A FILTERED LANCZOS PROCEDURE FOR EXTREME AND INTERIOR EIGENVALUE PROBLEMS *

HAW-REN FANG† AND YOUSEF SAAD†

Abstract.
When combined with Krylov projection methods, polynomial filtering can provide a powerful method for extracting extreme or interior eigenvalues of large sparse matrices. This general approach can be quite efficient in the situation when a large number of eigenvalues is sought. However, its competitiveness depends critically on a good implementation. This paper presents a technique based on such a combination to compute a group of extreme or interior eigenvalues of a real symmetric (or complex Hermitian) matrix. The technique harnesses the effectiveness of the Lanczos algorithm with partial reorthogonalization and the power of polynomial filtering. Numerical experiments indicate that the method can be far superior to competing algorithms when a large number of eigenvalues and eigenvectors is to be computed.

Key words. Lanczos algorithm; polynomial filtering; partial reorthogonalization; interior eigenvalue problems.

1. Introduction. The problem addressed in this paper is to compute eigenvalues located in a specified interval of a large real symmetric or complex Hermitian matrix, along with their associated eigenvectors. The interval, which we will also refer to as a 'window', can consist of a subset of the largest or smallest eigenvalue, in which case the eigenvalues requested are in one of the two ends of the spectrum. When the window is well inside the interval containing the spectrum, this is often referred to as an 'interior eigenvalue problem'. Eigenvalues in the inner portion of the spectrum are called 'interior eigenvalues', though this is clearly a loose definition.

Computing a large number of interior eigenvalues of a large symmetric matrix remains one of the most difficult problems in computational linear algebra today. The classical approach to the problem is to use a form of shift-and-invert technique. If we are interested in eigenvalues around a certain shift σ, shift-and-invert consists of using a projection-type method (subspace iteration, Lanczos) to compute the eigenvalues of the matrix $(A-\sigma I)^{-1}$. The eigenvalues $(\lambda_i-\sigma)^{-1}$ of this matrix become the dominant eigenvalues for those λ_i's close to σ and as a result they are easy to compute with the projection method. This approach has been the most common in structural analysis codes [1]. Computational codes based on this approach select a shift dynamically and perform a factorization of the matrix $A-\sigma I$ (or $A-\sigma B$ for the generalized case).

There are a number of situations when shift-and-invert will be either inapplicable, or too slow to be of practical interest. For example, it is known that problems based on a 3-D physical mesh will tend to give matrices that are very expensive to factor due to both computational and memory requirements. There are also situations when the matrix A is not available explicitly but only through subroutines to perform matrix-vector products. Finally, in the situation, common in electronic structure calculations, when a very large number of eigenvalues is to be computed, the number of factorizations to be performed, i.e., the number of shifts necessary to obtain all wanted eigenvalues, can be quite high. Since the cost of each factorization is expensive, the approach will loose its appeal. In fact for the example of electronic structure calculations, one is struck by the double whammy of the 3-D nature of the problem and the large number of eigenvalues.

In this paper we address this problem by combining two major ingredients: the Lanczos algorithm on the one hand and polynomial filtering on the other. A key feature of the Lanczos algorithm is to exploit partial reorthogonalization [27]. This is favored over the alternatives of 'selective reorthogonalization' [22] and 'full reorthogonalization' [25], due to its compromise between accuracy, cost, and ease of implementation.

Our focus is on the eigen-space rather than individual eigenvectors. Standard diagonalization software places too big an emphasis on obtaining accurate eigenvectors, at a cost that is often quite high. Focusing on invariant subspaces is much more natural and can be more cost effective. All that is needed is that a good basis of the subspace be computed, but this basis does not need to be a basis of accurate eigenvectors. This principle was exploited with good success by Zhou *et al.* [30, 31], where a Chebyshev-type approach was used in the nonlinear context of self-consistent field (SCF) iterations in electronic structure calculations.

*Work supported by DOE under grant DE-SC0001878 and by the Minnesota Supercompter Institute.
†Computer Science & Engineering, University of Minnesota; Minneapolis, MN 55455.

The specific problem considered in this paper can be stated as follows. Given a real symmetric (or complex Hermitian matrix), compute *all* the eigenvalues in a given interval $[\xi, \eta]$, which is a given sub-interval of the interval $[\lambda_{min}, \lambda_{max}]$ containing all eigenvalues. It is assumed that an interval $[a, b]$ which (tightly) contains $[\lambda_{min}, \lambda_{max}]$ is available a priori. This means that two scalars a, b with $a \lessapprox \lambda_{min}$ and $\lambda_{max} \lessapprox b$ are already available, typically computed in a pre-processing stage. For example, a and b can be estimated by the Gershgorin circle theorem or via a simple modification of the approximate smallest and largest eigenvalues obtained from the standard Lanczos procedure.

There are three type of problems. If $a = \xi < \eta < b$, then the requested eigenvalues are in the 'lower end' of the spectrum. If $a < \xi < \eta < b$, then the requested eigenvalues are 'interior'. If $a < \xi < \eta = b$, then the requested eigenvalues are in the 'upper end' of the spectrum. Matrix polynomials and filtering matrices have been used in linear scaling and related methods, see for example [9, 10, 15, 16, 17]. In some cases, these methods will compute the entire density matrix [16], or just a small part of it as an approximation [10]. Along these lines, a polynomial filtered Lanczos procedure with partial reorthogonalization has been utilized for electronic structure calculations [2]. The main difference between the present work and that in [2] is that we consider the computation of interior eigenvalues computations as well extreme eigenvalues, whereas [2] addresses only extreme eigenvalue problems.

Section 2 reviews the Lanczos method and sketches the basic idea of polynomial filtering. Section 3 gives an overview of the reorthogonalization schemes and provides some details of partial reorthogonalization. Section 4 provides some details on the incorporation of polynomial filtering to extract eigenvalues in a given interval $[\xi, \eta]$ and their associated eigenvectors. Experimental results are reported in Section 5 and a conclusion is given in Section 6.

2. The Lanczos method with polynomial filtering. Given a Hermitian matrix $A \in \mathbb{C}^{n\times n}$ with a unit (typically random) column vector $q_1 \in \mathbb{C}^n$, the Lanczos algorithm [12] (see also [4, 5, 21, 25]) builds a sequence of vectors $q_1, q_2, \ldots, q_m \in \mathbb{C}^n$ which form an orthonormal basis of the Krylov subspace[1]

$$\mathcal{K}_m(A, q_1) = \mathtt{Span}\{q_1, Aq_1, A^2q_1, \ldots, A^{m-1}q_1\}. \tag{2.1}$$

2.1. Background: The basic Lanczos algorithm. A sketch of the Lanczos procedure is given in Algorithm 1. Essentially, at each step we compute Aq_j which is orthogonalized against q_j and (when $j > 1$) against q_{j-1}.

```
1: {Given a Hermitian matrix A ∈ C^{n×n}, with an initial unit vector q_1 ∈ C^n.}
2: q_0 := 0, β_1 := 0
3: for all i = 1, 2, ... do
4:     w := Aq_i − β_i q_{i−1}
5:     α_i := q_i^H w
6:     {Check convergence.}
7:     w := w − α_i q_i
8:     {Apply reorthogonalization.}
9:     β_{i+1} := ‖w‖
10:    if β_{i+1} == 0 then
11:        Pick q_{i+1} := a unit vector orthogonal to q_1, ..., q_i.
12:    else
13:        q_{i+1} := w/β_{i+1}
14:    end if
15: end for
```

Algorithm 1: The Lanczos algorithm.

Note that Algorithm 1 implements a modified Gram-Schmidt process. Paige [19] and Parlett [21] suggest this as the preferred implementation among several other options. The Lanczos algorithm requires the matrix A only in the form of matrix-vector products which can be quite appealing in some situations, such as when A is available in stencil form. The sequence of vectors computed in the course of the Lanczos

[1] Here we assume that q_1 is in the span of at least m eigenvectors of A.

algorithm satisfies the 3-term recurrence:

$$\beta_{i+1}q_{i+1} = Aq_i - \alpha_i q_i - \beta_i q_{i-1}. \tag{2.2}$$

Therefore, in principle only three Lanczos vectors need to be stored in main memory. As is well-known, *in exact arithmetic*, this 3-term recurrence, as implemented by Algorithm 1, would deliver an orthonormal basis $q_1, \ldots, q_m$, of the Krylov subspace K_m. In the presence of rounding, orthogonality between the q_i's is quickly lost, and a form of reorthogonalization is needed in practice. Alternative schemes obviate the need for reorthogonalization at the expense of a large increase the number of steps, see [4].

Let $Q_m = [q_1, \ldots, q_m]$ and T_m denote the symmetric tridiagonal matrix

$$T_m = \text{Tridiag}[\beta_i, \alpha_i, \beta_{i+1}], \tag{2.3}$$

the tridiagonal matrix with entries $\beta_i, \alpha_i, \beta_{i+1}$ in the ith row, where the scalars α_i, β_i are those produced by the Lanczos algorithm. Aggregating (2.2) into matrix form, we obtain

$$AQ_m = Q_m T_m + \beta_{m+1} q_{m+1} e_m^H, \tag{2.4}$$

where e_m is the mth column of the canonical basis and q_{m+1} is the last vector computed by the Lanczos algorithm. In exact arithmetic $\alpha_1, \ldots, \alpha_m$ are real, since $T_m = Q_m^H A Q_m$ is Hermitian. In addition we choose β_i's as real numbers. Therefore T_m is a *real* symmetric matrix. However, due to rounding, α_i's may have small imaginary values, which can be dropped in practice.

Let θ_i, y_i be an eigen-pair of T_m. In case of ambiguity, $\theta_i^{(m)}, y_i^{(m)}$ will denote the same eigen-pair at the mth step of the process. Then the eigenvalues $\theta_i^{(m)}$, known as Ritz values, will approximate some of the eigenvalues of A as m increases. Specifically, the extreme eigenvalues are often approximated first. The vectors $Q_m y_i^{(m)}$, referred to as Ritz vectors, will approximate the related eigenvectors of A. The Lanczos algorithm quickly yields good approximations to extreme eigenvalues of A while convergence is often much slower for the interior of the spectrum.

A scheme based on the Lanczos procedure, relies on the orthogonality of the Lanczos vectors $q_1, \ldots, q_m$. As was mentioned above, the q_i's, which in theory form an orthonormal basis, loose their orthogonality in practice. Orthogonality is lost very rapidly after one eigenvector starts converging, leading to an unstable underlying computation. This was studied in detail by Paige in the 1970s [18, 19, 20]. A remedy to this problem is to reorthogonalize the vectors when needed. A further discussion will be given in Section 3. For now we only need to know that a form of reorthogonalization is applied.

2.2. Filtered Lanczos algorithms. When the number of requested eigenvalues becomes large, the number of needed Lanczos vectors may increase so much that the orthogonalization process becomes very expensive. One way to circumvent this is to apply a spectral transformation [21], i.e., to compute the eigenvalues of either $(A - \sigma)^{-1}$ (shift-and-invert) or $\rho(A)$, where ρ is a certain polynomial.

The method we use is based on spectral transformation using polynomials, i.e., replacing A by $\rho(A)$ in order to compute the requested eigenvalues in far fewer steps than would be required with A directly. This can be illustrated with a very simple example. Suppose we need to compute all the eigenvalues in the interval $[\xi, \eta] = [3.9, 4.1]$ of a matrix whose spectrum is known to be included in $[a, b] = [0, 8]$. Our first option is to simply use the Lanczos algorithm to compute as many eigenvalues as needed until those in the desired interval are captured, i.e., until they all converge. This is likely to require a great many steps when n is large.

Our second option is to factor the matrix $A - \sigma I$, with $\sigma = (\xi + \eta)/2 = 4$, and apply Lanczos to the inverse of $A - \sigma I$. This is likely to require far fewer steps to yield convergence to the desired eigenvalues – but a factorization is now needed, possibly at the expense of enormous fill-in.

A third option is to use spectral transformation with a polynomial. For example, if a quadratic polynomial is used, we would compute the eigenvalues of the matrix $B = [\alpha I - (A - \sigma I)^2]/d$. We would like to find the parameters α, d in such way that all unwanted eigenvalues of A are transformed to eigenvalues of B that are < 1, and those that are wanted should be transformed into eigenvalues of B that are ≥ 1. This is achieved by taking $\alpha = (\sigma^2 + \delta^2)/2$, and $d = (\sigma^2 - \delta^2)/2$, with $\sigma = 4, \delta = 0.1$ in the above example. Note that the scaling by d has no effect, so we can define $B = \rho(A) \equiv \alpha I - (A - \sigma I)^2$. We can then apply the Lanczos algorithm to B to compute those wanted eigenvalues and corresponding eigenvectors. The

eigenvectors of the original matrix A are the same as those of B. The eigenvalues can be obtained from the eigenvectors, for example from their Rayleigh quotients.

