* we keep $k_{sub} \equiv 1$ and set $V(:, 1) = t/\|t\|$, and in step *(h).13* we set $x = V(:, 1)$. According to the expression (3.2) of the polynomial used in the Chebyshev procedure, and the fact that the interval of the eigenvalues to be dampened at each step is adaptively changing, we see that the matrix applied at the j-th step is

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

For simplicity, assume that the interval of the eigenvalues to be dampened at each step is fixed. Then, the algorithm is simply the standard power method with the matrix $p_m(A) \equiv C_m((A - cI)/e)$. As a result the convergence will be governed by the ratio of the two dominant eigenvalues. Note that the interval $[a, b]$ of ther 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 when we do two steps of the algorithm, i.e., the dimension of the subspace is two. As it turns out the resulting method has a simple interpretation. The first vector of the basis is simply $p_m(A)x$. The second is obtained as $p_m(A)x_1$ where x_1 is an approximate eigenvector from the 1-dimensional space spanned by the the first vector, which is simply a multiple of $p_m(A)x$. So the subspace used in this case, is

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

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 on the subspace K_2, and it is a linear combination of vectors from K_2, so we can write $x_2 = \alpha_1 p_m(A)x + \alpha_2 p_m(A)^2 x$. What is remarkable is that K_3 the new subspace is again a Krylov subspace. Indeed,

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

This can be easily seen by re-writing the vector injected to the basis as

$$p_m(A)x_2 = p_m(A)[\alpha_1 p_m(A)x + \alpha_2 p_m(A)^2 x] = \alpha_1 p_m(A)^2 x + \alpha_2 p_m(A)^3 x \ .$$

So the linear span of the 3 vectors $p_m(A)x$, $p_m(A)^2x$, and $p_m(A)x_2$, is identical with that of the vectors $p_m(A)x$, $p_m(A)^2x$, and $p_m(A)^3x$. The result can be easily extended to an arbitrary step j and that it is true as long as the polynomial does not change (i.e., the interval of the eigenvalues to be damped is fixed).

PROPOSITION 5.1. *Step j of Algorithm 4.1 without restart is mathematically equivalent with 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 generates an orthogonal basis V_j of the Krylov subspace $K_j(p_m(A), x)$ and computes the eigenvalues of $V_j^T A V_j$, these eigenvalues would be identical with those of Algorithm 4.1, under the restriction that the polynomial does not change. 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 to analyse the (restricted) 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 [19] for details.

Although the restricted algorithm can be viewed from the angle of Krylov subspaces, there are a number of distinguishing features, related to implementation and other practical aspects. For example, the above proposition assumes that the polynomial is fixed but being able to adapt the polynomial is a crucial ingredient of the procedure. The unrestricted Algorithm 4.1 adapts the polynomial by dynamically selecting the bounds of the interval of eigenvalues to be dampened at each iteration. Taking (5.1) into account, we see that

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

$$K_3 = \text{span}\{p_m^{(1)}(A)x, p_m^{(2)}(A)p_m^{(1)}(A)x, p_m^{(3)}(A)p_m^{(1)}(A)x, p_m^{(3)}(A)p_m^{(2)}(A)p_m^{(1)}(A)x\} \ .$$

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 is $\Phi_j(A)x$, and this term 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 [30, 13]. This rate can be considerably faster than that obtained from using a fixed 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 Discussions.

6.1. Comparisons in Matlab. In this section, we compare the Chebyshev-Davidson method (denoted as ChebyD) with several other Davidson-type methods, mostly in the non-preconditioned case and also in the preconditioned case when the test matrices are positive definite. We also make some comments on preconditioned eigensolvers based on the observed numerical results.

