SIAM J. SCI. COMPUT.
Vol. 38, No. 4, pp. A2512–A2534

A THICK-RESTART LANCZOS ALGORITHM WITH POLYNOMIAL FILTERING FOR HERMITIAN EIGENVALUE PROBLEMS*

RUIPENG LI†, YUANZHE XI‡, EUGENE VECHARYNSKI§, CHAO YANG§, AND YOUSEF SAAD‡

Abstract. Polynomial filtering can provide a highly effective means of computing all eigenvalues of a real symmetric (or complex Hermitian) matrix that are located in a given interval, anywhere in the spectrum. This paper describes a technique for tackling this problem by combining a thick-restart version of the Lanczos algorithm with deflation ("locking") and a new type of polynomial filter obtained from a least-squares technique. The resulting algorithm can be utilized in a "spectrum-slicing" approach whereby a very large number of eigenvalues and associated eigenvectors of the matrix are computed by extracting eigenpairs located in different subintervals independently from one another.

Key words. Lanczos algorithm, polynomial filtering, thick-restart, deflation, spectrum slicing, interior eigenvalue problems

AMS subject classifications. 65F15, 65F25, 65F50

DOI. 10.1137/15M1054493

1. Introduction.

The problem of computing a very large number of eigenvalues of a large sparse real symmetric (or complex Hermitian) matrix is common to many applications in the physical sciences. For example, it arises in the density functional theory (DFT) based electronic structure calculations for large molecular systems or solids, where the number of wanted eigenvalues can reach the order of tens of thousands or more. While a number of codes were developed in the past for solving large-scale eigenvalue problems [1, 2, 3, 7, 10, 11, 31, 37], these have not been designed specifically for handling the situation when the number of targeted eigenpairs is extremely large and when the eigenvalues are located well inside the spectrum. It is only in the last few years that this difficult problem has begun to be addressed by algorithm developers [7, 21, 29, 34].

Given an $n \times n$ real symmetric (or complex Hermitian) matrix A, the problem addressed in this paper is to compute *all* of its eigenvalues that are located in a given interval $[\xi, \eta]$, along with their associated eigenvectors. The given interval should be a subinterval of the interval $[\lambda_n, \lambda_1]$, where λ_n and λ_1 are the smallest and largest

*Submitted to the journal's Methods and Algorithms for Scientific Computing section December 28, 2015; accepted for publication (in revised form) June 14, 2016; published electronically August 16, 2016.

http://www.siam.org/journals/sisc/38-4/M105449.html

Funding: The work of the first author was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under contract DE-AC52-07NA27344 (LLNL-JRNL-685856). The work of the second author was supported by the High-Performance Computing, and the Scientific Discovery Through Advanced Computing (SciDAC) program funded by U.S. Department of Energy, Office of Science, Advanced Scientific Computing Research, and Basic Energy Sciences DE-SC0008877. The work of the third and fourth authors was partially supported through the Scientific Discovery Through Advanced Computing (SciDAC) program funded by the U.S. Department of Energy, Office of Science, Advanced Scientific Computing Research DE-AC02-05CH11231.

†Center for Applied Scientific Computing, Lawrence Livermore National Laboratory, Livermore, CA 94551 (li50@llnl.gov).

‡Department of Computer Science and Engineering, University of Minnesota Twin Cities, Minneapolis, MN 55455 (yxi@cs.umn.edu, saad@cs.umn.edu).

§Computational Research Division, Lawrence Berkeley National Laboratory, Berkeley, CA 94720 (evecharynski@lbl.gov, cyang@lbl.gov).

eigenvalues of A, respectively. In this setting, two types of problems can be distinguished. The most common situation treated so far in the literature is the case when the interval $[\xi, \eta]$ is located at one end of the spectrum, i.e., the case when either $\xi = \lambda_n$ or $\eta = \lambda_1$. These are often termed extreme eigenvalue problems. Computing all eigenvalues in a given interval is typically not an issue in this case. Here most methods will work well. The second situation, when $\lambda_n < \xi < \eta < \lambda_1$, is harder to solve in general and is often called an "interior" eigenvalue problem.

Being able to efficiently compute the eigenvalues inside a given interval $[\xi, \eta]$ constitutes a critical ingredient in an approach known as "spectrum slicing" for extracting a large number of eigenpairs. Spectrum slicing is a divide and conquer strategy in which eigenvalues located in different subintervals are computed independently from one another. This is discussed in detail in the next section. A common approach to obtain the part of spectrum in $[\xi, \eta]$ is to apply the Lanczos algorithm or subspace iteration to a transformed matrix $B = \rho(A)$, where ρ is either a rational function or a polynomial.

For extreme intervals, a standard Chebyshev acceleration within a subspace iteration code is exploited in [13] in DFT self-consistent field calculations. For interior intervals, the best known strategy is based on the shift-and-invert transformation, where the Lanczos algorithm or subspace iteration is applied to $B = (A - \sigma I)^{-1}$, with the shift σ selected to point to the eigenvalues in the wanted interval (e.g., σ can be selected as the middle of the interval). The shift-and-invert transformation maps the eigenvalues of A closest to σ to the extreme ones of B. In particular, a restarted shift-and-invert Lanczos method with an inertia-based spectrum slicing was proposed in [4]. This technique may be effective in some situations but it requires a factorization of the matrix $A - \sigma I$ which can be prohibitively expensive for large matrices produced from three-dimensional (3D) models. In contrast, polynomial filtering essentially replaces $(A - \sigma I)^{-1}$ by a polynomial $\rho(A)$ such that all eigenvalues of A inside $[\xi, \eta]$ are transformed into dominant eigenvalues of $\rho(A)$. Our experience in electronic structure calculations indicates that polynomial filtering can perform quite well.

The earlier paper [7] described a filtered Lanczos approach for solving the same problem. Though the basic idea of the present paper is also based on a combination of polynomial filtering and the Lanczos process, the two approaches have fundamental differences. First, the polynomial filters used in [7] are different from those of this paper. They are based on a two-stage approach in which a spline function, called the base filter, is first selected and then a polynomial is computed to approximate this base filter. In the present paper, the filter is a simpler least-squares approximation to the Dirac-delta function with various forms of damping.

The second difference is that the projection method used in [7] is the Lanczos algorithm with partial reorthogonalization [30] and no restart. In contrast, the present paper uses the Lanczos algorithm with a combination of explicit deflation and implicit restart. A subspace iteration approach is also considered. In essence, the projection methods used in this paper are geared toward a limited memory implementation. The choice of the filters puts an emphasis on simplicity relative to [7].

The paper is organized as follows. Section 2 provides a high-level description of the spectrum slicing strategy for computing a large subset of eigenvalues, which is the main motivation for this work. Section 3 introduces a least-squares viewpoint for deriving polynomial filters used in the eigenvalue computation. Section 4 discusses how to efficiently combine the restarted Lanczos algorithm with polynomial filtering and deflation. Numerical examples are provided in section 5 and the paper ends with concluding remarks in section 6.

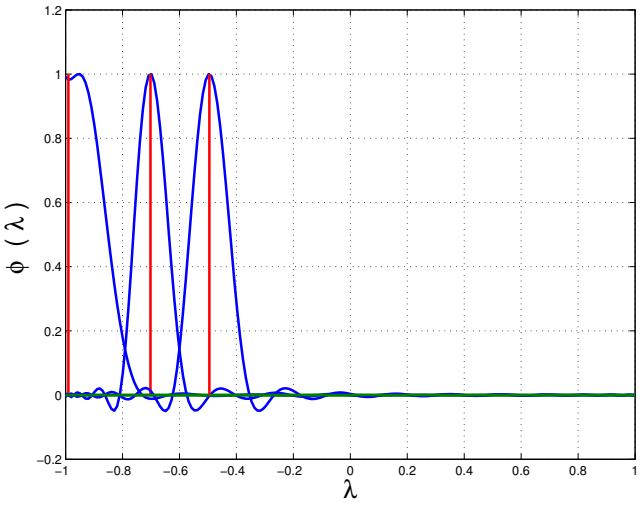

FIG. 1. *Polynomial filters for three different slices.*

2. Motivation: Spectrum slicing. The algorithms studied in this paper are part of a bigger project to develop a parallel package named EVSL (Eigen-Value Slicing Library) for extracting very large numbers of eigenvalues and their associated eigenvectors of a matrix. The basic methodology adopted in EVSL is a divide and conquer approach known as *spectrum slicing.*

2.1. Background. The principle of spectrum slicing is conceptually simple. It consists of dividing the overall interval containing the spectrum into small subintervals and then computing eigenpairs in each subinterval independently.

For this to work, it is necessary to develop a procedure that is able to extract all eigenvalues in a given arbitrary small interval. Such a procedure must satisfy two important requirements. The first is that the eigenpairs in each subinterval under consideration are to be computed independently from any of the other subintervals. The procedure should be as oblivious as possible to any other calculations. The only possible exception is that we may have instances where checking for orthogonality between nearby pairs will be warranted; this is because eigenvectors computed from different slices may not be orthogonal. The other requirement is that the procedure under consideration should not miss any eigenvalue.

The idea of spectrum slicing by polynomial filtering is illustrated in Figure 1. In this approach, the spectrum is first linearly transformed into the interval $[-1, 1]$. This transformation is necessary because the polynomials are often expressed in Chebyshev bases. The interval of interest is then split into p subintervals ($p = 3$ in the illustration). In each of the subintervals we select a filter polynomial of a certain degree so that eigenvalues within the subinterval are amplified. In the illustration shown in Figure 1, the polynomials are of degree 20 (left), 30 (middle), and 32 (right). High intersection points of the curves delineate the final subintervals used.

2.2. Slicing strategies. The simplest way to slice a target interval is to divide it uniformly into several subintervals of equal size. However, when the distribution of eigenvalues is nonuniform, some intervals may contain many more eigenvalues than others. An alternative is required so that the cost of computing eigenvalues within each subinterval is not excessive.