Using a degree 2 polynomial is usually not optimal. In particular, it may not be a good choice when the interval of desired eigenvalues is not exactly in the middle of $[a, b]$ as the case in the above example. In addition, the separation of the eigenvalues $\alpha - (\lambda_i - \sigma)^2$ can be very poor for those eigenvalues near σ, and this can cause slow convergence. For these reasons it is important to allow the polynomial to be of higher degree and to select this polynomial very carefully. However, whether a low or high degree polynomial is used, the principle is the same. The desired polynomial will have a property illustrated in Figure 2.1. *We need to obtain a scalar γ with the property that $\rho(\lambda) \geq \gamma$ for $\lambda \in [\xi, \eta]$, and $\rho(\lambda) < \gamma$ for $\lambda \notin [\xi, \eta]$ but $\lambda \in [a, b]$.* This is desired for all three types of problems, for smallest eigenvalues, for interior eigenvalues, and for largest eigenvalues. The corresponding filters are called low-pass filters, mid-pass filters, and high-pass filters, respectively.

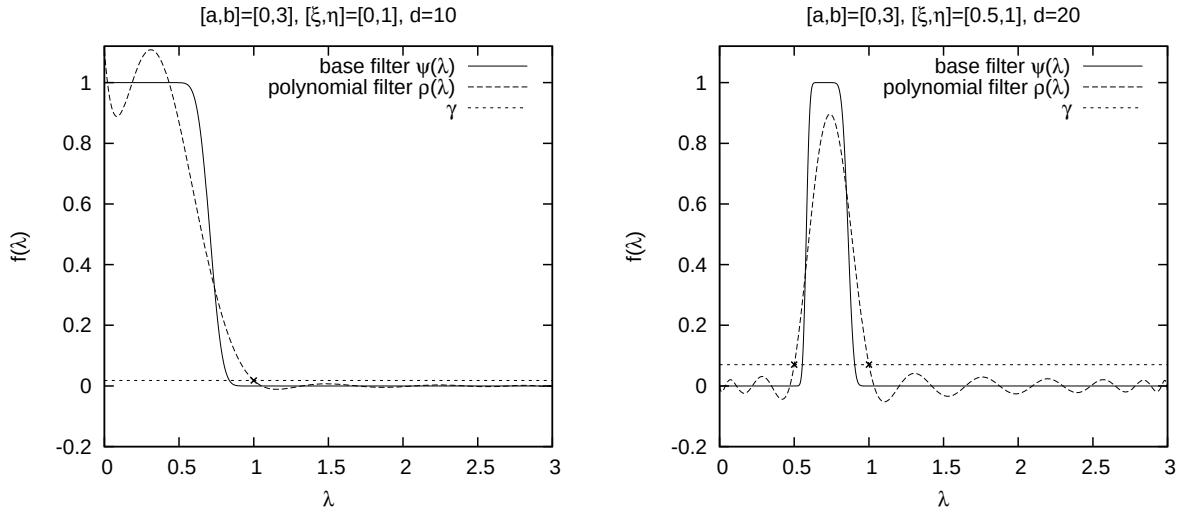

FIG. 2.1. *A basic high-pass polynomial filter (left) and a basic mid-pass polynomial filter (right).*

A polynomial filtered Lanczos algorithm for computing interior eigenvalues is sketched in Algorithm 2. A few explanations and comments follow. The polynomial filter $\rho(\lambda)$ is determined by an algorithm to be seen in Section 4 (Algorithm 4) which also provides the scalar γ that separates the $\rho(\lambda_j)$'s according to whether $\lambda_j \in [\xi, \eta]$ or not. The convergence test in line 9 is performed every 'stride' iterations. The eigenvalues θ_j's of T_i in line 22, called the Ritz values, approximate eigenvalues of $\rho(A)$. Since $\rho(A)$ and A share the same eigenvectors, in line 23 we can obtain approximate eigenvalues of A by $\langle y_j, Ay_j \rangle$ with the approximate eigenvectors y_j's of $\rho(A)$. The test in the last line rejects the eigenvalues not in the requested interval $[\xi, \eta]$. This is required if the polynomial filter does not fully satisfy the separation property illustrated in Figure 2.1.

Convergence of the algorithm is checked in line 9 of Algorithm 2. This is performed as follows. Let $\theta_j^{(i)}$'s be the eigenvalues of T_i and $\theta_j^{(i-1)}$'s be the eigenvalues of T_{i-1}, where the symmetric tridiagonal matrices T_i and T_{i-1} are defined in (2.3). With a preset tolerance ϵ, the desired eigenvalues are deemed to have converged if the number of eigenvalues of T_i satisfying $\theta_j^{(i)} \geq \gamma$ is the same as the number of eigenvalues of T_{i-1} satisfying $\theta_j^{(i-1)} \geq \gamma$, and

$$\left| \frac{\sigma_i - \sigma_{i-1}}{\sigma_{i-1}} \right| < \epsilon, \text{ where } \sigma_i = \sum_{\theta_j^{(i)} \geq \gamma} \theta_j^{(i)} \text{ and } \sigma_{i-1} = \sum_{\theta_j^{(i-1)} \geq \gamma} \theta_j^{(i-1)}. \tag{2.5}$$

By the interlacing property of eigenvalues of a reducible symmetric tridiagonal matrix, e.g., [6, Theorem 8.5.1], $|\sigma_i - \sigma_{i-1}|$ is the sum of Ritz value differences $|\theta_j^{(i)} - \theta_j^{(i-1)}|$ for $\theta_j^{(i)}, \theta_j^{(i-1)} \geq \gamma$. Therefore, the error is measured in the relative and average sense.

Input: matrix $A \in \mathbb{C}^{n\times n}$; initial unit vector $q_1 \in \mathbb{C}^n$; interval of desired eigenvalues $[\xi, \eta]$, and degrees of base filter and polynomial filter $\{d_i\}_{i=1}^4$ and d.
Output: eigenvalues λ_j's in the interval $[\xi, \eta]$ and the corresponding eigenvectors y_j's.
Determine a polynomial filter $\rho(\lambda)$ with γ such that $\rho(\lambda) \geq \gamma$ for $\lambda \in [\xi, \eta]$. ▷ Algorithm 4
$q_0 := 0$, $\beta_1 = 0$
for all $i = 1, 2, \ldots$ **do**
    $w := \rho(A)q_i - \beta_i q_{i-1}$ ▷ $\rho(A)q_i$ is computed via Algorithm 3.
    $\alpha_i := q_i^H w$
    if rem$(i, \text{stride}) == 0$ **then** ▷ Check convergence.
        If it converges, break the loop.
    end if
    $w := w - \alpha_i q_i$
    if w fails to pass the semi-orthogonality test **then**
        Apply a reorthogonalization scheme here. ▷ We use partial reorthogonalization.
    end if
    $\beta_{i+1} := \|w\|$
    if $\beta_{i+1} == 0$ **then**
        Pick q_{i+1} as a unit vector orthogonal to $q_1, \ldots, q_i$.
    else
        $q_{i+1} := w/\beta_{i+1}$
    end if
end for
Compute the Ritz values θ_j's of T_i for $\theta_j \geq \gamma$ and the corresponding Ritz vectors y_j's.
Compute the approximate eigenvalues $\lambda_j := \langle y_j, Ay_j \rangle$ for all y_j's from above.
Reject all λ_j, y_j pairs such that $\lambda_j \notin [\xi, \eta]$.

Algorithm 2: The polynomial filtered Lanczos algorithm.

To complete the description of our algorithm, two ingredients are now needed: 1) How to reorthogonalize the Lanczos vectors; 1) How to compute the desired polynomial filters. These two issues are discussed in turn in Sections 3 and 4, respectively.

3. Reorthogonalization schemes. Section 3.1 reviews the orthogonality issue in the Lanczos process. Section 3.2 gives some details of the partial reorthogonalization scheme.

3.1. Orthogonality in the Lanczos process. Various reorthogonalization schemes have been proposed to correct the loss of orthogonality of the Lanczos vectors. The simplest but most expensive one is that of full reorthogonalization, whereby the orthogonality of the current Lanczos vector q_i against all previous vectors $q_1, \ldots, q_{i-1} \in \mathbb{C}^n$ is reinstated at each step i. When summed over m steps, the overall cost of orthogonalization, by the Gram-Schmidt process for instance, is $O(m^2 n)$. This is not an issue if m is small but when m is large, the orthogonalization procedure eventually dominates the cost of the Lanczos algorithm. When computing many eigenvalues, the number of steps m will usually be quite large and this can result in a very expensive scheme.

In the literature there are actually three different approaches to reduce the cost of reorthogonalization, namely *periodic reorthogonalization* [7], *selective reorthogonalization* [22], and *partial reorthogonalization* [27]. See also [28, chapter 5.3] for an introduction. Methods based on partial reorthogonalization, a variant of which is used in this paper, attempt to reorthogonalize only when it is deemed necessary. The goal is not to guarantee that the vectors are orthogonal to the machine precision level, but to ensure that they are at least nearly orthogonal. Typically, the loss of orthogonality is allowed to grow to roughly $\sqrt{\epsilon_M}$, the square root of the machine precision, before a reorthogonalization is performed. A result by Simon [26] ensures that we can still get accurate eigenvalue approximations by the Ritz values, i.e., eigenvalues of the tridiagonal matrix T_m, despite the reduced level of orthogonality of the Lanczos vectors.

For all reorthogonalization schemes (full, periodic, selective, and partial), we no longer have a 3-term recurrence. This results in an increased cost related to increased computations as well as memory traffic. As will be discussed in Section 4, the use of polynomial filtering significantly reduces the number of required

Lanczos vectors needed to extract the desired interior eigenvalues, thereby reducing drastically the cost of reorthogonalization at the expense of a higher cost related to matrix-vector products. See also [2, 3].

3.2. Semi-orthogonality and partial reorthogonalization. In what follows, we use a hat for each computed quantity (i.e., with rounding errors), and rewrite (2.2) as

$$\hat{\beta}_{i+1}\hat{q}_{i+1} = A\hat{q}_i - \hat{\alpha}_i\hat{q}_i - \hat{\beta}_i\hat{q}_{i-1} - \Delta f_i, \tag{3.1}$$

where Δf_i accounts for rounding errors. If a polynomial filter $\rho(\lambda)$ is utilized, $A\hat{q}_i$ is replaced by $\rho(A)\hat{q}_i$ and the discussion remains the same. To measure the loss of orthogonality, we define the level of orthogonality

$$\kappa_i = \max_{1\le j\le i-1} |\hat{q}_i^H \hat{q}_j|.$$

Full reorthogonalization aims at keeping κ_i at the roundoff level. However, in practice semi-orthogonality, i.e., requiring only that $\kappa_i \le \sqrt{\epsilon_M}$ with ϵ_M the machine epsilon, is sufficient to prevent spurious duplicate copies of eigenvalues in the computation. Therefore, the objective is to find an upper bound on κ_i which can be computed inexpensively. Reorthogonalization is applied when this bound is greater than $\sqrt{\epsilon_M}$.

Let $\omega_{ij} = \hat{q}_i^H \hat{q}_j$ for $i \ne j$ and $\omega_{ii} = 1$. Replacing i by j in (3.1), we obtain

$$\hat{\beta}_{j+1}\hat{q}_{j+1} = A\hat{q}_j - \hat{\alpha}_j\hat{q}_j - \hat{\beta}_j\hat{q}_{j-1} - \Delta f_j. \tag{3.2}$$

After simplification in the expression $\hat{q}_j^H \times$(3.1) - $\hat{q}_i^H \times$(3.2), we obtain

$$\hat{\beta}_{i+1}\omega_{i+1,j} = \hat{\beta}_{j+1}\omega_{i,j+1} + (\hat{\alpha}_j - \hat{\alpha}_i)\omega_{ij} + \hat{\beta}_j\omega_{i,j-1} - \hat{\beta}_i\omega_{j,i-1} + \hat{q}_i^H\Delta f_j - \hat{q}_j^H\Delta f_i. \tag{3.3}$$

The equation (3.3) was first shown by Paige [18] and by Takahasi and Natori [29]. Note that here we assumed A is real symmetric to simplify the discussion. If A is complex Hermitian, the discussion will be similar.