For the JD method we use the publicly available Matlab codes JDQR [10] [*] and JDCG [16][†]. The JDCG code is tuned for symmetric eigenproblems, the linear solver used for JDCG is (preconditioned) CG. The JDQR code can solve both symmetric and nonsymmetric problems; GMRES[24] is the default linear solver that JDQR supplies and we use it for the numerical tests. Since we solve symmetric eigenproblems, it is less costly to use MINRES[17] in JD. 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 the purpose of further comparisons, we implemented our own JD code using MINRES as the linear solver. This code is denoted as JDminres. JDminres is mainly based on Algorithm 4.1 except that step *(h).1* is replaced by a linear equation solve using MINRES. The preconditioned eigensolver LOBPCG [11] [‡] is also used for comparison because it is a representative preconditioned eigensolver.

All the Matlab numerical experiments are performed on a Dell PC with dual Intel Xeon 2.66GHz CPU and 1GB RAM. One of the CPU is dedicated to the computation. The OS is Debian Linux with Linux kernel version 2.4.27. We use Matlab version 7.0 (R14) for the computations.

For all of the test examples listed in Table 6.1—6.8, we compute the $k_{want} = 50$ 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 accuracy is reported as $\|AV - VD\|/\|A\|_1$, where V contains the fifty converged eigenvectors, and the diagonal of D contains the fifty corresponding eigenvalues. The relative convergence tolerance is set to 10^{-10} for all methods. In order for these results to be reproducible, we use $ones(n, 1)$ as the initial vector throughout. This is also to avoid biases due to more or less favorable initial directions. We note that LOBPCK requires another $k_{want} - 1$ vectors for the initial block, and we used random vectors for these.

For the symmetric positive definite Laplaceans in Tables 6.1–6.4, we use the incomplete Cholesky factor $R = cholinc('A', 0)$ as the preconditioner. For each method, an added suffix (p) denotes using the incomplete Cholesky factor as the preconditioner, while the suffix (np) means no preconditioner is used.

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

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

[‡]Code available at: `http://www-math.cudenver.edu/~aknyazev/software/CG/toward/lobpcg.m`

Method	CPU (sec.)	#iter.	#mvp	$\|AV - VD\|/\|A\|_1$
ChebyD	380	434	13424	1.17e-11
JDminres	664	482	13469	5.75e-12
JDQR	1415	2449(-)	13761	2.36e-12
JDCG (p)	1877	269	12625(+)	1.58e-11
JDCG (np)	1180	444	40349	1.40e-11
LOBPCG(p)	1995	685	–	1.25e-11
LOBPCG(np)	7904	5193	–	1.00e-10

TABLE 6.1
2D-Laplacean on a L shape square constructed by A = delsq(numgrid('L', 250)). dim=46128, m=30 for ChebyD, #max_le_solve=25 for JDminres.

Method	CPU (sec.)	#iter.	#mvp	$\|AV - VD\|/\|A\|_1$
ChebyD	150	324	10014	1.51e-11
JDminres	275	378	10557	1.81e-11
JDQR	652	1875(-)	10425	2.12e-12
JDCG (p)	784	241	9797(+)	1.53e-11
JDCG (np)	394	362	23170	1.48e-11
LOBPCG(p)	805	419	–	5.43e-11
LOBPCG(np)	1617	1397	–	4.01e-09

TABLE 6.2
2D-Laplacean on a nested dissection ordering of the square constructed by $A = delsq(numgrid('N', 160))$. dim=24964. #max_le_solve=25 for JDminres, m=30 for ChebyD.

For the symmetric indefinite problems in Tables 6.5—6.8, our experiments showed that JDCG does not work with preconditioners from incomplete LU decomposition. As for LOBPCG, using incomplete LU factors as preconditioners took far more CPU time than without preconditioning, and the results are not always correct for the preconditioned case (we note that LOBPCG is primarily developed for SPD problems). Hence for indefinite problems, we do not use the preconditioned version of JDCG and LOBPCG. Without preconditioning, both JDCG and LOBPCG converged to correct eigenpairs. These experiments also support the point that for situations where preconditioners are hard to obtain, approaches that do not rely on correction-equations have a clear advantage.