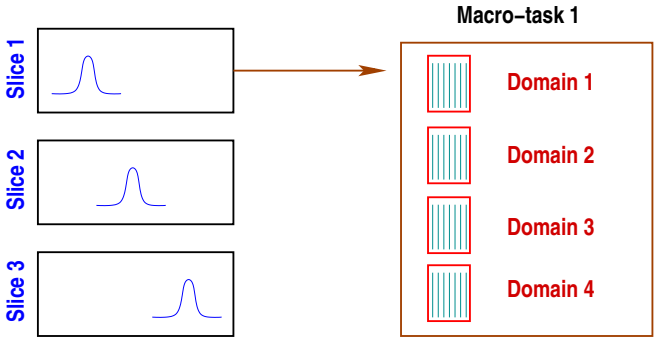

FIG. 2. *The two main levels of parallelism in* EVSL.

A better slicing strategy is to divide the interval based on the distribution of eigenvalues. This can be done by exploiting algorithms for computing density of states (DOS) [16]. We can use Newton's method or a form of bisection to divide the interval in such a way that each subinterval contains roughly the same number of eigenvalues. However, when the eigenvalues are not uniformly distributed, the size of each partitioned subinterval can be quite different. As a result, we may have to use filter polynomials with varying degrees on different slices. One advantage of this slicing strategy is that we can ensure that roughly the same amount of memory is required to compute eigenpairs within each subinterval. This is especially important for a parallel computational environment.

The optimal number of slices into which to partition the spectrum depends on a number of factors such as the efficiency of matrix-vector multiplications, the total number of eigenpairs to be computed, etc. In a parallel computing environment, it also depends on the number of processors available.

2.3. Parallel strategies. If a given interval contains many eigenpairs, we divide it into a number of subintervals and map each subinterval to a group of processors, so that eigenvalues contained in different subintervals can be computed in parallel. This strategy forms the basic configuration of parallel spectrum slicing and is illustrated in Figure 2. A division of the initial interval into subintervals containing roughly the same number of eigenvalues will prevent the orthogonalization and Rayleigh–Ritz procedure from becoming a bottleneck. It will also limit the amount of work (hence the granularity) associated with each concurrent subtask. When the number of subintervals is much larger than the number of processor groups, dynamic scheduling should be used to map each subinterval to a processor group and launch the computation for several subintervals in parallel. Parallelism across slices constitutes only one of the levels of parallelism. Each group of processors will store a copy of the matrix. So another level of parallelism corresponds to parallel matrix and vector operations within each group.

3. Least-squares polynomial filters. The article [7] relied on polynomial filters developed in [25]. These polynomials are computed as least-squares approximations to a base filter, typically a spline function. The base filter transforms eigenvalues in the desired interval to values that are close to one and those outside the interval to values that are close to zero. The advantage of this procedure is that it is very flexible since the base filter can be selected in many different ways to satisfy any desired requirement. On the other hand, the procedure is somewhat complex. In EVSL we

will only use one type of filter, namely, Chebyshev polynomial approximation to the Dirac-delta function.

3.1. Approximating the Dirac-delta function. Since Chebyshev polynomials are defined over the reference interval $[-1, 1]$, a linear transformation is needed to map the eigenvalues of a general Hermitian matrix A to this reference interval. This is achieved by the following transformation:

$$\widehat{A} = \frac{A - cI}{d} \qquad \text{with} \quad c = \frac{\lambda_1 + \lambda_n}{2}, \quad d = \frac{\lambda_1 - \lambda_n}{2}. \tag{3.1}$$

In other words, all work will be performed on the transformed matrix $\widehat{A}$ which now has its eigenvalues in the interval $[-1, 1]$. In practice, the maximum (λ_1) and the minimum (λ_n) eigenvalues of A can be replaced by an upper bound $\tilde{\lambda}_1$ and a lower bound $\tilde{\lambda}_n$ obtained by adequate perturbations of the largest and smallest eigenvalues obtained from a few steps of the standard Lanczos algorithm [38].

Assume that the matrix A is thus linearly transformed so that its eigenvalues are in $[-1, 1]$ and let δ_γ be the Dirac-delta function centered at γ. Then, apart from a scaling by a constant, a formal expansion of this Dirac-delta function takes the form of

$$\rho_k(t) = \sum_{j=0}^{k} \mu_j T_j(t) \tag{3.2}$$

with

$$\mu_j = \begin{cases} \frac{1}{2} & \text{if } j = 0, \\ \cos(j \cos^{-1}(\gamma)) & \text{otherwise,} \end{cases} \tag{3.3}$$

where $T_j(t)$ is the Chebyshev polynomial of the first kind of degree j. Consider the normalized sequence of Chebyshev polynomials,

$$\hat{T}_j = \begin{cases} T_j/\sqrt{\pi} & \text{if } j = 0, \\ T_j/\sqrt{\pi/2} & \text{otherwise,} \end{cases} \tag{3.4}$$

which are orthonormal in that $\langle \hat{T}_i, \hat{T}_j \rangle_w = \delta_{ij}$, where $\langle \cdot, \cdot \rangle_w$ represents the Chebyshev L^2 inner product, and δ_{ij} is the Kronecker delta function. Then, the formal expansion of δ_γ is $\delta_\gamma \approx \sum_{j=0}^{k} \hat{\mu}_j \hat{T}_j$, where $\hat{\mu}_j = \langle \hat{T}_j, \delta_\gamma \rangle_w$. Making the change of variables $t = \cos\theta$ and setting $\theta_\gamma = \cos^{-1}(\gamma)$, we get

$$\hat{\mu}_j = \sqrt{\frac{2 - \delta_{j0}}{\pi}} \int_0^\pi \cos(j\theta) \delta_{\theta_\gamma} d\theta = \sqrt{\frac{2 - \delta_{j0}}{\pi}} \cos(j\theta_\gamma).$$

Thus, it can be seen that $\sum \hat{\mu}_j \hat{T}_j(t) = \frac{2}{\pi} \sum \mu_j T_j(t)$ with μ_j defined by (3.3).

The biggest attraction of the above expansion relative to the one used in [7] is its simplicity. It has no other parameters than the degree k and the point γ on which the Dirac-delta function is centered. It may be argued that it is not mathematically rigorous or permissible to expand the Dirac-delta function, which is a distribution, into orthogonal polynomials. Fortunately, the resulting polynomials obey an alternative criterion.

PROPOSITION 3.1. *Let $\rho_k(t)$ be the Chebyshev expansion defined by* (3.2)–(3.3) *and let $\hat{\rho}_k(t)$ be the polynomial that minimizes*

$$\|r(t)\|_w \tag{3.5}$$

over all polynomials r of degree $\leq k$, such that $r(\gamma) = 1$, where $\|.\|_w$ represents the Chebyshev L^2-norm. Then $\hat{\rho}_k(t) = \rho_k(t)/\rho_k(\gamma)$.

Proof. It is known [33] that the unique minimizer of (3.5) can be expressed via the kernel polynomial formula:

$$\hat{\rho}_k(t) \equiv \frac{\sum_{j=0}^{k} \hat{T}_j(\gamma)\hat{T}_j(t)}{\sum_{j=0}^{k} \hat{T}_j^2(\gamma)}. \tag{3.6}$$

From the expression of $\hat{T}_j$ in terms of T_j in (3.4), the above equation yields

$$\hat{\rho}_k(t) = \frac{T_0(\gamma)T_0(t) + \sum_{j=1}^{k} 2T_j(\gamma)T_j(t)}{T_0^2(\gamma) + \sum_{j=1}^{k} 2T_j^2(\gamma)} = \frac{T_0(t)/2 + \sum_{j=1}^{k} T_j(\gamma)T_j(t)}{T_0^2(\gamma)/2 + \sum_{j=1}^{k} T_j^2(\gamma)}.$$

The numerator is equal to ρ_k defined by (3.2)–(3.3) and so the polynomials ρ_k and $\hat{\rho}_k$ are multiples of one another. ∎

Least-squares polynomials of this type have been advocated and studied in the context of polynomial preconditioning, where $\gamma = 0$; see [24, section 12.3.3] and [23]. In this case the polynomial $\hat{\rho}_k(t)$ is the "residual polynomial" used for preconditioning. A number of results have been established for the special case $\gamma = 0$ in [23]. Here we wish to consider additional results for the general situation where $\gamma \neq 0$.

The starting point is the alternative expression (3.6) of the normalized filter polynomial $\hat{\rho}_k(t)$. Next, we state a corollary of the well-known Christoffel–Darboux formula on kernel polynomials shown in [6, Corollary 10.1.7]. When written for Chebyshev polynomials (3.4) of the first kind, this corollary gives the following result.

LEMMA 3.2. *Let $\hat{T}_j$, $j = 0, 1, \ldots$, be the orthonormal Chebyshev polynomials of the first kind defined in* (3.4). *Then*

$$\sum_{j=0}^{m} [\hat{T}_j(t)]^2 = \frac{1}{2}\left[\hat{T}'_{m+1}(t)\hat{T}_m(t) - \hat{T}'_m(t)\hat{T}_{m+1}(t)\right]. \tag{3.7}$$

This will allow us to analyze the integral of $\hat{\rho}_k^2$ with respect to the Chebyshev weight.

THEOREM 3.3. *Assuming $k \geq 1$, the following equalities hold:*

$$\int_{-1}^{1} \frac{[\hat{\rho}_k(s)]^2}{\sqrt{1-s^2}} ds = \frac{1}{\sum_{j=0}^{k} [\hat{T}_j(\gamma)]^2} \tag{3.8}$$

$$= \frac{2\pi}{(2k+1)} \times \frac{1}{1 + \frac{\sin(2k+1)\theta_\gamma}{(2k+1)\sin\theta_\gamma}}, \tag{3.9}$$

where $\theta_\gamma = \cos^{-1}\gamma$.