Writing (3.3) as a recurrence, we have for $i = 1, 2, \ldots$,

$$\begin{aligned} \omega_{i+1,j} &= \frac{1}{\hat{\beta}_{i+1}}\left[\hat{\beta}_{j+1}\omega_{i,j+1} + (\hat{\alpha}_j - \hat{\alpha}_i)\omega_{ij} + \hat{\beta}_j\omega_{i,j-1} - \hat{\beta}_i\omega_{j,i-1} + \hat{q}_i^H\Delta f_j - \hat{q}_j^H\Delta f_i\right], \quad j = 1, \ldots, i-1 \\ \omega_{i+1,i} &= \hat{q}_{i+1}^H\hat{q}_i, \qquad \omega_{i+1,i+1} = 1, \end{aligned} \tag{3.4}$$

where the base case is $\omega_{1,1} = 1$ and $\omega_{i,0} = 1$ for all i. Note that the recursion involves $\hat{q}_i^H\Delta f_j$ and $\hat{q}_j^H\Delta f_i$ for $j = 1, \ldots, i-1$. These rounding error vectors $\Delta f_i, \Delta f_j$ are not computable without higher precision arithmetic. In addition, the vector inner products should be avoided, in an effort to keep the estimate of the level of the orthogonality inexpensive to evaluate. To this end we consider the following recurrence. For $i = 1, 2, \ldots$,

$$\begin{aligned} \bar{\omega}_{i+1,j} &:= \frac{1}{\hat{\beta}_{i+1}}\left[\hat{\beta}_{j+1}\bar{\omega}_{i,j+1} + (\hat{\alpha}_j - \hat{\alpha}_i)\bar{\omega}_{ij} + \hat{\beta}_j\bar{\omega}_{i,j-1} - \hat{\beta}_i\bar{\omega}_{j,i-1} + \vartheta_{ij}\right], \quad j = 1, \ldots, i-1 \\ \bar{\omega}_{i+1,i} &:= \varphi_i, \qquad \bar{\omega}_{i+1,i+1} := 1, \end{aligned} \tag{3.5}$$

with the base case $\bar{\omega}_{1,1} = 1$ and $\bar{\omega}_{i,0} = 1$ for all i. Note that ϑ_{ij} and φ_i in (3.5) correspond to $\hat{q}_i^H\Delta f_j - \hat{q}_j^H\Delta f_i$ and $\hat{q}_{i+1}^H\hat{q}_i$ in (3.4), respectively. The goal is to choose ϑ_{ij}'s and φ_i's properly, such that in practice the computed $\bar{\omega}_{ij}$'s satisfy the property,

$$\bar{\kappa}_{i+1} \ge \kappa_{i+1}, \text{ where } \bar{\kappa}_{i+1} = \max_{1\le j\le i} |\bar{\omega}_{i+1,j}| \text{ and } \kappa_{i+1} = \max_{1\le j\le i} |\omega_{i+1,j}|.$$

Reorthogonalization is performed whenever $\bar{\kappa}_{i+1}$ is greater than $\sqrt{\epsilon_M}$. Simon [27] used normally distributed random numbers for ϑ_{ij} and φ_i. We adopt the approach used in `PROPACK` [13], which potentially gives a tighter upper bound $\bar{\kappa}_{i+1}$ on κ_{i+1}.

Recall that φ_i in (3.5) plays the role of $\hat{q}_{i+1}^H\hat{q}_i$ in (3.4). Since $\hat{q}_{i+1}$ and $\hat{q}_i$ are unit vectors at roundoff level, a standard rounding error analysis, e.g., [8], yields a bound on $|\hat{q}_{i+1}^H\hat{q}_i|$ proportional to $n\epsilon_M$, where ϵ_M is the machine epsilon. However this bound is too pessimistic. In practice we use the rule of thumb

that the mean rounding error is proportional to the square root of number of operations. Hence, we set $\varphi_i = \sqrt{n}\epsilon_M/2$ in the recursion (3.5). Note that $\epsilon_M/2$ is the unit roundoff.

For an appropriate value of ϑ_{ij} in (3.5), we consider $\hat{q}_i^H \Delta f_j - \hat{q}_j^H \Delta f_i$ or simply $\hat{q}_j^H \Delta f_i$ in (3.4). The rationale for ignoring $\hat{q}_i^H \Delta f_j$ and considering only $\hat{q}_j^H \Delta f_i$ is that $\|\Delta f_i\|$ may be larger than in $\|\Delta f_j\|$, since Δf_i is from an later iteration, i.e., $i > j$, and the accumulated rounding errors may be larger. The product $\hat{q}_j^H \times$ (3.1) yields

$$\omega_{j,i+1} = \hat{q}_j^H A \hat{q}_i - \hat{\alpha}_i \omega_{ij} - \hat{\beta}_i \omega_{j,i-1} - \hat{q}_j^H \Delta f_i.$$

We should focus on $\hat{q}_j^H A \hat{q}_i$, since the factors $\omega_{j,i+1}$, $\hat{\alpha}_i \omega_{ij}$, and $\hat{\beta}_i \omega_{j,i-1}$ are modest under the assumption of semi-orthogonality.

A standard rounding error analysis gives a bound on $|\hat{q}_j^H A \hat{q}_i|$ proportional to $n\epsilon_M \|A\|_2$ for $j \le i-2$. Again, this bound is too pessimistic, and therefore we use the rule of thumb to obtain the estimate $(\sqrt{n}\epsilon_M/2)\|A\|_2$. When the polynomial filtering technique is used, an interval $[a, b]$ which tightly contains the spectrum is sought first. Since $\|A\|_2$ is the largest singular value of A, we can use $\max\{|a|, |b|\}$ as a tight bound on $\|A\|_2$.

Alternatively, since $\|A\|_2$ can be approximated quickly by the largest singular value of the symmetric tridiagonal matrix T_i from the Lanczos algorithm, we consider $(\sqrt{n}\epsilon_M/2)\|T_i\|_2$ instead. We can apply the Gershgorin circle theorem to T_i for a bound on $\|T_i\|_2$. Furthermore, a sophisticated strategy is to apply the Gershgorin circle theorem to obtain a bound on $\|T_i^2\|_2$, whose square root usually provides a tighter bound on $\|T_i\|_2$. Note that this bound can be updated recursively when i increases. Let $g_i^{(j)}$ be the upper bound of the jth Gershgorin circle of T_i^2, and $g_i = \max\{g_i^{(j)} | j = 1, \ldots, i\}$ be an upper bound of $\|T_i^2\|_2$. Recall that T_i is symmetric tridiagonal with diagonal elements $\alpha_1, \ldots, \alpha_i$ and sub-diagonal elements $\beta_2, \ldots, \beta_i$. We expand T_i^2 as

$$\begin{bmatrix}
\alpha_1^2+\beta_2^2 & (\alpha_1+\alpha_2)\beta_2 & \beta_2\beta_3 & & & & \\
(\alpha_1+\alpha_2)\beta_2 & \beta_2^2+\alpha_2^2+\beta_3^2 & (\alpha_2+\alpha_3)\beta_3 & \beta_3\beta_4 & & & \\
\beta_2\beta_3 & (\alpha_2+\alpha_3)\beta_3 & \beta_3^2+\alpha_3^2+\beta_4^2 & (\alpha_3+\alpha_4)\beta_4 & \beta_4\beta_5 & & \\
 & \ddots & \ddots & \ddots & \ddots & \ddots & \\
 & & \beta_{i-2}\beta_{i-1} & (\alpha_{i-2}+\alpha_{i-1})\beta_{i-1} & \beta_{i-1}^2+\alpha_{i-1}^2+\beta_i^2 & (\alpha_{i-1}+\alpha_i)\beta_i & \\
 & & & \underline{\beta_{i-1}\beta_i} & \underline{(\alpha_{i-1}+\alpha_i)\beta_i} & \underline{\beta_i^2+\alpha_i^2} &
\end{bmatrix},$$

where the underlined terms are those *not* in T_{i-1}^2. Therefore, the following recurrence is clear.

$$\begin{aligned}
g_i^{(i-2)} &= g_{i-1}^{(i-2)} + |\beta_{i-1}\beta_i|, \\
g_i^{(i-1)} &= g_{i-1}^{(i-1)} + \beta_i^2 + |(\alpha_{i-1}+\alpha_i)\beta_i|, \\
g_i^{(i)} &= \beta_i^2 + \alpha_i^2 + |(\alpha_{i-1}+\alpha_i)\beta_i| + |\beta_{i-1}\beta_i|, \\
g_i &= \max\{g_{i-1}, g_i^{(i-2)}, g_i^{(i-1)}, g_i^{(i)}\},
\end{aligned}$$

for $i \ge 3$. The cases $i = 1$ and $i = 2$ are trivial. Since we just need to update the Gershgorin circle bounds of T_i^2 from those of T_{i-1}^2 with the underlined terms, the cost is negligible. The computed $\sqrt{g_i}$ is an upper bound of $\|T_i\|$. To conclude, we use $\vartheta_{ij} = (\sqrt{n}\epsilon_M/2)\sqrt{g_i}$ in the recursion (3.5). Note that this factor can be adjusted according to the degree of sparsity of matrix A, as well as whether a polynomial filter is used and the degree of the polynomial. In practice we use $\vartheta_{ij} = (\sqrt{n}\epsilon_M/2)\sqrt{g_i}$ and this provides a good safeguard for enforcing semi-orthogonality.

Once $\bar{\kappa}_{i+1}$ comes above $\sqrt{\epsilon_M}$, semi-orthogonality is no longer guaranteed, and reorthogonalization must be performed. A natural question is against which Lanczos vectors should the current vector be reorthgonalized. A simple but effective strategy is to reorthogonalize the current Lanczos vector against all previous Lanczos vectors. This is more than what is needed since the goal is to maintain only semi-orthogonality among the Lanczos vectors. More economic strategies are proposed by Simon [27]. An important idea in partial reorthogonalization is that, once the violation of a semi-orthogonality criterion forces us to reorthogonalize q_{i+1} at step i, we must also reorthogonalize q_{i+2} at the $(i+1)$st step as well [22, 27]. A justification can be found in [27].

Finally, we have also incorporated the extended local reorthogonalization and double reorthogonalization schemes. In short, the local reorthogonalization (against q_i and q_{i-1} twice at iteration i) is performed or the global reorthogonalization (against all selected previous vectors) is doubled, whenever some condition is satisfied. Details are omitted.

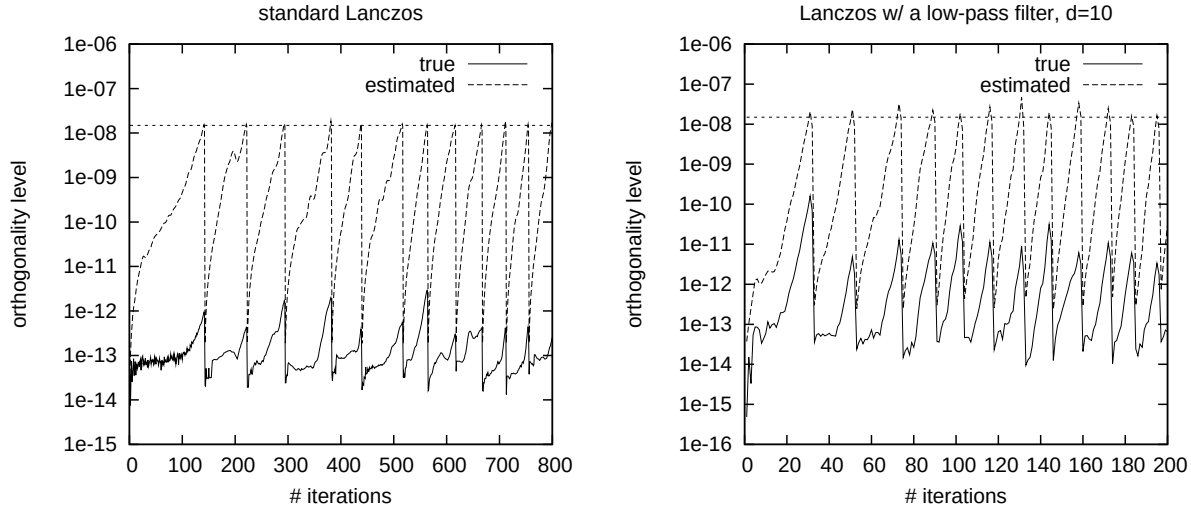

FIG. 3.1. *Level of orthogonality with partial reorthogonalization in the standard Lanczos algorithm (left) and in the filtered Lanczos algorithm (right, with a low-pass filter with $d = 10$).*

The discussion for the filtered Lanczos method is identical since the method consists simply of applying the Lanczos algorithm to $\rho(A)$ instead of A. However, the rate of the loss of orthogonality can be very different. A common observation is that the reorthogonalization frequency increases as the number of iterations grows. An example is illustrated in Figure 3.1, where we use the Hamiltonian $\mathtt{Ge}_{99}\mathtt{H}_{100}$. The plots show the true level of orthogonality κ_i versus the estimated level of orthogonality $\bar{\kappa}_i$ in the standard Lanczos algorithm (left) and in the filtered Lanczos algorithm with a low-pass filter with polynomial degree $d = 10$ (right). In both cases, the estimated level of orthogonality $\bar{\kappa}_i$ is larger than the true level of orthogonality κ_i as needed. In this example, the filtered Lanczos algorithm lost orthogonality faster than the standard Lanczos algorithm.

4. Polynomial filters. Polynomial filtered Lanczos algorithms replace the matrix-vector product Aq_i in the standard Lanczos algorithm by $\rho(A)q_i$, where A is real symmetric or complex Hermitian and ρ is a polynomial. Note that A and $\rho(A)$ share the same eigenvectors, and $\rho(A)$ has eigenvalues $\rho(\lambda_1), \ldots, \rho(\lambda_n)$, where $\lambda_1, \ldots, \lambda_n$ are eigenvalues of A.