In each table, '#iter' counts the total number of the outer loop, '#mvp' the number of matrix-vector products, and '#max_le_solve' shows the maximum inner iteration number for the linear equation solve by MINRES in JDminres. We were not able to find how to output '#mvp' from `help lobpcg`, hence this value is not reported for LOBPCG. For JDQR, `help jdqr` indicates where #iter and #mvp are stored, but the observed output values from history(:,2) for #iter seems 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 as reference, and put a (-) sign to signal differences. For all the examples, the CPU time of JDminres is smaller than that of JDQR.

Note that for the JDCG(p) case, #mvp does not count the number of preconditioned equation solves, hence even though JDCG(p) has lower #iter and #mvp,

Method	CPU (sec.)	#iter.	#mvp	$\|AV - VD\|/\|A\|_1$
ChebyD	277	234	7224	1.38e-11
JDminres	590	332	7614	1.15e-11
JDQR	1096	1270(-)	7002	1.37e-12
JDCG (p)	2018	207	6802(+)	1.02e-11
JDCG (np)	798	292	18025	9.94e-12
LOBPCG(p)	1354	168	–	8.18e-12
LOBPCG(np)	2329	580	–	9.90e-11

TABLE 6.3

3D-Laplacean on a unit cube with grid $(45 \times 30 \times 50)$, *dim=67500. m=30 for ChebyD, #max_le_solve=25 for JDminres.*

Method	CPU (sec.)	#iter.	#mvp	$\|AV - VD\|/\|A\|_1$
ChebyD	503	1967	60947	2.53e-12
JDminres	477	1395	39033	2.12e-12
JDQR	2120	12499(-)	69386	4.27e-12
JDCG (p)	40.9	113	1018(+)	8.40e-11
JDCG (np)	4788	1435	242308	2.70e-11
LOBPCG(p)	28.9	27	–	1.59e-11
LOBPCG(np)	3282	1977	–	1.21e-07

TABLE 6.4

1D-Laplacean, dim=12500, m=30 for ChebyD, #max_le_solve=25 for JDminres. Note that each (p) is more than 100 times faster than its (np) counterpart. This is an excellent example to demonstrate how important the efficiency of a preconditioner can be for a preconditioned eigensolver. We also note that because of the very unfavorable spectrum of this sample for CG, the JDCG(np) is more than twice slower than JDQR even though the former is tuned for SPD problems.

Method	CPU (sec.)	#iter.	#mvp	$\|AV - VD\|/\|A\|_1$
ChebyD	115	243	5083	2.45e-11
JDminres	199	357	6409	5.21e-11
JDQR	293	987(-)	5294	3.70e-13
JDCG (np)	253	355	11882	3.12e-12
LOBPCG(np)	571	390	–	1.08e-10

TABLE 6.5

Silicon quantum dot model $Si_{10}H_{16}$*, indefinite. dim=17077. m=20 for ChebyD, #max_le_solve=20 for JDminres.*

Method	CPU (sec.)	#iter.	#mvp	$\|AV - VD\|/\|A\|_1$
ChebyD	995	336	7036	2.82e-11
JDminres	1273	326	7476	2.61e-11
JDQR	1924	1227(-)	6714	3.67e-13
JDCG (np)	2230	424	18488	2.78e-12
LOBPCG(np)	8485	1926	–	1.04e-10

TABLE 6.6

Silicon quantum dot model $Si_{34}H_{36}$*, indefinite. dim=97569. Structure plot shown in Figure 6.1. m=20 for ChebyD, #max_le_solve=20 for JDminres.*