Proof. Let $D_k \equiv \sum_{j=0}^{k} [\hat{T}_j(\gamma)]^2$. Since the sequence of polynomials $\hat{T}_j$ is orthonormal, (3.6) implies that

$$\|\hat{\rho}_k\|_w^2 = \left[\frac{1}{D_k}\right]^2 \sum_{j=0}^{k} [\hat{T}_j(\gamma)^2] = \frac{1}{D_k}.$$

This shows (3.8). We now invoke Lemma 3.2 to evaluate D_k. Recall that $T_j'(t) = j\sin(j\theta)/\sin(\theta)$, where $\cos(\theta) = t$. Then we obtain, for any $t = \cos\theta$,

$$\begin{aligned}\sum_{j=0}^{k}[\hat{T}_j(t)]^2 &= \frac{1}{2}\left[\hat{T}'_{k+1}(t)\hat{T}_k(t) - \hat{T}'_k(t)\hat{T}_{k+1}(t)\right] \\ &= \frac{1}{\pi}\frac{(k+1)\sin((k+1)\theta)\cos(k\theta) - k\sin(k\theta)\cos((k+1)\theta)}{\sin\theta} \\ &= \frac{k}{\pi} + \frac{1}{\pi}\frac{\sin((k+1)\theta)\cos(k\theta)}{\sin\theta} \\ &= \frac{k}{\pi} + \frac{1}{\pi}\frac{\sin((k+1)\theta + k\theta) + \sin((k+1)\theta - k\theta)}{2\sin\theta} \\ &= \frac{2k+1}{2\pi} + \frac{1}{\pi}\frac{\sin((2k+1)\theta)}{2\sin\theta}.\end{aligned}$$

This leads to (3.9), by setting $\theta_\gamma = \cos^{-1}\gamma$, and factoring the term $2\pi/(2k+1)$. ∎

The term $\sin((2k+1)\theta)/((2k+1)\sin\theta)$ in the denominator of (3.9) is related to the Dirichlet kernel [6]. It is a Chebyshev polynomial of the second kind of degree $2k$ normalized so that its maximum value is equal to one. Recall that for these polynomials, which are often denoted by U_m, we have $U_m(1) = m+1$ and $U_m(-1) = (-1)^m(m+1)$ and these are the points of largest magnitude of U_m. Hence $\sin((2k+1)\theta)/((2k+1)\sin\theta) = U_{2k}(t)/(2k+1) \le 1$, with the maximum value of one reached. Just as important is the behavior of the minimum value which seems to follow a pattern similar to that of other Gibbs oscillation phenomena observed. Specifically, the minimum value seems to converge to the value $-0.217\ldots$ as the degree increases. Three plots are shown in Figure 3 for illustration.

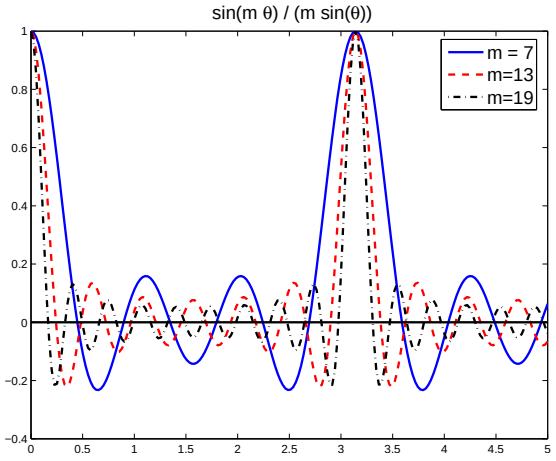

FIG. 3. *The function* $\sin m\theta/(m\sin\theta)$ *for* $m = 7, 13, 19$.

The above analysis shows that the integral in (3.8) decreases roughly like $2\pi/(2k+1)$. It can help understand how the function is expected to converge. For example, in section 3.3 we will exploit the fact that at the (fixed) boundaries ξ, η of the interval the polynomial value is bound to decrease as the degree increases; otherwise the integral would remain higher than a certain value, which would lead to a contradiction.

3.2. Oscillations and damping. As is well known, expansions of discontinuous functions lead to oscillations near the discontinuities known as *Gibbs oscillations*. To alleviate this behavior it is customary to add damping multipliers so that (3.2) is actually replaced by

$$\rho_k(t) = \sum_{j=0}^{k} g_j^k \mu_j T_j(t). \tag{3.10}$$

Thus, the original expansion coefficients μ_j in the expansion (3.2) are multiplied by smoothing factors g_j^k. These tend to be quite small for the larger values of j that correspond to the highly oscillatory terms in the expansion. Jackson smoothing (see, e.g., [22, 12]) is the best known approach. The corresponding coefficients g_j^k are given by the formula

$$g_j^k = \frac{\sin((j+1)\alpha_k)}{(k+2)\sin(\alpha_k)} + \left(1 - \frac{j+1}{k+2}\right)\cos(j\alpha_k),$$

where $\alpha_k = \frac{\pi}{k+2}$. More details on this expression can be seen in [12]. Not as well known is another form of smoothing proposed by Lanczos [14, Chapter 4] and referred to as σ-smoothing. It uses the following simpler damping coefficients instead of g_j^k, called σ factors by the author:

$$\sigma_0^k = 1; \quad \sigma_j^k = \frac{\sin(j\theta_k)}{j\theta_k}, j = 1, \ldots, k, \quad \text{with} \quad \theta_k = \frac{\pi}{k+1}.$$

Figure 4 shows an illustration of three filters of degree 20, one of which is without damping and the others using the Jackson damping and the Lanczos σ-damping, respectively. Scaling is used so that all three polynomials take the same value 1 at γ.

3.3. Choosing the degree of the filter. A good procedure based on polynomial filtering should begin by selecting the polynomial and this cannot be left to the user. It must be done automatically, requiring only the subinterval $[\xi, \eta]$ as input, and optionally the type of damping to be used. The procedure we currently use starts with a low degree polynomial (e.g., $k = 3$) and then increases k until the values of $\rho_k(\xi)$ and $\rho_k(\eta)$ both fall below a certain threshold ϕ (e.g., $\phi = 0.6$ for midinterval

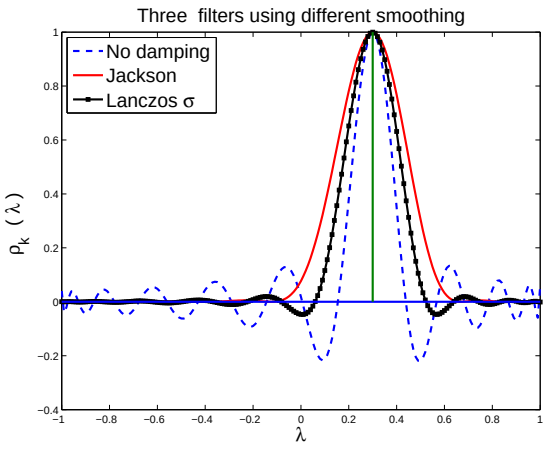

FIG. 4. *Polynomial filters of degree* 20 *using three different damping techniques.*

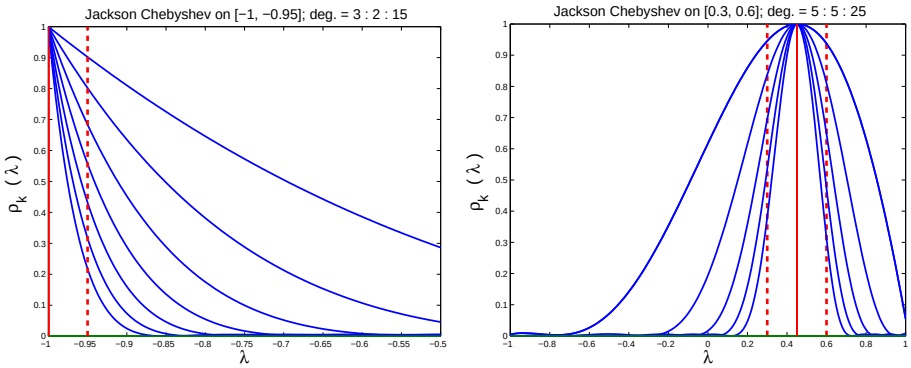

FIG. 5. *Jackson–Chebyshev polynomial filters for a left-end interval (left) and a middle interval (right).*

filters, $\phi = 0.3$ for end intervals). For example, in Figure 5, on the left side we would get a degree of $k = 15$ when $\phi = 0.3$ and on the right side $k = 20$ when $\phi = 0.6$. Once the degree has been selected, a postprocessing is carried out to try to get a "balanced polynomial," i.e., one whose values at ξ and η are the same. This is discussed next.

3.4. Balancing the filter. It is preferable to have a filter ρ_k whose values at ξ and η are the same to facilitate the selection of desired eigenvalues during the Lanczos procedure. For this, a process is required in which the center γ of the delta function is moved. This is not a costly process but its effects are important. For example, it now becomes easy to determine if a computed eigenvalue $\theta = \rho_k(\lambda)$ corresponds to an eigenvalue λ that is inside or outside the interval $[\xi, \eta]$. If $\phi \equiv \rho_k(\xi) = \rho_k(\eta)$, then

$$\lambda \in [\xi, \eta] \quad \text{iff} \quad \theta \equiv \rho_k(\lambda) \geq \phi.$$

Thus, to find all eigenvalues $\lambda \in [\xi, \eta]$ it suffices to find all eigenvalues of $\rho_k(\widehat{A})$ that are greater than or equal to ϕ. In the actual algorithm, this serves as a preselection tool only. All eigenvalues θ_j's that are above the threshold ϕ are preselected. Then the corresponding eigenvectors $\tilde{u}_j$'s are computed along with the Rayleigh quotients $\tilde{u}_j^H A \tilde{u}_j$, which will be ignored if they do not belong to $[\xi, \eta]$. Additional details will be given in the full algorithm described in section 4.2.