Suppose we are given an interval $[a, b]$ which tightly contains the spectrum (i.e., all eigenvalues) of A, and another interval $[\xi, \eta] \subset [a, b]$ in which the eigenvalues are desired. The polynomial $\rho(\lambda)$ is chosen such that $\rho([\xi, \eta])$ is in an extreme region of $\rho([a, b])$. Therefore, because of the nature of the Lanczos algorithm, the eigenvalues of $\rho(A)$ in $\rho([\xi, \eta])$ are approximated first. The corresponding eigenvectors can be used to extract the eigenvalues of A in $[\xi, \eta]$.

If the interval $[a, b]$ is not provided, one can apply the Gershgorin circle theorem to obtain such an interval. However a loose interval $[a, b]$ may decrease the effectiveness of the resulting polynomial filter. Therefore we use a small number of Lanczos steps for approximate smallest and largest eigenvalues, denoted by θ_a and θ_b. The corresponding Ritz vectors and residuals are denoted by y_a and y_b, and $r_a = Ay_a - \theta_a y_a$ and $r_b = Ay_b - \theta_b y_b$, respectively. From a standard theorem, e.g., [21, Theorem 4.5.1], there is a eigenvalue of A in $[\theta_a - \|r_a\|, \theta_a + \|r_a\|]$, and it is likely that this eigenvalue is the smallest one since the Lanczos algorithm approximates the extreme eigenvalues fast. Likewise the largest eigenvalue is likely in $[\theta_b - \|r_b\|, \theta_b + \|r_b\|]$. Hence we can set $a = \theta_a - \|r_a\|$ and $b = \theta_b + \|r_b\|$, which practically bound the spectrum. Alternatively, we may simply use $a = \theta_a$ and $b = \theta_b$. In case any unwanted eigenvalue spills into the region of eigenvalues being approximated, we can reject it posteriorly. See line 24 of Algorithm 2.

4.1. Least-squares filter polynomial. We can naively use a polynomial which approximates a step function to cover the interval of desired eigenvalues $[\xi, \eta]$. Such a polynomial can be computed in the least-squares sense. However, a step function is discontinuous and a polynomial approximation to it will exhibit wide oscillations when the polynomial degree is high. Therefore, we adapt a two-stage process [24]. We first use a smooth function $\phi(\lambda)$, called a *base filter*, similar to the step function in shape, and then use a polynomial $\rho(\lambda)$ to approximate $\phi(\lambda)$ in the least-squares sense. Three cases of $\phi(\lambda)$ are considered in this paper: 1) a low-pass filter for smallest eigenvalues, 2) a middle-pass filter for interior eigenvalues, and 3) a high-pass filter for largest eigenvalues. These cases are illustrated in Figure 4.1.

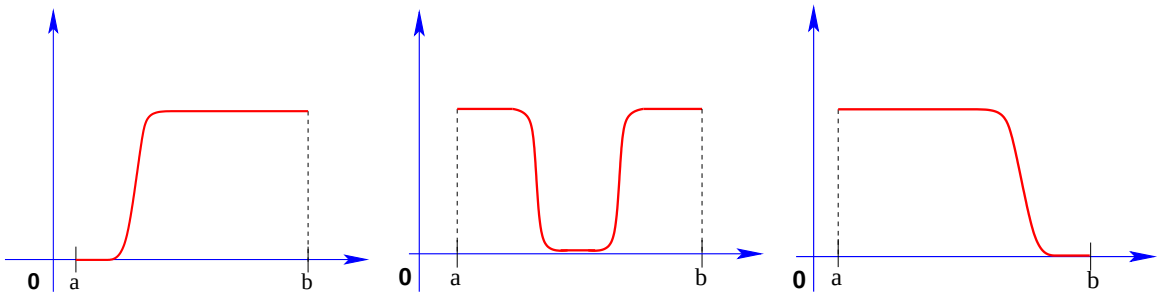

FIG. 4.1. *Low-pass (left), middle-pass (center), and high-pass (right) filter functions* $\phi(\lambda)$.

To find the least-squares polynomial $\rho(\lambda)$ approximation to the base filter $\phi(\lambda)$, we apply the conjugate residual method in *polynomial space* [24]. One can also use the conjugate gradient algorithm instead. However, we favor the conjugate residual algorithm for the least residual norm property. Since the procedure is performed in polynomial space, the matrix is never invoked, and the resulting cost for computing the polynomial is negligible. The polynomial is of the form

$$\rho(\lambda) = 1 - \lambda s(\lambda), \tag{4.1}$$

where $s(\lambda)$ is a polynomial. In other words, our polynomial $\rho(\lambda)$ satisfies $\rho(0) = 1$. The approximation aims to minimize

$$\|\rho(\lambda) - \phi(\lambda)\|_w,$$

where $\|\cdot\|_w$ is the norm induced by some inner product $\langle\cdot,\cdot\rangle_w$ of two functions.

Let $\psi(\lambda) = 1 - \rho(\lambda)$. Then by (4.1), it is equivalent to minimizing

$$\|\psi(\lambda) - \lambda s(\lambda)\|_w.$$

We call $\psi(\lambda)$ the *dual* base filter. Since $\psi(\lambda)$ is approximated by $\lambda s(\lambda)$, we require that $\psi(0) = 0$, or equivalently $\phi(0) = 1$. Consequently, we need $a = 0$ for both high-pass filters and middle-pass filters, where a is the left endpoint of the interval $[a, b]$ containing the eigenvalues of A. This can be achieved by adding $-aI$ to A, if a is not zero. For low-pass filters, we work with the translated matrix $bI - A$, and hence the situation is the same as the case of high-pass filters. Without loss of generality, we assume $a = 0$ hereafter.

4.2. The filtered conjugate residual polynomials algorithm. What is required by the filtered Lanczos procedure is a procedure to compute $\rho(A)q$, for a given vector q. The polynomial $\rho(\lambda)$ does not have to be explicitly formed. The conjugate residual algorithm iterates in the polynomial space can be used to compute $\rho(A)q$. A *corrected* variant, known as the filtered conjugate residual polynomials algorithm, is also proposed in [24, Algorithm 2.3]. The pseudo-code is given in Algorithm 3.

The lines commented with (P) in Algorithm 3 are those that correspond to the conjugate residual algorithm applied in the polynomial space to approximate 1 by $\lambda s(\lambda)$. Therefore,

$$\langle \delta_i(\lambda), \lambda\delta_j(\lambda)\rangle_w = 0, \quad \langle \lambda\pi_i(\lambda), \lambda\pi_j(\lambda)\rangle_w = 0, \qquad i \neq j. \tag{4.2}$$

The conjugate residual method is equivalent to the GMRES algorithm in the Hermitian case [23, Section 6.8], i.e., when the operator is self-adjoint. Recall that the GMRES algorithm minimizes the residual

1: **Input:** matrix $A \in \mathbb{C}^{n\times n}$; vector $q \in \mathbb{C}^n$; base filter $\psi(\lambda) \equiv 1 - \rho(\lambda)$.
2: **Output:** $x_{d+1} = \widehat{s}_{d+1}(A)q$, where $\widehat{s}_{d+1}(\lambda)$ is the polynomial of degree $\le d$ which minimizes $\|\psi(\lambda) - \lambda\widehat{s}(\lambda)\|_w$.
3: $x_0 := 0$
4: $\pi_0 := 1, \quad \delta_0 := 1$ ▷ (P)
5: $p_0 := r_0, \quad r_0 := q$
6: **for all** $j = 0, 1, \ldots, d$ **do**
7: $\quad \alpha_j := \langle \delta_j, \lambda\delta_j\rangle_w / \langle \lambda\pi_j, \lambda\pi_j\rangle_w$ ▷ (P)
8: $\quad \widehat{\alpha}_j := \langle \psi, \lambda\pi_j\rangle_w / \langle \lambda\pi_j, \lambda\pi_j\rangle_w$ ▷ (P)
9: $\quad x_{j+1} = x_j + \widehat{\alpha}_j p_j$
10: $\quad \delta_{j+1} = \delta_j - \alpha_j \lambda\pi_j$ ▷ (P)
11: $\quad r_{j+1} = r_j - \alpha_j A p_j$
12: $\quad \beta_j := \langle \delta_{j+1}, \lambda\delta_{j+1}\rangle_w / \langle \delta_j, \lambda\delta_j\rangle_w$ ▷ (P)
13: $\quad \pi_{j+1} := \delta_{j+1} + \beta_j \pi_j$ ▷ (P)
14: $\quad p_{j+1} := r_{j+1} + \beta p_j$
15: **end for**

Algorithm 3: The filtered conjugate residual polynomials algorithm.

over the Krylov subspace. If we add $s_{j+1} := s_j + \alpha_j\pi_j$ (with base case $s_0 = 0$) after line 7 in Algorithm 3, then s_{j+1} is the minimizer of $\|1 - \lambda s(\lambda)\|_w$ among all polynomials $s(\lambda)$ of degree $\le j$.

In Algorithm 3, lines 5, 11, and 14 resemble lines 4, 10, and 13, respectively. Thus, the conjugate residual iterations can also be performed to compute $s(A)q$ for some polynomial $s(\lambda)$. In practice, the two calculations are decoupled, i.e., the polynomial is obtained in a pre-processing phase (only the (P) part of the code is executed). Then, the calculations $\rho(A)v$ are performed subsequently from the saved polynomials. If the update to x_{j+1} in line 9 uses the coefficient α_j (i.e., $x_{j+1} := x_j + \alpha_j p_j$) instead, then $x_{j+1} \equiv s_{j+1}(A)q$ with $s_{j+1}(\lambda)$ the polynomial minimizing $\|1 - \lambda s(\lambda)\|_w$. However, we are interested in the polynomial $\widehat{s}_{j+1}(\lambda)$ which minimizes $\|\psi(\lambda) - \lambda\widehat{s}(\lambda)\|_w$. Lines 9 and 10 are the required updates as the following theorem from [24, Proposition 2.2] shows.

THEOREM 4.1. *The solution vector x_{j+1} computed at the jth step of Algorithm 3 is of the form $x_{j+1} = \widehat{s}_{j+1}(A)q$, where*

$$\widehat{s}_{j+1}(\lambda) = \widehat{\alpha}_0\pi_0(\lambda) + \cdots + \widehat{\alpha}_j\pi_j(\lambda), \tag{4.3}$$

such that $\widehat{s}_{j+1}(\lambda)$ minimizes $\|\psi(\lambda) - \lambda\widehat{s}(\lambda)\|_w$ among all polynomials $\widehat{s}(\lambda)$ of degree $\le j$.

Note that what we want is $A\widehat{s}_{d+1}(A)q$ with $\lambda\widehat{s}_{d+1}(\lambda)$ approximating $\psi(\lambda)$. The output of Algorithm 3 is $x_d = \widehat{s}_{d+1}(A)q$. So we need to multiply it by A to get $Ax_d = A\widehat{s}_{d+1}(A)q$ and use it to replace Aq in the standard Lanczos process.

A consequence of Theorem 4.1 is that $\lambda\widehat{s}_{d+1}(\lambda)$ converges to $\psi(\lambda)$ uniformly when $d \to \infty$. This can be seen by the Weierstrass approximation theorem, e.g., [11, chapter 6]. Without loss of generality, let q be a unit vector. Then

$$\|\psi(A)q - A\widehat{s}_{d+1}(A)q\|_2 \le \|\psi(A) - A\widehat{s}_{d+1}(A)\|_2 = \max_{i=1,\ldots,n} |\psi(\lambda_i) - \lambda_i\widehat{s}_{d+1}(\lambda_i)|,$$

where $\lambda_1, \ldots, \lambda_n$ are eigenvalues of A. Therefore, $A\widehat{s}_{d+1}(A)q$ approximates $\psi(A)q$ while $\lambda\widehat{s}_{d+1}(\lambda)$ approximates $\psi(\lambda)$.

The iterations in Algorithm 3 can also be used to compute $\lambda\widehat{s}_{d+1}(\lambda)$ for a given $\lambda \in \mathbb{C}$. The computation does not involve matrix-vector product and hence the cost is negligible. This can be very useful for verification purposes and for adjusting the intervals for a base filter to obtain the final polynomial filter by approximation. Details are discussed next.

4.3. Filter polynomial processing and selection. In what follows we consider the situation of interior eigenvalues. The cases for extreme eigenvalues are similar and indeed simpler. Suppose we want the eigenvalues in the interval $[\xi, \eta] \subset [0, b]$. In principle we need a dual base filter ψ with a shape similar to the step function which takes value one in the interval $[\xi, \eta]$ and zero in the intervals $[0, \xi]$ and $[\eta, b]$. We

partition $[0, b]$ into five sub-intervals, $[0, b] \equiv [0, \tau_1] \cup [\tau_1, \tau_2] \cup \cdots \cup [\tau_4, b]$, such that $0 < \tau_1 < \tau_2 < \tau_3 < \tau_4 < b$ and ideally $\xi \in [\tau_1, \tau_2]$ and $\eta \in [\tau_3, \tau_4]$. We let $\psi(\lambda) = 0$ for $\lambda \in [0, \tau_1] \cup [\tau_4, b]$ and $\psi(\lambda) = 1$ for $\lambda \in [\tau_2, \tau_3]$. For the last two intervals, called *transition* intervals, $[\tau_1, \tau_2]$ and $[\tau_3, \tau_4]$, we use the standard Hermite interpolation such that $\psi(\lambda)$ is continuous and smooth with

$$\psi^{(j)}(\tau_i) = 0, \qquad j = 1, \ldots, d_i, \qquad i = 1, \ldots, 4. \tag{4.4}$$

Here d_i's are preset degrees that determine the smoothness of $\psi(\lambda)$.