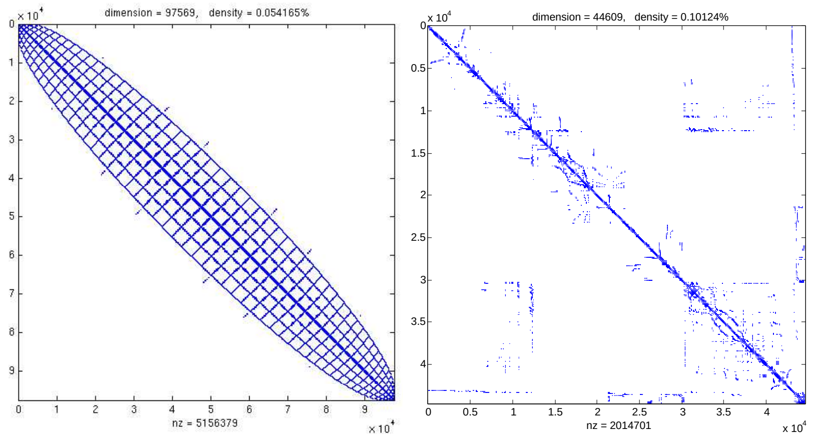

FIG. 6.1. *Structure plots of two test matrices $Si_{34}H_{36}$ and* `bcsstk32`.

Method	CPU (sec.)	#iter.	#mvp	$\|AV - VD\|/\|A\|_1$
ChebyD	270	325	6805	1.97e-11
JDminres	339	323	5797	1.87e-11
JDQR	457	852(-)	4505	8.11e-14
JDCG (np)	703	318	11642	7.03e-13
LOBPCG(np)	2202	1824	–	9.51e-11

TABLE 6.7
`bcsstk31` *from the NIST Matrix Market, dim=35588, indefinite. m=20 for ChebyD, #max_le_solve=15 for JDminres.*

it does not usually win in CPU time because of the additional cost of the preconditioned equation solves. We use (+) to denote that extra #mvp's are required for preconditioned equation solves. In all the results reported here, the preconditioned version only wins for the 1D-Laplacean case (Table 6.4), which in this case shows a drastic superiority. This is because constructing the $R = cholinc('A', 0)$ is easy in this case, and the preconditioned equations are trivial to solve because R is of bandwidth one. For other cases, constructing the preconditioner and solving the preconditioned equations can be expensive. We did not experiment with other preconditioners, but the results in Table 6.1–6.4 clearly show that preconditioned eigensolvers need very good preconditioners to be effective. When efficient preconditioners are available, preconditioned eigensolvers can significantly accelerate convergence.

The reported results of ChebyD are typical for the method proposed in this paper. For all the test problems, we did not fine tune m, we just selected a value that is reasonable. The results already show that the Chebyshev polynomial filtered method outperformed other Davidson-type methods, especially in the non-preconditioned setting. As to robustness, our experience is that JDCG is more likely to miss eigenvalues than other methods compared in this paper. It is also worth mentioning that for a Davidson-type method based on solving correction-equations, the preconditioned version can often be more robust than its non-preconditioned counterpart.

Regarding global convergence, Figure 6.2 shows one example where convergence of Chebyshev-Davidson is much faster than that of the standard JD. However, we would like to mention that for symmetric eigenproblems, JD often has good global

Method	CPU (sec.)	#iter.	#mvp	$\|AV - VD\|/\|A\|_1$
ChebyD	387	320	6700	2.21e-11
JDminres	508	341	6112	2.45e-11
JDQR	624	840(-)	4467	7.59e-14
JDCG (np)	609	308	10925	6.24e-13
LOBPCG(np)	1285	480	–	9.63e-11

Table 6.8
`bcsstk32` *from the NIST Matrix Market, dim=44609, indefinite. Structure plot shown in Figure 6.1. m=20 for ChebyD, #max_le_solve=15 for JDminres.*