To adjust the center γ so that $\rho_k(\xi) = \rho_k(\eta)$, it is important to use the variable $\theta = \cos^{-1} t$ which plays a prominent role in the definition of Chebyshev polynomials. We will denote by θ_x the angle $\theta_x = \cos^{-1}(x)$. Thus $\cos(\theta_\xi) = \xi$, $\cos(\theta_\eta) = \eta$, $\cos(\theta_\gamma) = \gamma$, etc. We then apply Newton's method to solve the equation

$$\rho_k(\xi) - \rho_k(\eta) = 0, \tag{3.11}$$

with respect to θ_γ, through the coefficients μ_j (see (3.10) and (3.3)). The polynomial ρ_k can be written in terms of the variable $\theta = \cos^{-1}(t)$ as

$$\rho_k(\cos\theta) = \sum_{j=0}^{k} g_j^k \cos(j\theta_\gamma)\cos(j\theta).$$

Note that the first damping coefficient g_0^k is multiplied by $1/2$ to simplify notation, so that the first term with $j = 0$ in the expansion is not 1 but rather $1/2$. In this way

(3.11) becomes

$$f(\theta_\gamma) \equiv \sum_{j=0}^{k} g_j^k \cos(j\theta_\gamma)[\cos(j\theta_\xi) - \cos(j\theta_\eta)] = 0. \tag{3.12}$$

In order to solve this equation by Newton's method, we need the derivative of f with respect to θ_γ which is readily computable:

$$f'(\theta_\gamma) = -\sum_{j=0}^{k} g_j^k j \sin(j\theta_\gamma)[\cos(j\theta_\xi) - \cos(j\theta_\eta)].$$

Furthermore, as it turns out, the midangle

$$\theta_c = \frac{1}{2}(\theta_\xi + \theta_\eta) \tag{3.13}$$

already provides a good initial guess for a balanced filter, though the values at the boundaries differ slightly, as shown in Figure 6. In fact, the initial value θ_c in (3.13) is good enough to yield convergence of the Newton iteration in one or two steps in many cases. However, there may be difficulties for low degree polynomials. Thus, if the Newton's scheme fails to converge in two steps, we compute the roots of (3.12) exactly via an eigenvalue problem; see Appendix 6.

4. The thick-restart filtered Lanczos algorithm with deflation. We now discuss how to efficiently combine a thick-restart (TR) version of the Lanczos algorithm with the polynomial filtering and deflation to compute all the eigenpairs inside an arbitrary interval. Since we will apply the Lanczos algorithm to $\rho_k(\widehat{A})$ instead of A, the following discussion will deal with a given Hermitian matrix denoted by B, to remind the reader of the use of spectral transformations in the actual computation. A filtered subspace iteration scheme is also discussed.

4.1. The Lanczos algorithm. The Lanczos algorithm builds an orthonormal basis of the Krylov subspace

$$\mathcal{K}_m = \text{span}\{q_1, Bq_1, \ldots, B^{m-1}q_1\}$$

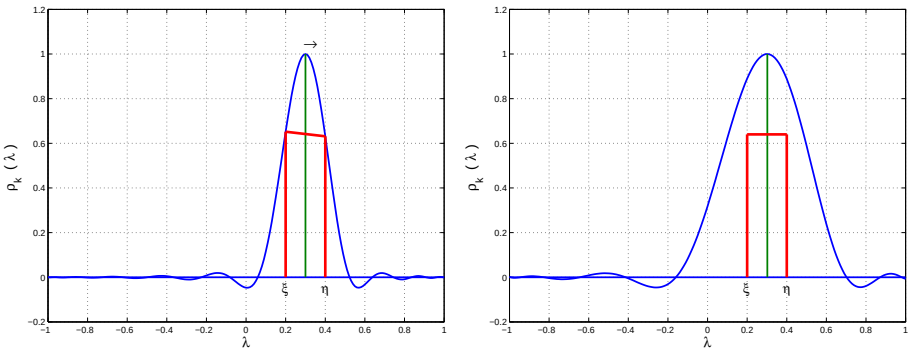

FIG. 6. *Balancing the polynomial filter. Here the center γ of the Dirac-delta function is moved slightly to the right to yield a balanced polynomial.*

by a Gram–Schmidt process in which, at step i, the vector Bq_i is orthogonalized against q_i and (when $i > 1$) against q_{i-1}:

$$\beta_{i+1}q_{i+1} = Bq_i - \alpha_i q_i - \beta_i q_{i-1}. \tag{4.1}$$

As is well known, in *exact* arithmetic, this three-term recurrence would deliver an orthonormal basis $\{q_1, \ldots, q_m\}$ of $\mathcal{K}_m$. In the presence of rounding, orthogonality between the q_i's is quickly lost, and so a form of reorthogonalization is needed in practice and this will be discussed shortly. The Lanczos procedure is sketched in Algorithm 1, in which matrix $Q_j \equiv [q_1, \ldots, q_j]$ contains the basis constructed up to step j as its column vectors.

Algorithm 1. The m-step Lanczos algorithm.

1: Input: a Hermitian matrix $B \in \mathbb{C}^{n\times n}$, and an initial unit vector $q_1 \in \mathbb{C}^n$.
2: $q_0 := 0$, $\beta_1 := 0$
3: **for** $i = 1, 2, \ldots, m$ **do**
4: $\quad w := Bq_i - \beta_i q_{i-1}$
5: $\quad \alpha_i := q_i^H w$
6: $\quad w := w - \alpha_i q_i$
7: $\quad$ Reorthogonalize: $w := w - Q_j(Q_j^H w)$ ▷ classical Gram–Schmidt
8: $\quad \beta_{i+1} := \|w\|_2$
9: $\quad$ **if** $\beta_{i+1} = 0$ **then**
10: $\quad\quad q_{i+1} =$ a random vector of unit norm that is orthogonal to $q_1, \ldots q_i$
11: $\quad$ **else**
12: $\quad\quad q_{i+1} := w/\beta_{i+1}$
13: $\quad$ **end if**
14: **end for**

Let T_m denote the symmetric tridiagonal matrix

$$T_m = \text{tridiag}(\beta_{\text{i}}, \alpha_{\text{i}}, \beta_{\text{i+1}}),$$

where the scalars α_i, β_i are those produced by the Lanczos algorithm. Relation (4.1) can be rewritten in the form

$$BQ_m = Q_m T_m + \beta_{m+1} q_{m+1} e_m^H, \tag{4.2}$$

where e_m is the mth column of the canonical basis and q_{m+1} is the last vector computed by the m-step Lanczos algorithm. Let (θ_i, y_i) be an eigenpair of T_m. In case of ambiguity, $(\theta_i^{(m)}, y_i^{(m)})$ will denote the same eigenpair at the mth step of the process. Then the eigenvalues $\theta_i^{(m)}$, known as Ritz values, will approximate some of the eigenvalues of B as m increases. The vectors $u_i^{(m)} = Q_m y_i^{(m)}$, referred to as Ritz vectors, will approximate the related eigenvectors of B. The Lanczos algorithm quickly yields good approximations to extreme eigenvalues of B while convergence is often much slower for those eigenvalues located deep inside the spectrum [20, 26].

As mentioned, the q_i's form an orthonormal basis in theory, but in practice they lose their orthogonality soon after at least one eigenvector starts converging, leading to an unstable underlying computation. This was studied in detail by Paige in the 1970s [17, 18, 19]. A remedy to this problem is to reorthogonalize the vectors when needed. Since we will use a restarted form of the Lanczos algorithm with moderate dimensions, we decided to apply full reorthogonalization to enforce orthogonality among the q_i's to working precision (line 7 in Algorithm 1).

4.1.1. Thick restart. The method `Filtlan` described in [7] essentially employs the nonrestarted Lanczos algorithm with B in the form of a polynomial in A and the i loop is halted not after m steps but only when all eigenvalues inside the interval are captured. One main issue with `Filtlan` is that if the number of eigenvalues inside the interval is large or the eigenvalues near the boundaries of the interval are clustered, then the number of Lanczos steps required by `Filtlan` may become quite large and this can limit its applicability. This is because reorthogonalization is necessary and becomes expensive. Memory cost becomes another issue as large bases must now be saved. An undesirable feature of nonrestarted procedures is that we do not know in advance how big these bases can be, because the number of steps required for convergence to take place is unknown. Therefore, for very large problems for which limited memory procedures are required, restarting becomes mandatory. In the algorithms described in this paper, we limit the total memory need to that of $m+1$ vectors plus an additional set of n_{ev} vectors for the computed eigenvectors.

We adopt the TR procedure [32, 35] as it blends quite naturally with the filtering technique employed here. The main idea of the TR procedures is to restart not with one vector but with multiple "wanted" Ritz vectors. This technique implements essentially the idea of implicit restarting [15] in a different form.

Here we recall the main steps of TR, assuming for simplicity that there are no locked vectors yet. In this case, suppose that after m steps of the Lanczos algorithm, we end up with, say, l Ritz vectors $u_1, u_2, \ldots, u_l$ that we wish to use as the restarting vectors along with the last vector q_{m+1}. An important observation is that each Ritz vector has a residual in the direction of q_{m+1} [26]. Specifically, we have

$$(B-\theta_i I)u_i = (\beta_{m+1}e_m^H y_i)q_{m+1} \equiv s_i q_{m+1} \quad \text{for} \quad i=1,\ldots,l. \tag{4.3}$$

Let $\hat{q}_i = u_i$ and $\hat{q}_{l+1} = q_{m+1}$. Rewriting (4.3) in a matrix form, it follows that

$$B\hat{Q}_l = \hat{Q}_l\Theta_l + \hat{q}_{l+1}s^H, \tag{4.4}$$

where $\hat{Q}_l = [\hat{q}_1, \ldots, \hat{q}_l]$, the diagonal matrix Θ_l consists of the corresponding Ritz values, and the vector s^H is given by $s^H = [s_1, \ldots, s_l]$. After restart, to perform one step from $\hat{q}_{l+1}$, we proceed as in the Arnoldi algorithm:

$$\beta_{l+2}\hat{q}_{l+2} = B\hat{q}_{l+1} - \sum_{i=1}^{l+1}(\hat{q}_i^H B\hat{q}_{l+1})\hat{q}_i = B\hat{q}_{l+1} - \sum_{i=1}^{l} s_i\hat{q}_i - \alpha_{l+1}\hat{q}_{l+1}, \tag{4.5}$$

where $\alpha_{l+1} = \hat{q}_{l+1}^H B\hat{q}_{l+1}$. Thus, after completing the first step after the restart, we obtain the factorization

$$B\hat{Q}_{l+1} = \hat{Q}_{l+1}\hat{T}_{l+1} + \beta_{l+2}\hat{q}_{l+2}e_{l+1}^H \quad \text{with} \quad \hat{T}_{l+1} = \begin{pmatrix} \Theta_l & s \\ s^H & \alpha_{l+1} \end{pmatrix}. \tag{4.6}$$

From this step on, the algorithm proceeds in the same way as the standard Lanczos algorithm, i.e., $\hat{q}_k$ for $k \geq l+3$ is computed using the three-term recurrence, until the dimension reaches m, and the matrix $\hat{T}_m$ will be completed by the tridiagonal submatrix consisting of α_k and β_k. An induction argument will show that a vector q_j computed in this way will be orthogonal to all previous q_i's.