In a filtered Lanczos procedure, a polynomial $\rho(\lambda) = \lambda s(\lambda)$ is used to approximate the (dual) base filter $\psi(\lambda)$, and conceptually apply the Lanczos algorithm on the matrix $\rho(A)$. In order not to miss eigenvalues in the desired interval $[\xi, \eta]$, it is important that $\rho(z_1) > \rho(z_2)$ for $z_1 \in [\xi, \eta]$ and $z_2 \in [0, \xi) \cup (\eta, b]$, which implies $\rho(\xi) = \rho(\eta)$. This goal can be achieved with appropriate intervals $[\tau_1, \tau_2]$ and $[\tau_3, \tau_4]$.

Our implementation uses a nested loop to determine $\tau_1, \tau_2, \tau_3, \tau_4$. Initially, $\tau_1 := \xi - \delta$ and $\tau_4 := \eta + \delta$ for some small $\delta > 0$. The inner loop is a bisection algorithm which adjusts τ_2, τ_3 until $\rho(\xi)$ and $p(\eta)$ are approximately equal. In each outer iteration, if $\rho(z) > \max\{\rho(\lambda) | \lambda \in [\xi, \eta]\}$ for some $z \in [0, \xi)$, then we decrease τ_1 by a preset small value δ to reduce the oscillations in $[0, \xi)$. Likewise, if $\rho(z) > \max\{\rho(\lambda) | \lambda \in [\xi, \eta]\}$ for some $z \in (\eta, b]$, then we increase τ_4 by δ. We found that an initial defined by $\delta = (\eta - \xi)/100$ is a good choice in practice. For faster convergence, we also increase δ by a preset factor at each outer iteration. The discussion is illustrated by Figure 4.2.

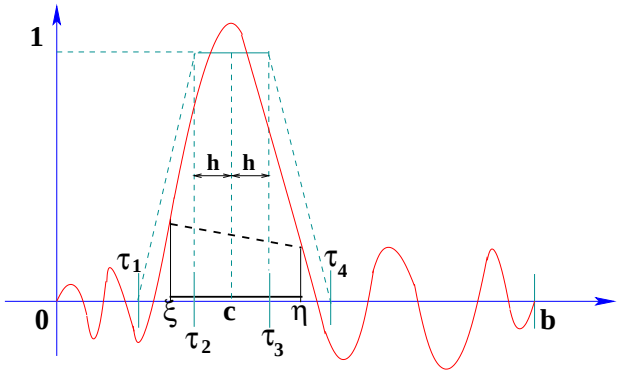

FIG. 4.2. *Illustration of the procedure to set up the partition and the dual base filter* $\psi(\lambda)$.

Some details on the inner loop for iterative bisections for $\rho(\xi) \approx \rho(\eta)$ are given next. We set τ_2 and τ_3 to be of the form $\tau_2 = c - h$ and $\tau_3 = c + h$, where c is initialized as the middle point of τ_1 and τ_2, and h is a small fraction of the interval width. This means that the desired 'plateau' interval for ρ is chosen to be of the form $[\tau_2, \tau_3] = [c - h, c + h]$, where h is fixed and c is determined by an iterative bisection procedure so that the resulting ρ satisfies $\rho(\xi) = \rho(\eta)$ approximately. In practice we usually initialize $h = (\eta - \xi)/20$, which can be decreased by a preset factor in each outer iteration. We summarize the discussion in Algorithm 4.

Three remarks of Algorithm 4 deserve noting. First, the routine get_polynomial(intv, $\{d_i\}_{i=1}^4, d$) invoked in lines 8, 12, 17 determines the dual base filter $\psi(\lambda)$ with degrees d_1, d_2, d_3, d_4 by Hermite interpolation such that $\psi(\lambda)$ is continuous and (4.4) is satisfied. Then $\psi(\lambda)$ is approximated by a polynomial $\rho(\lambda)$ of degree d. As discussed in Section 4.2, the approximation uses a conjugate-residual-type algorithm applied in the polynomial space which is simply Algorithm 3 restricted to the (P) part. This entails a negligible cost since no matrix-vector products are needed. Second, the γ in line 26 and the conditions in lines 27 and 30 can be estimated by uniformly distributed sample points and some tolerance is allowed. Third, in our system a few enhancements are made to improve the quality of the polynomial filter. For example we impose the constraint that a polynomial filter with peaks outside $[\xi, \eta]$ not too close to γ. The details are omitted.

The key point of the above discussion is that the quality of the polynomial filter is important for the convergence of the desired eigenvalues. The pre-processing to get appropriate intervals is inexpensive since all operations are performed in the polynomial space.

1: **Input:** interval of desired eigenvalues $[\xi, \eta]$; eigen-range $[0, b]$; base filter degrees d_1, d_2, d_3, d_4; polynomial filter degree d.
2: **Output:** partition points $\tau_1, \tau_2, \tau_3, \tau_4$ which defines the mid-pass filter.
3: $\delta := (\eta - \xi)/100$; $h := (\eta - \xi)/20$;
4: $\tau_1 := \xi - \delta$; $\tau_4 := \eta + \delta$;
5: **repeat** ▷ Outer loop adjusts τ_1, τ_4.
6: $c_1 := \tau_1 + h$;
7: intv $:= [0, \tau_1, c_1 - h, c_1 + h, \tau_4, b]$;
8: $\rho_1 :=$ get_polynomial(intv, $\{d_i\}_{i=1}^4, d$);
9: $f_1 := \rho_1(\eta) - \rho_1(\xi)$;
10: $c_2 := \tau_4 - h$;
11: intv $:= [0, \tau_1, c_2 - h, c_2 + h, \tau_4, b]$;
12: $\rho_2 :=$ get_polynomial(intv, $\{d_i\}_{i=1}^4, d$);
13: $f_2 := \rho_2(\eta) - \rho_2(\xi)$;
14: **while** $c_2 - c_1 >$ tol **do** ▷ Inner loop determines τ_2, τ_3.
15: $c := (c_1 + c_2)/2$;
16: intv $:= [0, \tau_1, c - h, c + h, \tau_4, b]$;
17: $\rho :=$ get_polynomial(intv, $\{d_i\}_{i=1}^4, d$);
18: $f := \rho(\eta) - \rho(\xi)$;
19: **if** $f \cdot f_2 < 0$ **then**
20: $c_1 := c$; $f_1 := f$;
21: **else**
22: $c_2 := c$; $f_2 := f$;
23: **end if**
24: **end while**
25: $\tau_2 := c - h$; $\tau_3 := c + h$;
26: $\gamma := \min\{\rho(\lambda) | \lambda \in [\xi, \eta]\}$
27: **if** there is $z \in [0, \xi)$ such that $z > \gamma$ **then**
28: $\tau_1 := \tau_1 - \delta$;
29: **end if**
30: **if** there is $z \in (\eta, b]$ such that $z > \gamma$ **then**
31: $\tau_4 := \tau_4 + \delta$;
32: **end if**
33: $\delta := \delta\mu_1$; $h := h/\mu_2$ ▷ A good choice in practice is $\mu_1 = \mu_2 = 1.5$.
34: **until** the values of τ_1, τ_4 are fixed.

Algorithm 4: An iterative process to determine the mid-pass filter.

An example is illustrated in Figure 4.3, where we consider computing 250 interior eigenvalues of a Hamiltonian $\mathtt{Ge}_{99}\mathtt{H}_{100}$ in $[-0.65, 0.0096]$. The characteristics of this matrix, such as the matrix dimension and the number of non-zero elements, are displayed in Table 5.1. These eigenvalues are a window containing the Fermi level, and are part of a Time-Dependent Density Functional Theory (TDDFT) calculation. The full spectrum is contained in $[-1.227, 32.71]$. In all plots we use degrees $d_1 = d_2 = d_3 = d_4 = 10$ for the dual base filter, followed by polynomial approximation with various degrees $d = 20, 30, 50, 100$. It is clear that the higher the degree, the better the shape of the filter.

4.4. The weight function. Algorithm 3 applies the conjugate residual algorithm in the polynomial space and requires the inner product $\langle \cdot, \cdot \rangle_w$ of two functions. In order for this approach to be viable, we need to avoid numerical integration when computing the inner products $\langle \cdot, \cdot \rangle_w$. This is actually achieved by a proper selection of the weight function w which allows to resort to a form of Gauss-Chebyshev quadrature, that provides exact values of the integrals without resorting to numerical integration.

The weight function is defined on each of the sub-intervals $[0, \tau_1], [\tau_1, \tau_2], [\tau_2, \tau_3], [\tau_3, \tau_4], [\tau_4, b]$ of $[0, b]$. There are two reasons for this. First, the dual base filter $\psi(\lambda)$, a piecewise polynomial function, is also needed in the inner product calculation (see line 8 in Algorithm 3). Second, we can assign weights to emphasize or de-emphasize specific sub-intervals. To avoid numerical integration, we expand all functions

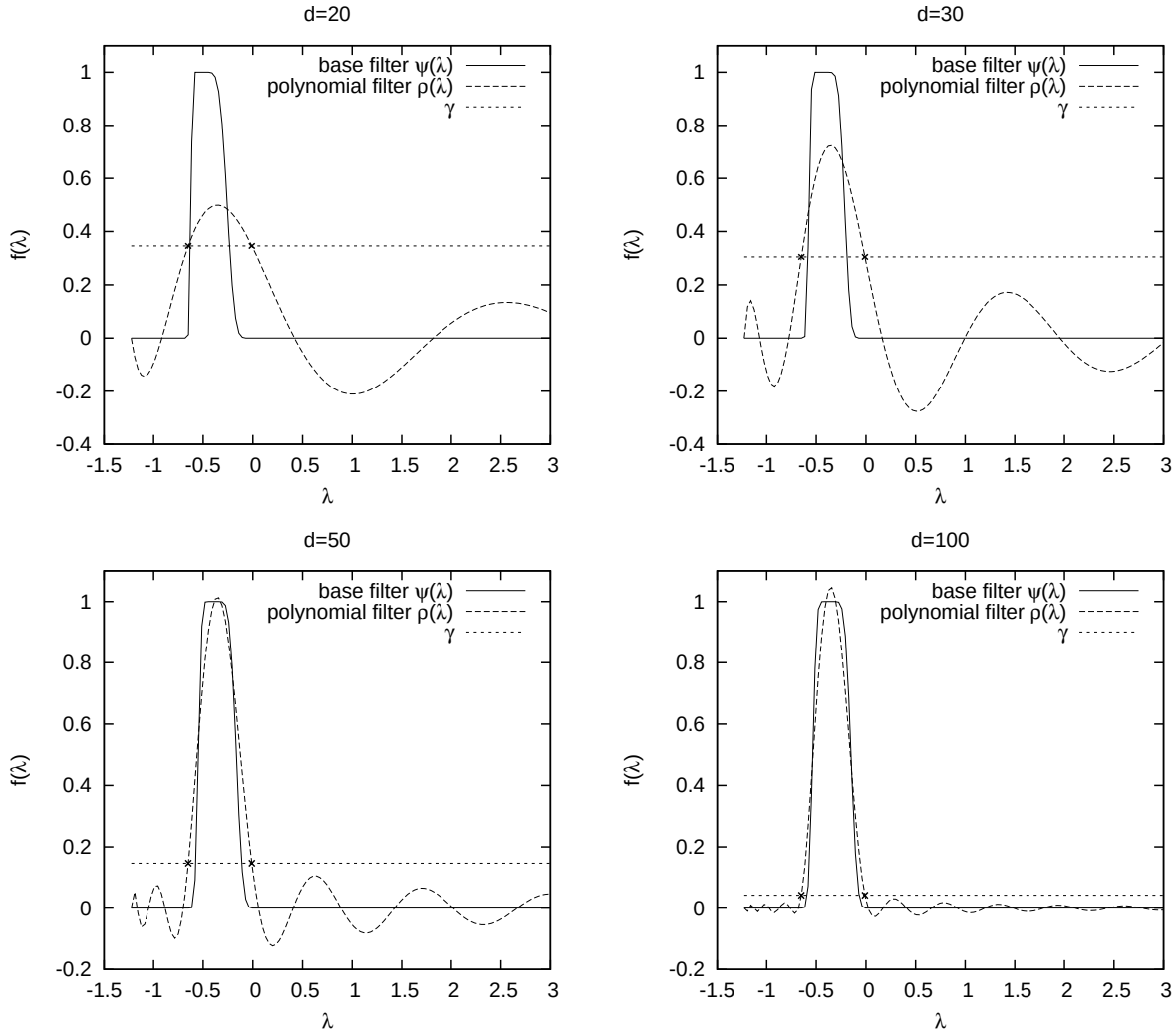

FIG. 4.3. *Four examples of middle-pass filters $\psi(\lambda)$ and their polynomial approximations $\rho(\lambda)$.*

in a basis of orthogonal polynomials for each sub-interval. This in effect amounts to Gaussian quadrature. We use Chebyshev polynomials but other orthogonal polynomials, such as Legendre, can also be exploited.