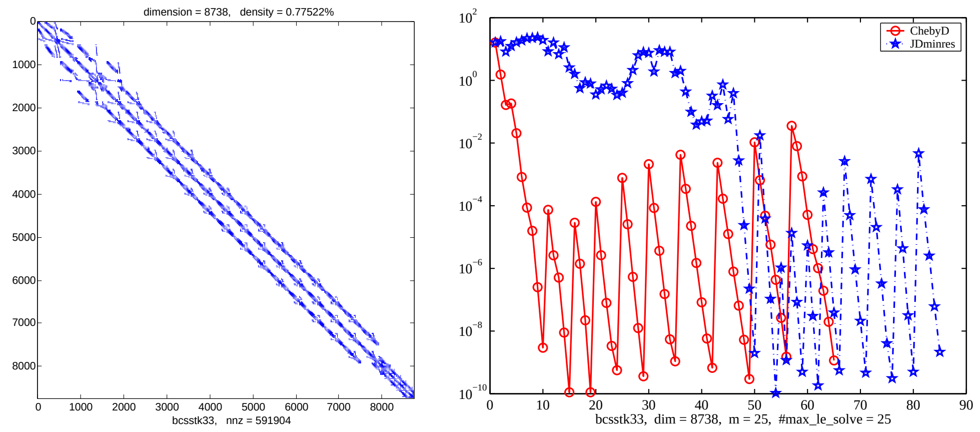

Fig. 6.2. *The matrix* `bcsstk33` *is from the NIST Matrix Market. Structure plot is on the left. On the right is the residual norm plot for the first 10 smallest eigenvalues. This shows ChebyD can have much better global convergence than JDminres. Here m = 25 for ChebyD and #max_le_solve=25 for JDminres. The initial vector used is ones$(n, 1)$ for both methods.*

convergence. For the same example as in Figure 6.2, a different value of #max_le_solve for JDminres can make the global convergence of ChebyD and JDminres become similar.

We made two other observations based on the numerical results. The first one is that a smaller #iter or #mvp count does not necessarily lead to a smaller CPU time. Some methods may require substantial additional work not related to matrix-vector products. The second one is that the CG based (preconditioned) eigensolvers are often not very efficient for indefinite eigenproblems.

6.2. Comparisons in FORTRAN. The Chebyshev-Davidson method was also implemented in FORTRAN95 and we denote by ChebyD95 the corresonding code. In the following we present some comparisons with ARPACK [14] which is one of the best eigenvalue packages available. Note that we only compare with the subroutines (`dsaupd` and `dseupd`) for symmetric eigenproblems in ARPACK. The test examples are 3D Laplaceans and one Hamiltonian from a problem in electronic structures. The FORTRAN numerical tests are performed on the IBM power4 supercomputer running AIX 5.2 of the Minnesota Supercomputing Institute. We use one processor of the 8-processor 1.7 GHz p655 node sharing 16 GB of memory. All the FORTRAN codes are compiled by the IBM FORTRAN compiler `xlf95_r` using optimization flags: `-O4 -qstrict -qmaxmem=-1 -qtune=pwr4 -qarch=pwr4`. The BLAS/LAPACK used is the vendor provided ESSL library.

In order to develop an efficient and competitive FORTRAN code, we implemented a block version of Algorithm 4.1. Moreover, a technique called *Inner-Outer-Restart* is exploited. In this technique, we set the maximum dimension (denoted as act_{max}) of the active subspace to a much smaller value than the allowed maximum subspace dimension (dim_{max}). The inner restart is performed when the active subspace dimension exceeds the act_{max}. The outer-restart, which is the restart in the standard sense, is performed when the total subspace dimension exceeds dim_{max}. This *Inner-Outer-Restart* does not require the array W in Algorithm 4.1 to be of the same dimension as V, we only need W to be of dimension act_{max}. The price paid for this technique is more memory copies (or additional deflation) for the vectors in W, while the gain is a much smaller memory requirement for our method than the standard restarted Lanczos method. We note that essentially it is the polynomial filtering technique presented in this paper that makes this *Inner-Outer-Restart* competitive. This technique is generally not efficient within a standard Lanczos-type method. More details, along with other techniques including ways to exploit a large number of good initial vectors, may be found in [35].