4.1.2. Deflation. In addition to TR, an essential ingredient to the success of the procedure is the inclusion of "locking," i.e., explicit deflation. Once an eigenpair has converged, we add the eigenvector as a column to a set U of the computed eigenvectors

and exclude it from the search subspace, i.e., we will compute the eigenpairs of $(I - UU^H)B$ in the subsequent iterations. There are several benefits to the use of locking. First, all the locked eigenvalues will be transformed to zeros. This is critical because when deflation is used, an eigenvector cannot be computed a second time. Second, deflation helps to capture any eigenpair that is still left in the interval. In essence this procedure acts as a form of preconditioning by creating larger gaps among the eigenpairs not yet discovered, resulting in faster convergence. Finally, locking provides an efficient means to compute multiple or clustered eigenvalues without resorting to a block method. Since the filter ρ_k may transform some distinct eigenvalues inside $[\xi, \eta]$ into multiple ones, locking becomes necessary to prevent missing eigenvalues.

4.2. Practical details. We now put together the three important ingredients of polynomial filtering, restarting, and deflation. In its simplest form, the filtered Lanczos procedure will first perform m steps of the Lanczos algorithm with the matrix $B = \rho_k(\widehat{A})$, and then it will restart. Once the m Lanczos steps are executed we obtain m Ritz values $\theta_1, \theta_2, \ldots, \theta_m$ such that

$$\theta_1 \geq \theta_2 \geq \cdots \geq \theta_l \geq \rho_k(\xi) > \cdots \geq \theta_m.$$

These are eigenvalues of the tridiagonal matrix T_m in Algorithm 1, with $y_1, y_2, \ldots, y_m$ being the associated eigenvectors. The approximate eigenvectors of B are the Ritz vectors $u_j = Q_m y_j$. Recall from section 3 that when the polynomial filter is constructed, we select a "*bar*" value ϕ, equal to $\rho_k(\xi)$ and $\rho_k(\eta)$, that separates the *wanted* eigenvalues (those in $[\xi, \eta]$) from *unwanted* ones. The eigenvalues θ_i below ϕ, i.e., $\theta_{l+1}, \ldots, \theta_m$, are discarded. For $\theta_1, \ldots, \theta_l$, we compute the associated (unit norm) Ritz vectors $u_1, u_2, \ldots, u_l$ and evaluate their Rayleigh quotients relative to A, which is $\tilde{\lambda}_i = u_i^H A u_i$. A second check is performed at this point that discards any $\tilde{\lambda}_i$ falling outside $[\xi, \eta]$.

For the remaining approximate eigenpairs $(\tilde{\lambda}_i, u_i)$ we compute the residual norms $\|Au_i - \tilde{\lambda}_i u_i\|_2$ associated with the original matrix A. If a Ritz pair has converged, we add the Ritz vector to the "locked set" of vectors. All future iterations will perform a deflation step against such a set. The other, unconverged, candidate eigenvector approximations are added to the "TR set," and the algorithm is then restarted. Thus, there are three types of basis vectors used at any given step: the locked vectors (converged eigenvectors), the vectors selected for TR, and the remaining Lanczos vectors. The whole procedure is sketched in Algorithm 2.

The following notes may help clarify certain aspects of the algorithm. Lines 8–11 execute a cycle of the Lanczos procedure that performs matrix-vector products with $\rho_k(\widehat{A})$ and orthonormalizes the resulting vectors against the vectors in the locked set. In line 13, eigenpairs of tridiagonal matrix T_m are computed. Ritz values that are below threshold ϕ are rejected and the Ritz vectors for the remaining Ritz values are computed. In the next for-loop, Rayleigh quotients associated with the original matrix A are first computed. Those that are outside interval $[\xi, \eta]$ are rejected (line 17). The remaining Ritz pairs will be put either into the locked set if converged (lines 21–22) or into the TR set (lines 24–25) based on their residual norms associated with A.

To reduce computational costs, it is preferable to restart whenever enough eigenvectors have converged for B. In Algorithm 2, the Lanczos process is restarted when the dimension reaches m, i.e., when $i = m$. A check for early restart is also triggered whenever every $Ncycle$ steps have been performed and $Ntest$ steps have been executed since the last restart, i.e., when $Its \bmod Ncycle = 0$ and $i \geq l + Ntest$. In our

Algorithm 2. Filtered Lanczos algorithm with TR and deflation.

1: **Input:** a Hermitian matrix $A \in \mathbb{C}^{n\times n}$ and an initial unit vector $q_1 \in \mathbb{C}^n$.
2: Obtain polynomial $\rho_k(t)$ and "bar" value ϕ
3: $q_0 := 0$, $\beta_1 := 0$, $Its := 0$, $lock := 0$, $l := 0$, $U := [\,]$
4: **while** $Its \le MaxIts$ **do**
5: **if** $l > 0$ **then**
6: Perform TR step (4.5), which results in $\hat{Q}_{l+2}$ and $\hat{T}_{l+1}$ in (4.6)
7: **end if**
8: **for** $i = l+1, \ldots, m$ **do**
9: Perform lines 4–13 of Algorithm 1 with $B = (I - UU^H)\rho_k(\widehat{A})$
10: Set $Its := Its + 1$
11: **end for**
12: Results: $\hat{Q}_{m+1} \in \mathbb{C}^{n\times(m+1)}$ and $\hat{T}_m \in \mathbb{R}^{m\times m}$
13: Compute candidate Ritz pairs, i.e., (θ_j, u_j) with $\theta_j \ge \phi$
14: Set $\hat{Q} := [\,]$ and $l := 0$
15: **for** each candidate pair (θ_j, u_j) **do**
16: Compute $\tilde{\lambda}_j = u_j^H A u_j$
17: **if** $\tilde{\lambda}_j \notin [\xi, \eta]$ **then**
18: Ignore this pair
19: **end if**
20: **if** $\{(\tilde{\lambda}_j, u_j)$ has converged$\}$ **then**
21: Add u_j to locked set $U := [U, u_j]$
22: Set $lock := lock + 1$
23: **else**
24: Add u_j to TR set $\hat{Q} := [\hat{Q}, u_j]$
25: Set $l := l + 1$
26: **end if**
27: **end for**
28: **if** {No candidates found} or {No vectors in TR set} **then**
29: Stop
30: **end if**
31: **end while**

implementation, $Ntest = 50$ and $Ncycle = 30$ are used. In this check, an eigendecomposition of T_i is performed and the number of eigenvalues larger than ϕ, that have converged, is counted. If this number is larger than $(n_{ev} - lock)/2$, where n_{ev} is an estimate of the number of eigenvalues inside the concerned interval given by the DOS algorithm, the algorithm breaks the i loop and restarts. For each candidate pair, only one matrix-vector product (matvec) with A is needed. So, the computational cost of this check is typically very small. Once no candidate eigenpairs are found and no vectors are in the TR set, we still run one more restart, as adopted in [7], to avoid missing eigenpairs.

Another implementation detail is that the orthogonalizations in line 7 of Algorithm 1 and line 9 of Algorithm 2 consist of at most two steps of the classical Gram–Schmidt orthogonalization [8, 9] with the DGKS [5] correction.

4.3. Filtered subspace iteration. The described filtering procedure can also be combined with subspace iteration [26], which may be advantageous in certain applications, such as in electronic structure calculations based on DFT [28]. This

section gives a brief description of this approach to point out the main differences with the TR Lanczos.

Given a matrix B and block X of s approximate eigenvectors, the subspace iteration is of the form $\bar{X} := BX$, where the columns of $\bar{X}$ are then transformed via the Rayleigh–Ritz step into Ritz vectors that are used as an input for the next iteration. The process is repeated until convergence is reached and is guaranteed to deliver eigenvectors corresponding to the s dominant eigenvalues of B, i.e., those with largest magnitude. The method is attractive for its remarkable simplicity, robustness, and low memory requirement [27]. In contrast to the Lanczos algorithm, it provides more flexibility in enabling the use of a good initial guess, which can lead to a significant reduction of computations in some situations. Furthermore, as a block method, the subspace iteration offers an additional level of concurrency and allows for a more intensive use of BLAS3 operations.

When combined with filtering, the subspace iteration is applied to the matrix $B = \rho_k(\widehat{A})$, whose largest eigenvalues are the images of the wanted eigenvalues of A from $[\xi, \eta]$ under the approximate delta function transform $\rho_k(t)$. This behavior is different from that of the filtered Lanczos method, which also converges to the bottom part of the spectrum of B, and therefore requires detection of converged smallest eigenvalues as is done in step 11 of Algorithm 2 that selects the Ritz values above the threshold ϕ.