We denote the partition of $[0, b]$ by $[0, b] \equiv [\tau_0, \tau_1] \cup \cdots \cup [\tau_{l-1}, \tau_l]$, where $\tau_0 = 0$ and $\tau_l = b$. For extreme eigenvalues, there are three sub-intervals, i.e., $l = 3$. For the interior eigenvalues, five sub-intervals are involved, i.e., $l = 5$. In the following discussion, there is no limit of the number of intervals l. The inner-product of two functions $\psi_1(\lambda)$ and $\psi_2(\lambda)$ on each sub-interval $[\tau_{i-1}, \tau_i]$ using Chebyshev weights is defined by

$$\langle \psi_1, \psi_2 \rangle_{[\tau_{i-1}, \tau_i]} = \int_{\tau_{i-1}}^{\tau_i} \frac{\psi_1(\lambda)\psi_2(\lambda)}{\sqrt{(\lambda - \tau_{i-1})(\tau_i - \lambda)}}\, d\lambda.$$

Then the inner product on the interval $[0, \beta]$ is defined as a 'weighted' sum of the inner products on the smaller intervals,

$$\langle \psi_1, \psi_2 \rangle_w = \sum_{i=1}^{l} \frac{2w_i}{\tau_i - \tau_{i-1}} \langle \psi_1, \psi_2 \rangle_{[\tau_{i-1}, \tau_i]} = \sum_{i=1}^{l} \frac{2w_i}{\tau_i - \tau_{i-1}} \int_{\tau_{i-1}}^{\tau_i} \frac{\psi_1(\lambda)\psi_2(\lambda)}{\sqrt{(\lambda - \tau_{i-1})(\tau_i - \lambda)}}\, d\lambda, \tag{4.5}$$

where w_i's are positive weights, and we have incorporated the normalization factors $\frac{2}{\tau_i - \tau_{i-1}}$'s for the integral calculation.

As was mentioned, we expand the functions in a basis of translated Chebyshev polynomials on each of the sub-intervals. When all polynomials are expanded in the proper scaled and shifted Chebyshev basis on each interval, then all inner product operations involved in Algorithm 3 are easily performed with the expansion coefficients [24]. In addition, adding and scaling two expanded polynomials is a trivial operation as it acts only on the expansion coefficients. Finally, multiplying a polynomial by λ can be easily implemented thanks to the 3-term recurrence of Chebyshev polynomials. Details are omitted and can be found in [24].

5. Experiments. The experiments were performed in sequential mode on a machine equipped with two dual-core AMD Opteron(tm) Processors 2214 @ 2.2GHz and 16 gigabytes memory.

We have used a number of real symmetric matrices in our tests. These matrices vary in size, degree of sparsity, and spectrum. We report results for five Hamiltonians ($\texttt{Ge}_{87}\texttt{H}_{76}$, $\texttt{Ge}_{99}\texttt{H}_{100}$, $\texttt{Si}_{41}\texttt{Ge}_{41}\texttt{H}_{72}$, $\texttt{Si}_{87}\texttt{H}_{76}$, and $\texttt{Ga}_{41}\texttt{As}_{41}\texttt{H}_{72}$) from electronic structure calculations, an integer matrix named `Andrews`, and a large matrix of discrete Laplacian by the finite difference method, denoted by `Laplacian`. These matrices are available from the University of Florida sparse matrix collection[2], except for `Laplacian` which we will describe in detail in Section 5.4. The matrix dimension n, the number of non-zero elements nnz, and the range of the spectrum $[a, b]$ for each matrix are listed in Table 5.1. Compared to the Hamiltonians, matrix `Andrews` is smaller and sparser. Matrix `Laplacian` is the largest in dimension and also the sparsest.

TABLE 5.1
Matrix characteristics.

matrix	n	nnz	nnz/n	full eigen-range $[a, b]$	Fermi n_0
$\texttt{Ge}_{87}\texttt{H}_{76}$	112,985	7,892,195	69.85	$[-1.21402, 32.7641]$	212
$\texttt{Ge}_{99}\texttt{H}_{100}$	112,985	8,451,395	74.80	$[-1.22642, 32.7031]$	248
$\texttt{Si}_{41}\texttt{Ge}_{41}\texttt{H}_{72}$	185,639	15,011,265	80.86	$[-1.21358, 49.8185]$	200
$\texttt{Si}_{87}\texttt{H}_{76}$	240,369	10,661,631	44.36	$[-1.19638, 43.0746]$	212
$\texttt{Ga}_{41}\texttt{As}_{41}\texttt{H}_{72}$	268,096	18,488,476	68.96	$[-1.25019, 1300.93]$	200
`Andrews`	60,000	760,154	12.67	$[0, 36.4853]$	N/A
`Laplacian`	1,000,000	6,940,000	6.94	$[0.002907, 11.9971]$	N/A

Each Hamiltonian has a number of occupied states of the molecular system, denoted by n_0. In the density functional theory (DFT) framework, n_0 is usually the number of smallest eigenvalues requested, and the n_0th smallest eigenvalue corresponds to the Fermi level. In other physics models, such as the time-dependent density functional theory (TDDFT), the requested eigenvalues may not be from the smallest end and further interior eigenvalues (i.e., larger than n_0th eigenvalue) may be useful. This characteristic number n_0 for each Hamiltonian is also listed in Table 5.1.

Another property to characterize the matrices is the sparsity pattern. The structure of a Hamiltonian typically forms grids around the diagonal. A typical case, $\texttt{Ge}_{99}\texttt{H}_{100}$, is illustrated in the left of Figure 5.1. The matrix `Andrews` has sparse elements spreading inside the matrix; however, denser elements cluster around the diagonal. For a clear presentation of this character, we show only the top-left 5000×5000 corner of `Andrews` in the middle of Figure 5.1. A three-dimensional discrete Laplacian by the finite difference scheme has three pairs non-zero off-diagonals. In the right of Figure 5.1 is a 125×125 discrete Laplacian (on a $5 \times 5 \times \times 5$ discrete cube) which is a very coarse version of the large `Laplacian`.

5.1. Experimental setting. For each test matrix, an interval $[\xi, \eta]$ is provided in which the eigenvalues and the corresponding eigenvectors are desired. For matrices `Andrews` and `Laplacian`, we arbitrarily set $[\xi, \eta] = [4, 5]$ and $[\xi, \eta] = [1, 1.01]$, respectively. For each Hamiltonian, $[\xi, \eta]$ is chosen to cover from about $\frac{1}{2}n_0$th to about $\frac{3}{2}n_0$th eigenvalues, where n_0 is the number of occupied states listed in Table 5.1. These eigenvalues are part of those which are of interest in TDDFT. Table 5.2 lists the interval $[\xi, \eta]$, the number of eigenvalues in $[\xi, \eta]$, and the number of eigenvalues in $[a, \eta]$ for each matrix.

In all cases we used the IEEE 754 double precision arithmetic. Four methods are compared in our experiments: 1) the polynomial filtered Lanczos algorithm with a mid-pass filter, 2) the polynomial filtered

[2] `http://www.cise.ufl.edu/research/sparse/matrices/`

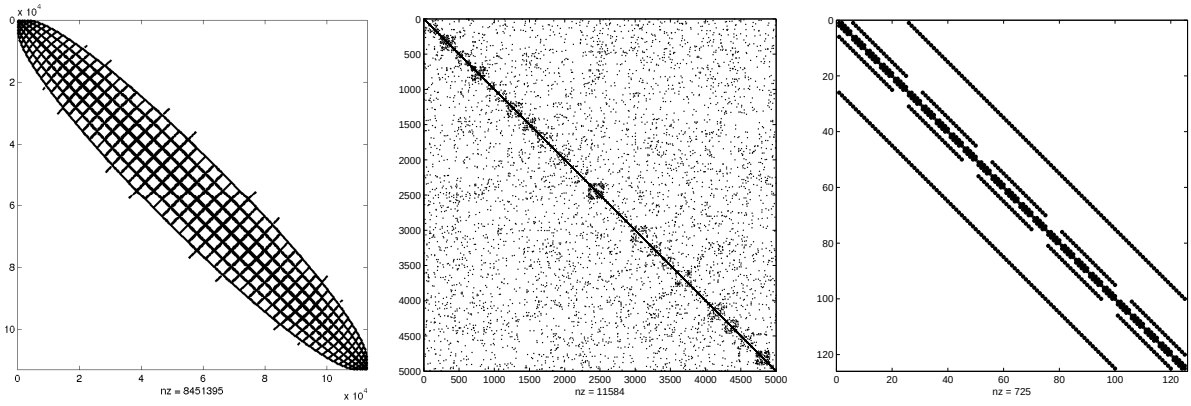

FIG. 5.1. *Structures of* $\mathtt{Ge}_{99}\mathtt{H}_{100}$ *(left),* `Andrews` *in part (middle), and a coarse* `Laplacian` *(right).*

TABLE 5.2
Experimental setting.

matrix	eigen-interval $[\xi, \eta]$	# eig. in $[\xi, \eta]$	# eig. in $[a, \eta]$	# eig. ratio	$\frac{\eta-\xi}{b-a}$	$\frac{\eta-a}{b-a}$
$\mathtt{Ge}_{87}\mathtt{H}_{76}$	$[-0.645, -0.0053]$	212	318	0.67	0.0188	0.0356
$\mathtt{Ge}_{99}\mathtt{H}_{100}$	$[-0.65, -0.0096]$	250	372	0.67	0.0189	0.0359
$\mathtt{Si}_{41}\mathtt{Ge}_{41}\mathtt{H}_{72}$	$[-0.64, -0.00282]$	218	318	0.69	0.0125	0.0237
$\mathtt{Si}_{87}\mathtt{H}_{76}$	$[-0.66, -0.33]$	212	317	0.67	0.0075	0.0196
$\mathtt{Ga}_{41}\mathtt{As}_{41}\mathtt{H}_{72}$	$[-0.64, 0.0]$	201	301	0.67	0.0005	0.0010
`Andrews`	$[4, 5]$	1,844	3,751	0.49	0.0274	0.1370
`Laplacian`	$[1, 1.01]$	276	17,865	0.0154	0.0008	0.0044

Lanczos algorithm with a low-pass filter, 3) the standard Lanczos algorithm without restarting, and 4) the `ARPACK` implementation of the implicitly restarted Lanczos algorithm [14]. For the first three methods, the implementation is in C/C++, and partial reorthogonalization [27] is incorporated to reduce the computational cost, and the convergence is checked every 10 iterations, i.e., stride = 10 in line 8 of Algorithm 2. For `ARPACK`, an eigenvalue software package written in Fortran 77, we set the maximum number of Lanczos basis vectors as three times of the number of requested eigenvalues. The convergence tolerance was set $\sqrt[5]{\epsilon_M^4} \approx 3 \times 10^{-13}$ for all methods.

Note that the tolerance in the convergence check (2.5) is not the tolerance of the residual norm, and it is defined in the relative and average sense. When a polynomial filter $\rho(\lambda)$ is used, the error measurement is indeed of the eigenvalues of $\rho(A)$ instead of the eigenvalues of A. In spite of these factors, we found that in practice a great part (usually more than 80%) of the computed eigenvalues have 10 or more significant digits, and most eigenvalues have at least 8 significant digits.

We used the filtered Lanczos algorithm with either a low-pass filer or a high-pass filter. When a low-pass filter is used, a partition of the spectrum consists of three intervals including one transition interval with two joints, and we set the base filter degrees $d_1 = d_2 = 10$ and interval weights $w_1 = 200$ and $w_2 = w_3 = 1$ throughout. When a mid-pass filter is used, a partition of the spectrum consists of five intervals including two transition intervals with four joints, and we set the base filter degrees $d_1 = d_2 = d_3 = d_4 = 10$, and interval weights $w_1 = w_5 = 200$ and $w_2 = w_3 = w_4 = 1$ straight, except for `Laplacian` where we decreased the value of w_1 and w_5. We applied various polynomial degrees d for each problem.

We also used the Lanczos algorithm without restarting and the `ARPACK` symmetric eigensolver [14] which incorporates the implicit restart technique, to find the extreme eigenvalues in $[a, \eta]$. For Hamiltonians this amounts to computing the $1.5n_0$ smallest eigenvalues.