Table 6.9 presents a test run for a relatively small 3D Laplacean on a 40x40x40 mesh. We compute $k_{want} = 400$ smallest eigenpairs. Note that some eigenvalues with multiplicity six or three (especially those at the higher end of the first k_{want} eigenvalues) may be missed if the tolerance for ARPACK is set too low; so we set the tolerance of ARPACK to 10^{-15}. While for ChebyD95, a tolerance around 10^{-9} is enough to capture all the eigenvalues with multiplicity six and three, even when the *block_size* is set to one. This is yet another confirmation of the superior robustness of the Chebyshev-Davidson method. For the numerical run, we set 10^{-10} as the tolerance for ChebyD95, but we note that 10^{-9} is already much smaller than the discretization error.

It is common knowledge that ARPACK performs most efficiently when dim_{max} is around $2 * k_{want}$, (except when k_{want} is quite large, say, over 600), so we report performance with this selection of dim_{max}. The other cases when dim_{max} is set to a smaller value are reported for references.

Method	dim_{max}	#mvp	mvp CPU	non-mvp CPU	total CPU
ChebyD95	424	107379	220.61	1958.78	2179.39
ARPACK	800	5561	12.14	4033.17	4045.31
ARPACK	500	5789	12.67	4383.40	4396.07
ARPACK	450	6321	13.89	6511.67	6525.56

TABLE 6.9
Laplacean 3D: dim = $64,000$, $k_{want} = 400$. For ChebD95, $act_{max} = 42$, block_size = 3, m= 15. The maximum difference between the 400 *smallest eigenvalues computed by ChebyD95 and ARPACK is 1.89e-11. The unit for CPU time is second.*

A more realistic problem comes from an application in materials-science. It consists of solving the following time independent *Schrödinger equation* using the form resulting from the well-known Kohn-Sham approximation [12],

$$H\Psi = \left[\frac{-\hbar^2}{2M}\nabla^2 + V(x,y,z)\right]\Psi = E\Psi. \tag{6.1}$$

Here H is the so called Hamiltonian operator, E is the energy eigenvalue for the sys-

Method	dim_{max}	#mvp	mvp CPU	non-mvp CPU	total CPU
ChebyD95	424	163779	1060.68	9492.55	10553.23
ARPACK	800	7360	50.01	14326.81	14376.82

TABLE 6.10

Laplacean 3D: dim $= 200,000$*,* $k_{want} = 400$*. For ChebD95,* $act_{max} = 42$*, block_size* $= 3$*,* $m = 15$*. The maximum difference between the* 400 *smallest eigenvalues computed by ChebyD95 and ARPACK is 5.546e-11. The unit for CPU time is second.*

tem, $\Psi(x, y, z)$ is the wave function, and $V(x, y, z)$ is the system *potential*. M denotes the characteristic mass of particles, and $\hbar$ is the Planck's constant. The problem is set up on a sphere and the eigenfunctions satisfy Dirichlet boundary conditions on the surface, i.e., $\Psi(x, y, z) = 0$, $(\sqrt{x^2 + y^2 + z^2} = \vartheta)$. A 12-th order centered finite difference scheme [6] is used for (6.1). The material we test is $Si_{34}H_{36}$. The matrix is obtained when the self-consistency is reached. Its structure may be seen in the left plot of Figure 6.1. It is saved to the Compressed Sparse Row (CSR) format and a SPARSKIT subroutine is called for the matrix vector products. In contrast with the 3D Laplaceans, where the matrix vector products can be coded directly without any manipulation of sparse matrices, this matrix does not exploit stencils and the situation is therefore much less favorable to the Chebyshev filtering method since matrix-vector products are more expensive. However, even in this unfavorable situation, the total CPU time of ChebyD95 is still comparable with the best performance cases of ARPACK. We note that for this model the eigenvalue multiplicity is not a problem, so the tolerance for both ARPACK and ChebyD95 is set to 10^{-10}.