Note that subspace iteration requires a reasonable estimate s of the number of eigenvalues of A in the interval $[\xi, \eta]$. As has already been discussed, such an estimate can be obtained, from the DOS algorithm [16], and should ideally only slightly exceed the actual number of wanted eigenvalues. Moreover, subspace iteration can also benefit from deflation of the converged eigenpairs [26, 36].

5. Numerical experiments. In this section, we will report numerical results of the polynomial filtered Lanczos method with TR (`FiltLanTR`) in the EVSL package. `FiltLanTR` which was implemented in C and the experiments in sections 5.1–5.2 were performed in sequential mode on a Linux machine with an Intel Core i7-4770 processor and 16 GB memory. The OpenMP experiments in section 5.3 were performed on a node of Cori (Cray XC40 supercomputer, Phase I) at the National Energy Research Scientific Computing Center. Each node of this system has two sockets, with each socket populated with a 16-core Intel "Haswell" processor at 2.3 GHz (i.e., 32 cores per node), and is equipped with 128 GB of DDR4 2133 MHz memory. The code was compiled with the `gcc` compiler using the -O2 optimization level. The convergence tolerance for the residual norm was set at 10^{-8} and the Lanczos σ-damping was used for all polynomial filters.

5.1. 3D Laplacian. We first use a model problem to illustrate the effects of the "bar" value ϕ and the number of slices on the performance of `FiltLanTR`. The model problem was selected as a negative Laplacian $-\Delta$ operator subject to the homogeneous Dirichlet boundary conditions over the unit cube $[0, 1]^3$. Discretizing this operator on a $60 \times 60 \times 60$ grid with the seven-point stencil results in a matrix of size n =216,000. The spectrum of this matrix is inside the interval $[0.00795, 11.99205]$. The goal is to compute all 3406 eigenpairs inside the interval $[0.6, 1.2]$.

5.1.1. Spectrum slicing. We first utilized the DOS algorithm [16] to partition the interval of interest into 10 slices, namely, $[\xi_i, \eta_i]_{i=1,\ldots,10}$ (which are listed in the second column of Table 1), so that each subinterval contains roughly 341 eigenvalues. The computations of the eigenpairs were performed independently for each subinterval. The maximum Krylov subspace dimension and the maximum iteration

TABLE 1
Partitioning $[0.6, 1.2]$ *into* 10 *subintervals for the* 3*D discrete Laplacian example.*

i	$[\xi_i, \eta_i]$	$\eta_i - \xi_i$	$\nu_{[\xi_i,\eta_i]}$
1	[0.60000, 0.67568]	0.07568	337
2	[0.67568, 0.74715]	0.07147	351
3	[0.74715, 0.81321]	0.06606	355
4	[0.81321, 0.87568]	0.06247	321
5	[0.87568, 0.93574]	0.06006	333
6	[0.93574, 0.99339]	0.05765	340
7	[0.99339, 1.04805]	0.05466	348
8	[1.04805, 1.10090]	0.05285	339
9	[1.10090, 1.15255]	0.05165	334
10	[1.15255, 1.20000]	0.04745	348

number were set as $4n_{ev}$ and $16n_{ev}$, respectively, where n_{ev} is the estimated number of eigenvalues in each subinterval. In this example, we have $n_{ev} \approx 341$. To validate the effectiveness of this partitioning, the exact number of eigenvalues in each subinterval, $\nu_{[\xi_i,\eta_i]}$, is provided in the fourth column of the same table. As can be seen, the difference between the approximate and the exact number is less than 10 for most subintervals. In the following discussion, we will refer to each subinterval $[\xi_i, \eta_i]$ by its index i. In order to avoid detecting the same eigenpairs located at the boundaries of these subintervals, we consider half-closed subintervals $[\xi_i, \eta_i)$ for $i = 1, \ldots, 9$ in the actual computation.

5.1.2. Selection of ϕ. In this set of experiments, we fixed the number of slices to be 10 and varied the threshold ϕ for choosing the polynomial degree to study its influence on `FiltLanTR`. Table 2 presents the results with $\phi = 0.6, 0.8, 0.9$. The tables list the degree of the filter polynomials, the number of total iterations, number of matvecs, the CPU time for matrix-vector products, the total CPU time, as well as the maximum and average residual norms of the computed eigenpairs. The following observations can be made. With a fixed ϕ, the iteration number is almost the same for each $[\xi_i, \eta_i]$ (especially for $\phi = 0.6$ and 0.8). However, the filter degree grows for subintervals deeper inside the spectrum. This results in a higher computational cost per iteration, as indicated by the increase in the number of matvecs and the CPU time in columns five and six. Note that the computational accuracy remains identical for different subintervals.

We next study how the statistics in Table 2 change as ϕ increases. We illustrate this change through the bar charts in Figure 7. The first plot in Figure 7 shows that the degree of the filter decreases monotonically as ϕ increases for all 10 intervals. This is expected from the way the polynomial is selected; see section 3.1 and Figure 5. The second subfigure in the same row indicates that fewer iterations are needed if a smaller ϕ is specified. Thus, if available memory is tight, one can trade memory for computation by setting ϕ to a lower value, which would lead to high degree polynomials and fewer iterations of `FiltLanTR`. These first two subfigures lead to the conclusion that the total number of iterations and the polynomial degree change in opposite directions as ϕ changes. It is hard to predict the value of their product, which is the total number of matvecs.

However, the first plot in the second row of Figure 7 indicates that a smaller ϕ can lead to a larger number of matvecs. The second plot in the second row of Figure 7 suggests using a moderate ϕ to reach an optimal performance over all subintervals in terms of the iteration time. The default value of ϕ was set to 0.8 in the subsequent experiments.

TABLE 2

Numerical results for the 3D discrete Laplacian example with $\phi = 0.6, 0.8, 0.9$. The number of slices is 10. The number of eigenvalues found inside each $[\xi_i, \eta_i]$ is equal to the exact number.

(a) $\phi = 0.6$

i	deg	iter	matvec	CPU time (sec)		Residual	
				matvec	total	max	avg
1	172	1567	270055	551.36	808.33	3.30×10^{-09}	2.90×10^{-11}
2	192	1514	291246	594.83	840.29	4.70×10^{-09}	4.79×10^{-11}
3	216	1513	327395	671.19	918.16	3.30×10^{-09}	3.20×10^{-11}
4	237	1516	359863	736.95	973.66	7.80×10^{-09}	7.08×10^{-11}
5	254	1543	392525	803.71	1051.91	8.50×10^{-11}	1.20×10^{-12}
6	272	1511	411622	843.84	1084.82	4.90×10^{-09}	5.57×10^{-11}
7	294	1562	459897	943.33	1202.39	4.50×10^{-09}	3.61×10^{-11}
8	311	1511	470589	965.23	1205.87	1.50×10^{-09}	2.64×10^{-11}
9	325	1565	509308	1045.54	1300.20	1.10×10^{-09}	1.73×10^{-11}
10	361	1538	555948	1140.49	1392.46	3.80×10^{-09}	3.76×10^{-11}

(b) $\phi = 0.8$

i	deg	iter	matvec	CPU time (sec)		Residual	
				matvec	total	max	avg
1	116	1814	210892	430.11	759.24	6.90×10^{-09}	7.02×10^{-11}
2	129	2233	288681	587.14	986.67	5.30×10^{-09}	7.39×10^{-11}
3	145	2225	323293	658.44	1059.57	6.60×10^{-09}	5.25×10^{-11}
4	159	1785	284309	580.09	891.46	3.60×10^{-09}	4.72×10^{-11}
5	171	2239	383553	787.00	1180.67	6.80×10^{-09}	9.45×10^{-11}
6	183	2262	414668	848.71	1255.92	9.90×10^{-09}	1.13×10^{-11}
7	198	2277	451621	922.64	1338.47	2.30×10^{-09}	3.64×10^{-11}
8	209	1783	373211	762.39	1079.30	8.50×10^{-09}	1.34×10^{-10}
9	219	2283	500774	1023.24	1433.04	4.30×10^{-09}	4.41×10^{-11}
10	243	1753	426586	874.11	1184.76	5.70×10^{-09}	1.41×10^{-11}

(c) $\phi = 0.9$

i	deg	iter	matvec	CPU time (sec)		Residual	
				matvec	total	max	avg
1	80	2636	211409	436.11	1006.91	9.90×10^{-09}	1.90×10^{-10}
2	89	2549	227419	468.14	990.79	9.30×10^{-09}	1.62×10^{-10}
3	100	2581	258682	533.39	1076.35	8.80×10^{-09}	1.80×10^{-10}
4	110	2614	288105	594.18	1140.08	9.90×10^{-09}	1.50×10^{-10}
5	118	2606	308109	635.90	1185.56	9.80×10^{-09}	2.17×10^{-10}
6	126	2613	329855	681.94	1235.23	6.40×10^{-09}	1.18×10^{-10}
7	137	2629	360834	746.59	1311.34	8.10×10^{-09}	9.06×10^{-11}
8	145	2603	378099	781.48	1324.18	5.00×10^{-09}	3.45×10^{-11}
9	151	2581	390394	806.61	1339.95	1.00×10^{-08}	3.06×10^{-10}
10	168	2575	433317	894.58	1428.38	8.30×10^{-09}	2.84×10^{-10}

5.1.3. Selection of the number of slices. In the second experiment we fixed the value of ϕ to 0.8 and varied the number of slices. The same 3D discrete Laplacian eigenvalue problem was tested with 2, 5, 15, and 20 slices. For each value of the number of slices, we averaged the statistics across the different slices, and we show the results in Table 3. From the second column of Table 3, we see that on average the degree of the polynomial filters increases (roughly) linearly with the number of slices. This observation is further supported by the first plot in Figure 8.