Compared with low-pass filters, a mid-pass filter has two 'wings', and usually requires a much higher degree polynomial for a comparable filter quality. Therefore when the requested interior eigenvalues are close to one end of the spectrum, it is a good idea to use the filtered Lanczos algorithm with a low-

pass (or high-pass) filter. Indeed when the code is requested to compute the eigenvalues in the (interior) interval $[\xi, \eta]$, it is possible to estimate in advance if it is more economical to compute all eigenvalues in the (extreme) interval $[a, \eta]$. The indicator used for this is the ratio of the number of eigenvalues in $[\xi, \eta]$ and the number of eigenvalues in $[a, \eta]$, listed in the third to the last column of Table 5.2 for each problem. If the indicator is high, then the utilization of a low-pass filter may be preferable over a high-pass filter.

The discussion above assumes that we have a method to estimate the number of eigenvalues of a given interval. A traditional approach to count the eigenvalues in a given interval is by the LDL^T factorization coupled with the Sylvester inertia theorem. See, for example, [21, chapter 3] for an introduction. An alternative is given in [24, section 3.4] which utilizes polynomial filtering.

The required polynomial degree of a quality mid-pass filter highly depends on how narrow the interval $[\xi, \eta]$ is relative to the range of spectrum $[a, b]$. Therefore we list the index $(\eta - \xi)/(b - a)$ for each problem in Table 5.2. As can be expected, the cases $\mathtt{Si}_{41}\mathtt{Ge}_{41}\mathtt{H}_{72}$ and `Laplacian` need much higher degree mid-pass polynomial filters, since they have much lower $(\eta - \xi)/(b - a)$ values than the others. This effect is not so significant for low-pass filters though. When we use a low-pass filter for eigenvalues in $[a, \eta]$, the corresponding index is $(\eta - a)/(b - a)$, which is also displayed in Table 5.2 for each matrix.

The results are summarized in Tables 5.3–5.9. For each method and each parameter setting, we report the number of iterations, the number of matrix-vector products, the required memory, and the CPU time. The filtered Lanczos algorithm needs an interval containing the full spectrum $[a, b]$, which was computed by the standard Lanczos algorithm (i.e., without polynomial filtering). The cost of the matrix-vector products used for determining $[a, b]$ is not significant compared to the filtered Lanczos process, and it is not included in the number of matrix-vector products reported. Therefore, the number of matrix-vector products is always the number of iterations times the polynomial degree.

For `ARPACK` which implements the implicitly restarted Lanczos algorithm, the maximum number of Lanczos basis vectors is known *a priori*. Therefore the required memory can be allocated beforehand. On the other hand, the other methods used in our experiments are based on the unrestarted Lanczos framework, and one cannot easily predict the required number of Lanczos vectors for convergence. We allocate memory for allowing a modest number (e.g., 20) of Lanczos vectors at initialization, and expand the memory by a factor of 1.2 whenever memory is insufficient. The required memory reported is the sum of the memory lastly expanded and the memory to storing the sparse matrix, excluding other relatively small storages, e.g., for the tridiagonal matrix T_m in (2.3) and for the $\bar{\omega}$-recurrence in (3.5).

The total CPU time is reported in seconds for each case in the last columns of Tables 5.3–5.9. This is the CPU time used from start to finish. For the filtered Lanczos method, the CPU time for all $\rho(A)v$ computations is also reported. For the Lanczos method without restarting and `ARPACK`, the polynomial can be regarded as being equal to $\rho(\lambda) = \lambda$. Therefore what is reported is the CPU time for all matrix-vector products. The reorthogonalization cost in CPU time is also provided in the third to the last column. However for `ARPACK` what is reported in that column is the CPU time spent in the routine `DSAUPD` for the implicitly restarted Lanczos process, excluding the matrix-vector products. The cost to obtain the eigenvectors, listed in the second to the last column, may also be significant. It requires solving a symmetric tridiagonal eigenvalue problem and computing a matrix-matrix product for which we use a `BLAS3` routine `DGEMM`. This cost can be quite high if the number of Lanczos basis vectors and the number of requested eigenvalues are large. In the second to last column, the filled number for `ARPACK` is the CPU time used by the routine `DSEUPD` for retrieving eigenvalues and eigenvectors. Since the CPU times for `ARPACK` in these two columns have different meanings, we mark each of them with a †.

Finally, the Lanczos algorithm without restarting and `ARPACK` approximate eigenvalues directly by the corresponding Ritz values. The computation of eigenvectors is not required if only the eigenvalues are requested. On the other hand, the filtered Lanczos algorithm finds the eigenvectors first, and obtains the eigenvalues by the Rayleigh quotients. The computation of eigenvalues are not required if only the invariant subspace is needed.

5.2. Results for Hamiltonians. Tables 5.3–5.7 summarize the results for Hamiltonians $\mathtt{Ge}_{87}\mathtt{H}_{76}$, $\mathtt{Ge}_{99}\mathtt{H}_{100}$, $\mathtt{Si}_{41}\mathtt{Ge}_{41}\mathtt{H}_{72}$, $\mathtt{Si}_{87}\mathtt{H}_{76}$, and $\mathtt{Ga}_{41}\mathtt{As}_{41}\mathtt{H}_{72}$, respectively.

Compared to `ARPACK`, the Lanczos algorithm with partial reorthogonalization achieved great CPU time savings but required much more memory in all cases. This part is consistent with the results by Bekas *et. al.* [2, 3]. It is also clear that in all cases, the filtered Lanczos algorithm achieved great computational savings

TABLE 5.3
Results for $\mathtt{Ge_{87}H_{76}}$.

method	degree	# iter	# matvec	memory (MB)	CPU time (seconds) $\rho(A)v$	reorth	eigvec	total
filtered Lanczos (mid-pass filter)	$d = 20$	930	18,600	941	1,029	64	18	1,147
	$d = 30$	650	19,500	682	1,149	50	11	1,247
	$d = 50$	430	21,500	502	1,191	32	7	1,261
	$d = 100$	310	31,000	377	1,547	23	5	1,601
filtered Lanczos (low-pass filter)	$d = 10$	680	68,000	682	409	99	16	556
	$d = 20$	530	10,600	583	561	46	12	664
	$d = 30$	470	14,100	502	751	40	11	836
	$d = 50$	420	21,000	502	1,107	25	10	1,174
Lanczos w/ partial reoth.		4,880	4,880	4,998	222	1,387	677	2,716
ARPACK		5,365	5,365	924	232	†11,693	†471	12,396

TABLE 5.4
Results for $\mathtt{Ge_{99}H_{100}}$.

method	degree	# iter	# matvec	memory (MB)	CPU time (seconds) $\rho(A)v$	reorth	eigvec	total
filtered Lanczos (high-pass filter)	$d = 20$	1,020	20,400	1,117	1,283	77	23	1,417
	$d = 30$	710	21,300	806	1,343	55	14	1,440
	$d = 50$	470	23,500	508	1,411	32	9	1,479
	$d = 100$	340	34,000	440	1,866	26	7	1,930
filtered Lanczos (low-pass filter)	$d = 10$	770	7,700	806	483	124	21	668
	$d = 20$	600	12,000	688	663	57	21	777
	$d = 30$	530	15,900	590	1,017	49	15	1,123
	$d = 50$	470	23,500	508	1,254	26	13	1,342
Lanczos w/ partial reoth.		5,140	5,140	4,883	234	1,460	793	2,962
ARPACK		6,233	6,233	1,073	298	†17,503	†666	18,468

in both CPU time and memory, compared to the Lanczos algorithm with partial reorthogonalization.

When the filtered Lanczos algorithm was used, the higher the polynomial degree, the fewer the Lanczos iterations, and therefore the less memory required. The CPU time, in this set of experiments, was governed by the cost of $\rho(A)v$, or equivalently the number of matrix-vector products. When a mid-pass filter was used, the optimal polynomial degree for the minimum number of matrix-vector products varied. On the other hand, when a low-pass filter was used, a lower degree polynomial was sufficient for a decent filter, and the optimal polynomial degree for the minimum number of matrix-vector products was $d = 10$, except for $\mathtt{Ga_{41}As_{41}H_{72}}$. The Hamiltonian $\mathtt{Ga_{41}As_{41}H_{72}}$ is different from the others because it has a very narrow interval of requested eigenvalues $[\xi, \eta]$, relative to the range of the spectrum $[a, b]$. Therefore, higher degree polynomials are required for quality filters for $\mathtt{Ga_{41}As_{41}H_{72}}$.

5.3. Results for Andrews. Compared to the five Hamiltonians, the integer matrix `Andrews` is sparser and smaller. Therefore it serves as a good testbed for computing a larger number of further interior eigenvalues. As shown in Tables 5.1 and 5.2, the range of the spectrum is the interval $[a, b] = [0, 36.4853]$, and we arbitrarily requested the eigenvalues in the interval $[\xi, \eta] = [4, 5]$, which contains 1,844 eigenvalues. The interval $[a, \eta] = [0, 5]$ contains 3,751 eigenvalues. The results are reported in Table 5.8.

A few observations from Table 5.8 are the following. First, the eigenvalues in $[a, \eta]$ are about twice those in $[\xi, \eta]$. Using the filtered Lanczos algorithm to find the requested eigenvalues, the memory savings with a higher degree mid-pass filter, as one can expect, is significant compared to using a low-pass filter. Second, compared to Hamiltonians, the matrix `Andrews` is very sparse, and therefore the extra cost per iteration by increasing the polynomial degree is relatively modest. Also compared to the experimental setting for the five Hamiltonians, the number of eigenvalues to find is large, so the cost for the eigenvectors is high. Hence it is attractive to increase the polynomial degree in exchange of fewer iterations and a lower

TABLE 5.5
Results for $\mathtt{Si}_{41}\mathtt{Ge}_{41}\mathtt{H}_{72}$.

method	degree	# iter	# matvec	memory (MB)	CPU time (seconds) $\rho(A)v$	reorth	eigvec	total
filtered Lanczos (mid-pass filter)	$d = 20$	1,280	25,600	2,174	2,661	138	44	2,930
	$d = 30$	830	24,900	1,470	2,619	84	23	2,796
	$d = 50$	470	23,500	847	2,139	38	12	2,240
	$d = 100$	300	30,000	642	3,225	34	8	3,328
filtered Lanczos (high-pass filter)	$d = 10$	760	7,600	1,337	862	181	30	1,150
	$d = 20$	580	11,600	1,143	1,188	114	22	1,391
	$d = 30$	490	14,700	982	1,583	84	19	1,755
	$d = 50$	410	20,500	847	1,877	32	15	1,989
Lanczos w/ partial reoth.		5,680	5,680	9,304	488	2,979	1,626	5,827
ARPACK		6,332	6,332	1,460	572	†26,419	†620	27,612

TABLE 5.6
Results for $\mathtt{Si}_{87}\mathtt{H}_{76}$.

method	degree	# iter	# matvec	memory (MB)	CPU time (seconds) $\rho(A)v$	reorth	eigvec	total
filtered Lanczos (mid-pass filter)	$d = 20$	1,200	24,000	2,726	2,117	202	45	2,437
	$d = 30$	790	23,700	1,631	1,916	103	27	2,101
	$d = 50$	480	24,000	1,171	1,857	63	17	1,985
	$d = 100$	320	32,000	731	2,627	23	11	2,735
filtered Lanczos (low-pass filter)	$d = 10$	750	7,500	1,631	821	293	42	1,236
	$d = 20$	580	11,600	1,380	1,233	171	29	1,501
	$d = 30$	500	15,000	1,171	1,382	101	25	1,586
	$d = 50$	430	21,500	997	1,975	67	22	2,130
Lanczos w/ partial reoth.		5,650	5,650	10,562	384	3,638	417	5,947
ARPACK		6,315	6,315	1,888	479	†37,879	†1,084	39,443

cost for eigenvectors and for reorthogonalization. It is interesting to note that in this case, when a mid-pass filter is used, the cost for reorthogonalization is much lower than the cases with a low-pass filter. Finally, for such a number of eigenvalues and eigenvectors to be computed, both the implicitly restarted Lanczos algorithm (as implemented in ARPACK) and the Lanczos algorithm with partial reorthogonalization would choke if the matrix is much larger because of the the resulting high cost of reorthogonalization and the memory cost.

5.4. Results for the discrete Laplacian. The last experiment we report on is with a large and very sparse three-dimensional discrete Laplacian. The three-dimensional Laplacian Δ is a differential operator which maps a given twice-differentiable function $f : \mathbb{R}^3 \longrightarrow \mathbb{R}$ to $\Delta f \equiv \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2} + \frac{\partial^2 f}{\partial z^2}$. Using the standard finite difference method with the same grid size h in all three dimensions, we obtain a matrix A of size $n = n_x n_y n_z$, see, e.g., [25]. The pattern of a small matrix with $n_x = n_y = n_z = 5$ is shown on the right side of Figure 5.1.

In our experiments, we used the matrix Laplacian with $n_x = n_y = n_z = 100$, resulting in the dimension $n = 1,000,000$ and number of non-zero elements $nnz = 6,940,000$. We arbitrarily requested the eigenvalues in the interval $[\xi, \eta] = [1, 1.01]$, which is quite interior in the range of spectrum $[a, b] = [0.002907, 11.9971]$. In practice it is unlikely to compute all the eigenvalues from the smallest end $a = 0.002907$ up to $\eta = 1.01$. Indeed, the eigenvalues of the discrete Laplacian are explicitly known and we found that there are exactly 17,865 eigenvalues of the $100 \times 100 \times 100$ Laplacian in the interval $[a, \eta] = [0.002907, 1.01]$. Hence we used the filtered Lanczos algorithm with a mid-pass filter and found 276 eigenvalues in $[\xi, \eta] = [1, 1.01]$.