Method	dim_{max}	#mvp	mvp CPU	non-mvp CPU	total CPU
ChebyD95	230	54447	1290.38	724.88	2015.26
ARPACK	400	3825	134.92	2103.06	2237.98
ARPACK	230	4672	164.39	3115.40	3279.80

TABLE 6.11

$Si_{34}H_{36}$*: dim* $= 97569$*,* $k_{want} = 200$*. For ChebyD95,* $act_{max} = 42$*, block_size = 3, m= 25. The maximum difference between the* 200 *smallest eigenvalues computed by ARPACK and ChebyD95 is 2.07e-14. All the CPU time unit is second.*

Method	dim_{max}	#mvp	mvp CPU	non-mvp CPU	total CPU
ChebyD95	530	182289	4190.86	3907.55	8098.41
ARPACK	1000	7841	272.70	9044.63	9317.33
ARPACK	600	8033	279.22	11478.50	11757.72

TABLE 6.12

$Si_{34}H_{36}$*: dim* $= 97569$*,* $k_{want} = 500$*. For ChebyD95,* $act_{max} = 42$*, block_size = 3, m= 25. The maximum difference between the* 500 *smallest eigenvalues computed by ARPACK and ChebyD95 is 2.84e-14. All the CPU time unit is second.*

As seen from Tables 6.9, 6.10, 6.11 and 6.12, ChebyD95 often requires many more matrix-vector products than ARPACK. This is because each call to the filter subroutine requires $m * block\_size$ matrix-vector products. However, this extra cost in matrix-vector products, generally leads to very fast convergence for the Chebyshev-Davidson method, in terms of (outer) iterations. Moreover, ChebyD95 saves CPU time on reorthogonalization because it requires a smaller dimension projection subspace. These are shown by the CPU time spent on non-matrix-vector products. (denoted as "non-mvp CPU" in the Tables 6.9 to 6.12) Hence, the proposed filtering method is advantageous when matrix-vector products are inexpensive. Moreover, with *Inner-Outer-Restart*, the filtering method may require only slightly more than half of the memory requirement of ARPACK to achieve similar best efficiency. But when the matrix-vector products are expensive, then the implicit filtering as done in ARPACK should be considerably more efficient than filtering via matrix-vector products.

We note that our FORTRAN code is still a working prototype (It has been developed in about 3 months) and that some parts of the code may not have been tuned to yield the best efficiency. As for the numerical tests, we again selected some reasonable values for the polynomial degree m, $block\_size$ and act_{max}, and then ran the tests. It is likely that other choices for these parameters may lead to better performance. This additional freedom in the selection of parameters is sometimes considered a disadvantage. However, our experience shows that it is quite easy to choose parameters to achieve performance comparable with existing best codes for eigenvalue computations.

7. Conclusion.

A Chebyshev filter Davidson-type algorithm has been presented for solving large symmetric eigenvalue problems. It consists of essentially filtering out unwanted parts of the spectrum by using an adaptive Chebyshev polynomial of the matrix. Comparisons with existing Davidson-type methods based on solving correction-equation, as well as with ARAPCK, show that the Chebyshev-Davidson method is efficient and robust. It gains over standard restarted versions of the Lanczos algorithm (e.g., ARPACK), by reducing the required number of outer iterations and the orthogonalization costs.

An advantage of the Chebyshev filtering approach is that it does not require an explicit preconditioning solve (e.g., ILU or multigrid). In this regard, it can be compared with a JD-approach in the situation where the solve is performed with a (non-preconditioned) Krylov method. However, the Chebyshev filter is more easily controlable and can be designed and tuned to better filter wanted and unwanted components.

A second advantage of the approach is that the filtering is remarkably simple to implement, both in a single vector version and in a block version. By integrating Davidson method with polynomial filtering, using a large number of available good initial vectors becomes not only possible but also practical. More details may be found in [35].

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. Meth. Engng, (to appear).

[2] Z. Bai, J. Demmel, J. Dongarra, A. Ruhe, and H. van der Vorst, eds., *Templates for the solution of algebraic eigenvalue problems*, Society for Industrial and Applied Mathematics, Philadelphia, PA, 2000.

[3] C. A. Beattie, M. P. Embree, and J. Rossi, *Convergence of restarted Krylov subspaces to invariant subspaces*, SIAM J. Matrix Anal. Appl., 25 (2004), pp. 1074–1109.