If the number of iterations per slice decreases linearly with an increasing number of slices, then a constant matvec number and computational time per slice would be expected in the fourth and fifth columns of Table 3. However, this is not the case due to a slower reduction of the iteration number, as shown in the top-right plot

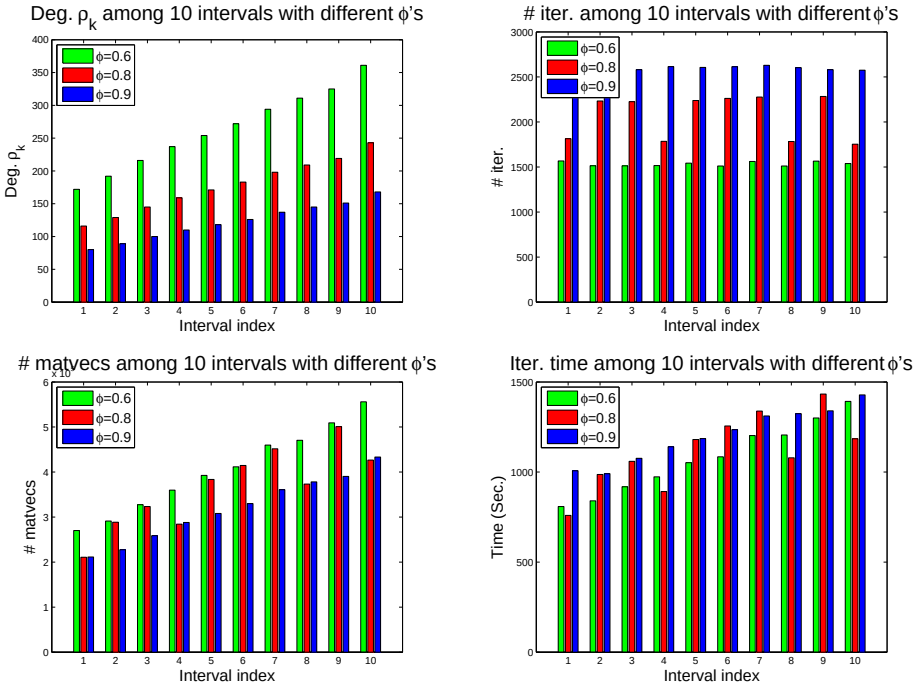

FIG. 7. *Comparison of four statistics in Table* 2 *as* ϕ *increases from* 0.6 *to* 0.9.

TABLE 3

Average statistics per slice for different numbers of slice (n_s)*, for the* 3*D discrete Laplacian example with* $\phi = 0.8$.

n_s	deg	iter	matvec		CPU time	
			Number	Time	Per slice	Total
2	34.5	9284.5	328832.0	681.74	11817.35	23634.69
5	88.0	3891.8	347704.6	715.98	2126.97	10634.85
10	177.2	2065.4	365758.8	747.69	1116.91	11169.13
15	266.1	1351.9	361809.0	746.04	911.54	13673.12
20	356.8	1081.7	392083.3	807.46	909.62	18192.45

of Figure 8. The total iteration time essentially consists of two parts: the matrix-vector product time and the reorthogonalization time. With too few slices, it is reorthogonalization that dominates the overall computational cost. By contrast, too many slices may lead to excessively high degree polynomial filters for each slice, thus increasing the matvec time. This can be seen in the bottom-left subfigure of Figure 8. Finally, when the number of slices doubles for a fixed interval, then, ideally, one would expect that the number of eigenpairs in each slice is almost halved. Then we would hope to reduce the computational cost per slice roughly by half each time the number of slices is doubled, but this is not true in practice. As shown in the bottom-right figure, by using more and more slices, the gain in computational time per slice will be eventually offset by the higher degree of the polynomial filters. We found that a general rule of thumb is to have roughly 200 to 300 eigenvalues per slice.

5.2. Matrices from electronic structure calculations. In this section we compute eigenpairs of five Hamiltonian matrices from electronic structure calculations.

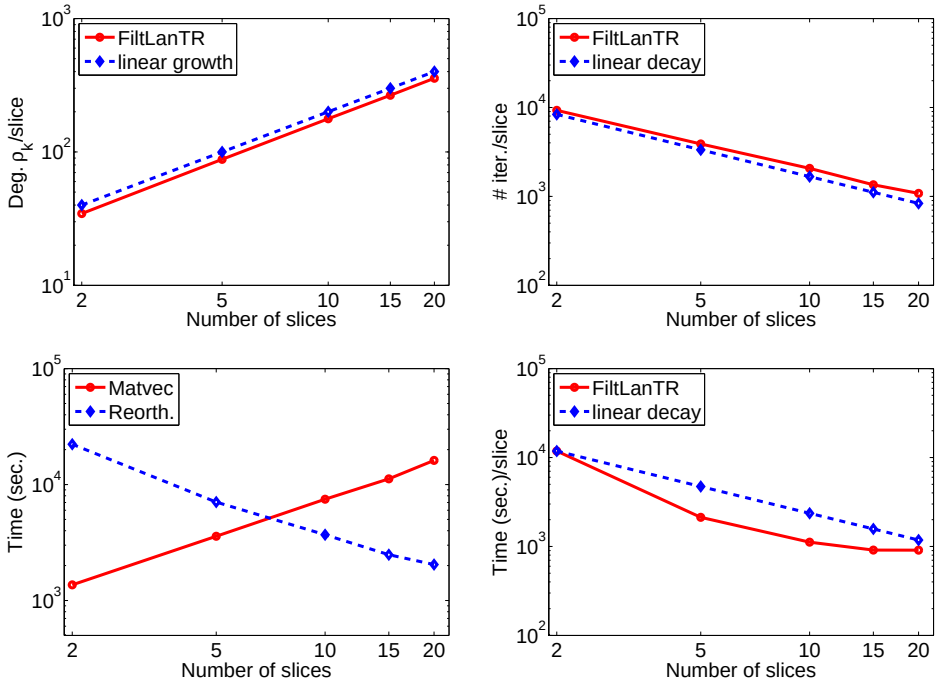

FIG. 8. *Trend plot of computed results in Table 3.*

TABLE 4
Hamiltonian matrices from the PARSEC set

Matrix	n	nnz	$[a, b]$	$[\xi, \eta]$	$\nu_{[\xi,\eta]}$
$Ge_{87}H_{76}$	112,985	7,892,195	$[-1.214, 32.764]$	$[-0.64, -0.0053]$	212
$Ge_{99}H_{100}$	112,985	8,451,295	$[-1.226, 32.703]$	$[-0.65, -0.0096]$	250
$Si_{41}Ge_{41}H_{72}$	185,639	15,011,265	$[-1.121, 49.818]$	$[-0.64, -0.0028]$	218
$Si_{87}H_{76}$	240,369	10,661,631	$[-1.196, 43.074]$	$[-0.66, -0.0300]$	213
$Ga_{41}As_{41}H_{72}$	268,096	18,488,476	$[-1.250, 1300.9]$	$[-0.64, -0.0000]$	201

These matrices are part of the PARSEC set of the University of Florida Sparse Matrix Collection and come from real space discretizations of Hamiltonians. In this set of experiments, we test `FiltLanTR` on a single interval. The matrix size n, the number of nonzeros nnz, the range of the spectrum $[a, b]$, the target interval $[\xi, \eta]$, as well as the exact number of eigenvalues inside this interval are reported in Table 4. Each interval is selected to contain the interval $[0.5n_0, 1.5n_0]$, where n_0 corresponds to the Fermi level of each Hamiltonian [7].

The Hamiltonians under consideration have roughly 70 nonzero entries per row and are much denser compared to the 3D Laplacian example. In fact, the number of nonzeros in their LU factors is around $n^2/5$, rendering the classical "shift-and-invert" Lanczos algorithm very inefficient. We then ran the experiments with `FiltLanTR` following the same settings as in the Laplacian example and report the computational results in Table 5.

Since these Hamiltonians are quite dense, the computational cost from matrix-vector products accounts for a large portion of the overall cost even when low degree filters are in use. This is illustrated in the first four tests. In addition, `FiltLanTR`

TABLE 5
Numerical results for matrices in the PARSEC set with $\phi = 0.8$.

Matrix	deg	iter	matvec	CPU time (sec)		Residual	
				matvec	Total	max	avg
$Ge_{87}H_{76}$	26	1431	37482	282.70	395.91	9.40×10^{-09}	2.55×10^{-10}
$Ge_{99}H_{100}$	26	1615	42330	338.76	488.91	9.10×10^{-09}	2.26×10^{-10}
$Si_{41}Ge_{41}H_{72}$	35	1420	50032	702.32	891.98	3.80×10^{-09}	8.38×10^{-11}
$Si_{87}H_{76}$	30	1427	43095	468.48	699.90	7.60×10^{-09}	3.29×10^{-10}
$Ga_{41}As_{41}H_{72}$	202	2334	471669	8179.51	9190.46	4.20×10^{-12}	4.33×10^{-13}

TABLE 6
Numerical results for matrices in the PARSEC set with $\phi = 0.9$.

Matrix	deg	iter	matvec	CPU time (sec)		Residual	
				matvec	Total	max	avg
$Ge_{87}H_{76}$	18	1981	36027	275.87	462.96	9.20×10^{-09}	5.15×10^{-10}
$Ge_{99}H_{100}$	19	2260	43346	351.68	603.72	7.10×10^{-09}	2.83×10^{-10}
$Si_{41}Ge_{41}H_{72}$	24	1976	47818	679.19	971.19	8.10×10^{-09}	3.17×10^{-10}
$Si_{87}H_{76}$	21	3258	68865	760.71	1297.41	9.90×10^{-09}	7.78×10^{-10}
$Ga_{41}As_{41}H_{72}$	140	2334	326961	5688.46	6703.93	1.20×10^{-10}	2.03×10^{-12}

selects a much higher degree polynomial filter for the Hamiltonian $Ga_{41}As_{41}H_{72}$ as compared to the others. This is because the target interval $[-0.64, 0.0]$ is quite narrow relative to the range of its spectrum $[-1.2502, 1300.93]$. For this kind of problem, it is better to reduce the polynomial degree by increasing the value of ϕ slightly. For example, as shown in Table 6, when ϕ increases to 0.9, the matrix-vector product time drops by 30.45% and the total iteration time drops by 18.04% for $Ga_{41}As_{41}H_{72}$. However, the performance for the other four Hamiltonians deteriorates significantly. Therefore, it is only beneficial to lower the degree of the filter for problems with expensive matrix-vector products and a high degree polynomial filter constructed by `FiltLanTR`.