Each Lanczos vector of Laplacian has $n = 1,000,000$ entries, and in double precision arithmetic it requires approximately 8 megabytes of memory. Therefore, 2,000 Lanczos vectors are allowed with 16 gigabytes memory. For 276 eigenvalues the polynomial filter must be of good enough quality so that we

TABLE 5.7
Results for $\mathtt{Ga}_{41}\mathtt{As}_{41}\mathtt{H}_{72}$.

method	degree	# iter	# matvec	memory (MB)	CPU time (seconds) $\rho(A)v$	reorth	eigvec	total
filtered Lanczos (mid-pass filter)	$d = 100$	1,300	130,000	3,116	17,179	202	55	17,553
	$d = 200$	590	118,000	1,615	15,030	83	21	15,230
	$d = 300$	410	123,000	1,171	15,593	52	15	15,748
	$d = 400$	360	144,000	1,026	17,908	53	13	18,067
filtered Lanczos (low-pass filter)	$d = 10$	3,560	35,600	8,872	5,108	1,649	473	7,549
	$d = 20$	1,810	36,200	4,390	4,795	764	125	5,842
	$d = 50$	760	38,000	1,895	4,796	284	39	5,222
	$d = 100$	580	58,000	1,615	6,815	101	30	7,054
Lanczos w/ partial reoth.		5,980	5,980	13,401	726	32,835	1,441	35,939
ARPACK		55,676	55,676	2,069	8,883	†509,552	†906	519,343

TABLE 5.8
Results for `Andrews`.

method	degree	# iter	# matvec	memory (MB)	CPU time (seconds) $\rho(A)v$	reorth	eigvec	total
filtered Lanczos (mid-pass filter)	$d = 20$	9,440	188,800	4,829	2,797	192	4,834	9,840
	$d = 30$	6,040	180,120	2,799	2,429	115	2,151	5,279
	$d = 50$	3,800	190,000	1,947	3,040	65	521	3,810
	$d = 100$	2,360	236,000	1,131	3,757	93	220	4,147
filtered Lanczos (high-pass filter)	$d = 10$	5,990	59,900	2,799	1,152	2,911	2,391	7,050
	$d = 20$	4,780	95,600	2,334	1,335	1,718	1,472	4,874
	$d = 30$	4,360	130,800	2,334	1,806	1,218	1,274	4,576
	$d = 50$	4,690	234,500	2,334	3,187	1,032	1,383	5,918
Lanczos w/ partial reoth.		22,345	22,345	10,312	217	30,455	64,223	112,664
ARPACK		30,716	30,716	6,129	345	†423,492	†18,094	441,934

can obtain the desired precision for eigenvalues with up to 2,000 Lanczos basis vectors. On the other hand, the requested window of eigenvalues $[\xi, \eta] = [1, 1.01]$ is very narrow compared to the range of spectrum $[a, b] = [0.002907, 11.9971]$. It means that we need a very high degree polynomial for a quality filter. In our experiments, we used polynomial filters with degrees $d = 600$, $d = 1,000$, and $d = 1,600$. A part of the filter of degree $d = 1,000$ is plotted in Figure 5.2.

One difficulty was encountered for this problem. When we tested the method for $d = 600$, after 48 eigenvalues in the specified window $[\xi, \eta] = [1, 1.01]$ converged, the other eigenvalues did not show up as filtered Ritz values yet. Therefore our program assumed that all eigenvalues were found and it stopped. The following remedy was incorporated as a result of this issue. Whenever the convergence check was passed, we applied 30 extra iterations to see whether new filtered Ritz values corresponding to the eigenvalues in $[\xi, \eta]$ would come out. The phase of such extra iterations may repeat for a few times until all eigenvalues in $[\xi, \eta]$ converge. The results are reported in Table 5.9. It is clear that the higher the degree of the polynomial, the fewer the iterations and therefore the less the memory requirement for the Lanczos vectors, but the more the matrix-vector products and therefore the longer the CPU time.

TABLE 5.9
Results for `Laplacian`.

Filtered Lanczos	degree	# iter	# matvec	memory (MB)	CPU time (seconds) $\rho(A)v$	reorth	eigvec	total
mid-pass filter	600	1,400	840,000	10,913	97,817	927	241	99,279
	1,000	950	950,000	7,640	119,242	773	162	120,384
	1,600	710	1,136,000	6,358	169,741	722	119	170,856

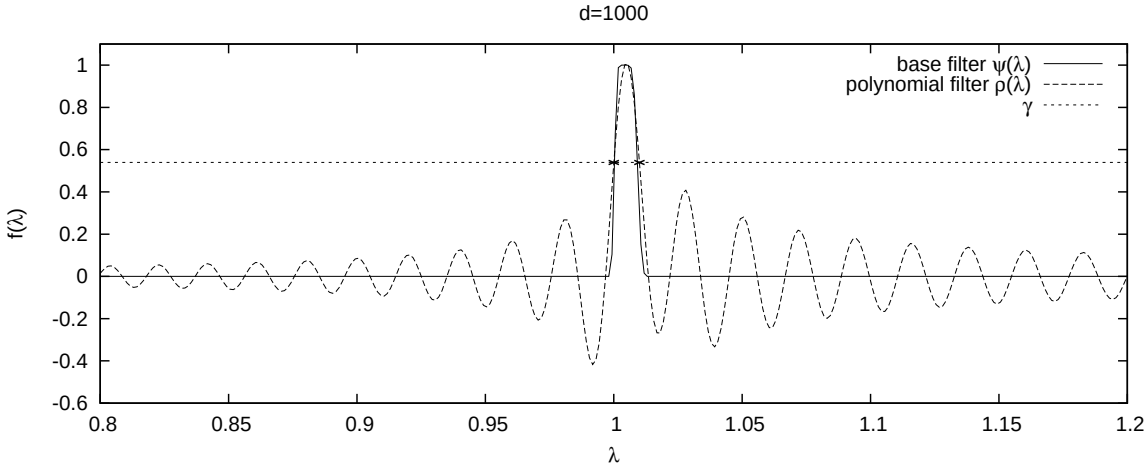

FIG. 5.2. *A high-degree ($d = 1000$) mid-pass filter for* `Laplacian`.

6. Conclusions. We presented a polynomial filtered Lanczos method for computing extreme and interior eigenvalues of large Hermitian matrices. The polynomial is computed by applying a conjugate-residual-type algorithm in polynomial space. The proposed method significantly reduces the number of Lanczos vectors required to compute all the desired eigenvalues and is especially appealing for situations where a large number of eigenvalues and eigenvectors is to be computed. The experimental results indicate that the proposed method can drastically outperform the implicitly restarted Lanczos algorithm implemented in `ARPACK` and also the Lanczos algorithm with partial reorthogonalization. In some cases, such as the example of the Laplacian shown in the numerical experiments, the method may be the only viable option either because factoring the matrix in a shift-and-invert approach is too costly, or because the number of vectors to store and orthogonalize in a standard or restarted Krylov subspace approach is too high. On the negative side, the method is not easily applicable to the generalized eigenvalue problem.

REFERENCES

[1] `http://en.wikipedia.org/wiki/Nastran`.

[2] C. BEKAS, E. KOKIOPOULOU, AND Y. SAAD, *Computation of large invariant subspaces using polynomial filtered Lanczos iterations with applications in density functional theory*, SIAM J. Matrix Anal. and Appl., 1 (2008), pp. 397–418.

[3] C. BEKAS, Y. SAAD, M. L. TIAGO, AND J. R. CHELIKOWSKY, *Computing charge densities with partially reorthogonalized Lanczos*, Comput. Phys. Comm., 171 (2005), pp. 175–186.

[4] J. CULLUM AND R. A. WILLOUGHBY, *Lanczos algorithms for large symmetric eigenvalue computations*, Volumes 1 and 2. Birkhäuser, Boston, 1985.

[5] G. H. GOLUB AND C. F. VAN LOAN, *Matrix Computations*, The Johns Hopkins University Press, Baltimore, MD, 3rd ed., 1996.

[6] GENE H. GOLUB AND CHARLES F. VAN LOAN, *Matrix Computations*, Johns Hopkins University Press, 3rd ed., 1996.

[7] J. GREAR, *Analyses of the Lanczos Algorithm and of the Approximation Problem in Richardson's Method*, PhD thesis, University of Illinois at Urbana-Champaign, 1981.

[8] N. J. HIGHAM, *Accuracy and Stability of Numerical Algorithms*, SIAM Publications, 2nd ed., 2002.

[9] Y. HUANG, D. J. KOURI, AND D. K. HOFFMAN, *Direct approach to density functional theory: iterative treatment using a polynomial representation of the heaviside step function operator*, Chem. Phys. Let., 243 (1995), pp. 367–377.

[10] L. O. JAY, H. KIM, Y. SAAD, AND J. R. CHELIKOWSKY, *Electronic structure calculations using plane wave codes without diagonalization*, Comput. Phys. Comm., 118 (1999), pp. 21–30.

[11] D. KINCAID AND W. CHENEY, *Numerical Analysis: Mathematics of Scientific Computing*, American Mathematical Society, 3rd ed., 2002.

[12] C. LANCZOS, *An iteration method for the solution of the eigenvalue problem of linear differential and integral operators*, J. Res. Nat. Bur. Standards, 45 (1950), pp. 255–282.

[13] R. M. LARSEN, *Lanczos bidiagonalization with partial reorthogonalization*, Technical Report DAIMI PB-357, Department of Computer Science, Aarhus University, 1998.

[14] R. LEHOUCQ, D. C. SORENSEN, AND C. YANG, *ARPACK User's Guide: Solution of Large-Scale Eigenvalue Problems With Implicitly Restarted Arnoldi Methods*, SIAM Publications, Philadelphia, 1998.
URL `http://www.caam.rice.edu/software/ARPACK`.

[15] W. Z. LIANG, C. SARAVANAN, Y. SHAO, A. BELL, AND H. HEAD-GRDON, *Improved Fermi operator expansion methods*

for fast electronic structure calculations, J. Chem. Phys., 119 (2003), pp. 4117–4125.
[16] A. M. N. Niklasson and M. Challacombe, *Density matrix perturbation theory*, J. Phys. Rev. Lett., 92 (2004), p. 193001.
[17] A. M. N. Niklasson, C. J. Tymczak, and M. Challacombe, *Trace resetting density matrix purification in $o(n)$ self-consistent-field theory*, J. Chem. Phy., 118 (2003), pp. 8611–8620.
[18] C. C. Paige, *The Computation of Eigenvalues and Eigenvectors of Very Large Sparse Matrices*, PhD thesis, University of London, 1971.
[19] ———, *Computational variants of the Lanczos method for the eigenproblem*, J. Inst. Math. Appl., 10 (1972), pp. 373–381.
[20] ———, *Error analysis of the Lanczos algorithm for tridiagonalizing a symmetric matrix*, J. Inst. Math. Appl., 18 (1976), pp. 341–349.
[21] B. N. Parlett, *The Symmetric Eigenvalue Problem*, no. 20 in Classics in Applied Mathematics, SIAM Publications, Philadelphia, PA, 1998.
[22] B. N. Parlett and D. S. Scott, *The Lanczos algorithm with selective orthogonalization*, Math. Comp., 33 (1979), pp. 217–238.
[23] Y. Saad, *Iterative methods for sparse linear systems*, SIAM Publications, Philadelphia, PA, 2nd ed., 2003.
[24] ———, *Filtered conjugate residual-type algorithms with applications*, SIAM J. Matrix Anal. and Appl., 28 (2006), pp. 845–870.
[25] ———, *Numerical Methods for Large Eigenvalue Problems*, SIAM Publications, Philadelphia, PA, 2nd ed., 2010.
[26] H. D. Simon, *Analysis of the symmetric Lanczos algorithm with reorthogonalization methods*, Linear Algebra Appl., 61 (1984), pp. 101–132.
[27] ———, *The Lanczos algorithm with partial reorthogonalization*, Math. Comp., 42 (1984), pp. 115–142.
[28] G. W. Stewart, *Matrix Algorithms, Volume II: Eigensystems*, SIAM Publications, Philadelphia, PA, 1st ed., 2001.
[29] H. Takahasi and M. Natori, *Eigenvalue problem of large sparse matrices*, Technical Report 4, Rep. Comp. Center, Univ. Tokyo, 1971–1972.
[30] Y. Zhou, Y. Saad, M. L. Tiago, and J. R. Chelikowsky, *Parallel self-consistent-field calculations via Chebyshev-filtered subspace acceleration*, Phy. rev. E, 74 (2006), p. 066704.
[31] ———, *Self-consistent-field calculation using Chebyshev-filtered subspace iteration*, J. Comp. Phys., 219 (2006), pp. 172–184.