[4] J. H. Brandts, *Solving eigenproblems: from Arnoldi via Jacobi-Davidson to the Riccati method*, Springer Lecture Notes in Computer Science, 2542 (2003), pp. 167–173.

[5] D. Calvetti, L. Reichel, and D. C. Sorensen, *An implicit restarted Lanczos method for large symmetric eigenvalue problem*, Elec. Trans. Numer. Anal., 2 (1994), pp. 1–21.

[6] 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.

[7] 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.

[8] 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.

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

[10] 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.

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

[12] W. Kohn and L. J. Sham, *Self-consistent equations including exchange and correlation effects*, Phys. Rev., 140 (1965), pp. A1133–A1138.

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

[14] R. B. Lehoucq, D. C. Sorensen, and C. Yang, *ARPACK USERS GUIDE: Solution of Large Scale Eigenvalue Problems by Implicitly Restarted Arnoldi Methods*, SIAM, Philadelphia, 1998. Available at `http//www.caam.rice.edu/software/ARPACK/`.

[15] R. B. Morgan and D. S. Scott, *Generalization of Davidson's method for computing eigenvalues of sparse symmetric matrices*, SIAM J. Statistical and Scientific Computing, 7 (1986), pp. 817–825.

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

[17] C. C. Paige and M. A. Saunders, *Solution of sparse indefinite systems of linear equations*, SIAM Journal on Numerical Analysis, 12 (1975), pp. 617–629.

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

[19] ———, *The Symmetric Eigenvalue Problem*, no. 20 in Classics in Applied Mathematics, SIAM, Philadelphia, PA, 1998.

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

[21] ———, *Simultaneous iteration method for symmetric matrices*, in Handbook for Automatic Computation (Linear Algebra), J. H. Wilkinson and C. Reinsh, eds., vol. II, Springer-Verlag, 1971, pp. 284–302.

[22] Y. Saad, *Chebyshev acceleration techniques for solving nonsymmetric eigenvalue problems*, Math. Comp., 42 (1984), pp. 567–588.

[23] ———, *Numerical Methods for Large Eigenvalue Problems*, John Wiley, New York, 1992. Available at `http://www.cs.umn.edu/~saad/books.html`.

[24] Y. Saad and M. Schultz, *GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems*, SIAM J. Sci. Stat. Comput., 7 (1986), pp. 856–869.

[25] 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.

[26] 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.

[27] D. C. Sorensen, *Implicit application of polynomial filters in a k-step Arnoldi method*, SIAM J. Matrix Anal. Appl., 13 (1992), pp. 357–385.

[28] D. C. Sorensen and C. Yang, *Accelerating the lanczos algorithm via polynomial spectral transformations*, Tech. Rep. TR97-29, Dept. of Comp. and Appl. Math., Rice University,, 1997.

[29] G. W. Stewart, *A Krylov–Schur algorithm for large eigenproblems*, SIAM J. Matrix Anal. Appl., 23 (2001), pp. 601–614.

[30] D. S. Watkins and L. Elsner, *Convergence of algorithms of decomposition type for the eigenvalue problem*, Linear Algebra Appl., 41 (1991), pp. 19–47.

[31] 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.

[32] K. Wu and H. Simon, *Thick-restart Lanczos method for large symmetric eigenvalue problems*, SIAM J. Matrix Anal. Appl., 22 (2000), pp. 602–616.

[33] Y. Zhou, *Studies on Jacobi-Davidson, Rayleigh Quotient iteration, inverse iteration generalized Davidson and Newton updates*, tech. rep., University of Minnesota, 2004.

[34] Y. Zhou and Y. Saad, *Block Krylov-Schur method for large symmetric eigenvalue problems*, tech. rep., Minnesota Supercomputing Institute, University of Minnesota, 2004.

[35] ———, *Block-wise polynomial filtered Davidson-type subspace iteration*, tech. rep., Minnesota Supercomputing Institute, University of Minnesota, 2005.