5.3. Parallel results with OpenMP. As has been demonstrated on the 3D Laplacian example in section 5.1, `FiltLanTR` can be naturally utilized within the divide and conquer strategy for computing a large number of eigenpairs. The ability to target different parts of the spectrum independently opens an additional level of concurrency, which can be efficiently exploited by modern parallel machines.

In this section, we demonstrate the scalability potential of this divide and conquer approach by simply adding a naive OpenMP parallelization across different spectral intervals. In each interval, `FiltLanTR` is invoked, so that different parts of the spectrum are handled independently by concurrent threads run on different computational cores.

As a test problem, we computed 1002 lowest eigenpairs of another matrix from the PARSEC set discussed earlier. This is the matrix SiO which has a size n = 33,401 and nnz = 1,317,655 nonzeros.

Figure 9 demonstrates parallel scaling of such a divide and conquer approach for computing 1002 lowest eigenpairs using 2, 4, and 10 spectral intervals. It can be seen that solution time is decreased by roughly the same factor by which the number of OpenMP threads is increased. The scaling, however, is not optimal because of the unbalanced tasks, where handling some intervals takes longer than others. Note that our test only illustrates parallelism achieved by independently treating different spectral regions. The second level of parallelism available in the reorthogonalization

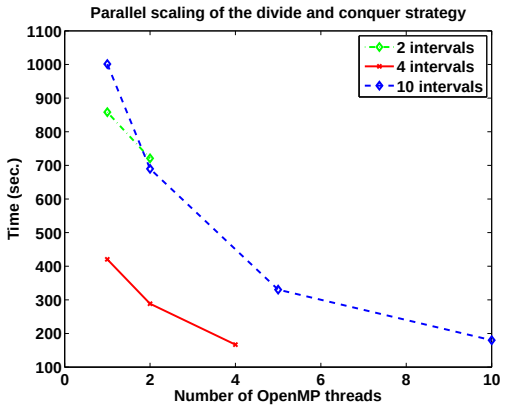

FIG. 9. *An OpenMP parallelization across* 2, 4, *and* 10 *spectral intervals of a divide and conquer approach for SiO matrix;* $n = 33,401$ *and* $nnz = 1,317,655$.

process and matrix-vector products as shown in Figure 2 will be addressed in our future work.

As has been observed in section 5.1, the choice of the number of intervals has a strong impact on the overall performance of `FiltLanTR`, with the recommended number of eigenvalues per slice around 200 to 300. The test in Figure 9 reaffirms this finding. It shows that faster solution times are obtained if 4 intervals, with approximately 250 eigenvalues per slice, are used as opposed to using 2 intervals with around 500 eigenvalues or 10 intervals with roughly 100 eigenvalues in each subinterval.

6. Conclusion. One of the critical components of a spectrum slicing algorithm designed to compute many eigenpairs of a large sparse symmetric matrix is a procedure that can be used to compute eigenvalues contained in a subinterval $[\xi, \eta]$. We developed an efficient way to accomplish this task. Our algorithm is based on combining polynomial filtering with a TR Lanczos algorithm. Thick restarting is employed to limit the memory cost. The polynomial filter that maps eigenvalues within $[\xi, \eta]$ to eigenvalues with the largest magnitude of the transformed problem is obtained from a least-squares approximation to an appropriately centered Dirac-delta distribution. Numerical experiments show that such a construction yields effective polynomial filters which along with a TR Lanczos procedure enable desired eigenpairs to be computed efficiently.

Appendix A. Balancing the filter via an eigenvalue problem. Another way to balance the filter is via the solution of an eigenvalue problem involving a $k \times k$ Hessenberg matrix. Let $t_j = \cos(j\theta_\gamma) = T_j(\gamma)$, for $j = 0, \ldots, k-1$. Then the three-term recurrence of Chebyshev polynomials yields

$$2\gamma \times t_j = \begin{cases} 2t_{j+1} & \text{if} \quad j = 0, \\ t_{j+1} + t_{j-1} & \text{if} \quad j > 0. \end{cases} \tag{6.1}$$

For γ that is a solution of (3.12) we have

$$t_k = -\sum_{j=0}^{k-1} \beta_j t_j \qquad \text{with} \qquad \beta_j = \frac{g_j^k[\cos(j\theta_\xi) - \cos(j\theta_\eta)]}{g_k^k[\cos(k\theta_\xi) - \cos(k\theta_\eta)]}. \tag{6.2}$$

Thus, when $j = k-1$ in (6.1) the following equation holds:

$$2\gamma \times t_{k-1} = -\sum_{j=0}^{k-1} \beta_j t_j + t_{k-2}. \tag{6.3}$$

As a result, denoting by $\mathbf{t}$ the vector with components $t_j, j = 0, \ldots, k-1$, relations (6.1) and (6.3) can be expressed in the form

$$H\mathbf{t} = 2\gamma\mathbf{t}, \tag{6.4}$$

where

$$H = \begin{bmatrix} 0 & 2 & & & & & \\ 1 & 0 & 1 & & & & \\ & 1 & 0 & 1 & & & \\ & & \ddots & \ddots & \ddots & & \\ & & & 1 & 0 & 1 & \\ -\beta_0 & -\beta_1 & \ldots & \ldots & 1-\beta_{k-2} & -\beta_{k-1} \end{bmatrix}.$$

This shows that γ is an eigenvalue of the Hessenberg matrix $H/2$. We can take the eigenvalue of $H/2$ that is closest to the value $\cos^{-1}(\theta_c)$ among those belonging to $[\xi, \eta]$ as the center.

Acknowledgments. YS had stimulating discussions with Jared Aurentz which lead to an improved scheme for locating the correct placement of the Dirac-delta function in the interval $[a, b]$. We thank the editor and two anonymous referees for very helpful comments and suggestions.

REFERENCES

[1] J. Baglama, D. Calvetti, and L. Reichel, *Algorithm* 827*: irbleigs: A MATLAB program for computing a few eigenpairs of a large sparse Hermitian matrix*, ACM Trans. Math. Software, 29 (2003), pp. 337–348.

[2] C. G. Baker, U. L. Hetmaniuk, R. B. Lehoucq, and H. K. Thornquist, *Anasazi software for the numerical solution of large-scale eigenvalue problems*, ACM Trans. Math. Software, 36 (2009), pp. 13:1–13:23.

[3] M. Bollhöfer and Y. Notay, *JADAMILU: A software code for computing selected eigenvalues of large sparse symmetric matrices*, Comput. Phys. Comm., 177 (2007), pp. 951–964.

[4] C. Campos and J. E. Roman, *Strategies for spectrum slicing based on restarted Lanczos methods*, Numer. Algorithms, 60 (2012), pp. 279–295.

[5] J. W. 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.

[6] P. J. Davis, *Interpolation and Approximation*, Blaisdell, Waltham, MA, 1963.

[7] H. R. Fang and Y. Saad, *A filtered Lanczos procedure for extreme and interior eigenvalue problems*, SIAM J. Sci. Comput., 34 (2012), pp. A2220–A2246.

[8] L. Giraud, J. Langou, M. Rozložník, and J. Van Den Eshof, *Rounding error analysis of the classical Gram-Schmidt orthogonalization process*, Numer. Math., 101 (2005), pp. 87–100.

[9] L. Giraud, J. Langou, and M. Rozloznik, *The loss of orthogonality in the Gram-Schmidt orthogonalization process*, Comput. Math. Appl., 50 (2005), pp. 1069–1075.

[10] V. Hernández, J. E. Román, A. Thomás, and V. Vidal, *A Survey of Software for Sparse Eigenvalue Problems*, Tech. report, Universidad Politecnica De Valencia, 2007.

[11] V. Hernández, J. E. Román, and V. Vidal, *SLEPc: A scalable and flexible toolkit for the solution of eigenvalue problems*, ACM Trans. Math. Software, 31 (2005), pp. 351–362.

[12] L. O. Jay, H. Kim, Y. Saad, and J. R. Chelikowsky, *Electronic structure calculations for plane-wave codes without diagonalization*, Comput. Phys. Comm., 118 (1999), pp. 21–30.

[13] L. Kronik, A. Makmal, M. L. Tiago, M. M. G. Alemany, M. Jain, X. Huang, Y. Saad, and J. R. Chelikowsky, *PARSEC: The pseudopotential algorithm for real-space electronic structure calculations: Recent advances and novel applications to nano-structure*, Phys. Stat. Sol. (B), 243 (2006), pp. 1063–1079.

[14] C. Lanczos, *Applied Analysis*, Dover, New York, 1988.
[15] R. B. Lehoucq and D. C. Sorensen, *Deflation techniques for an implicitly restarted Arnoldi iteration*, SIAM J. Matrix Anal. Appl., 17 (1996), pp. 789–821.
[16] L. Lin, Y. Saad, and C. Yang, *Approximating spectral densities of large matrices*, SIAM Rev., 58 (2016), pp. 34–65.
[17] C. C. Paige, *The Computation of Eigenvalues and Eigenvectors of Very Large Sparse Matrices*, Ph.D. thesis, University of London, 1971.
[18] C. C. Paige, *Computational variants of the Lanczos method for the eigenproblem*, J. Inst. Math. Appl., 10 (1972), pp. 373–381.
[19] C. C. Paige, *Error analysis of the Lanczos algorithm for tridiagonalizing a symmetric matrix*, J. Inst. Math. Appl., 18 (1976), pp. 341–349.
[20] B. N. Parlett, *The Symmetric Eigenvalue Problem*, Classics in Appl. Math. 20, SIAM, Philadelphia, 1998.
[21] E. Polizzi, *Density-matrix-based algorithm for solving eigenvalue problems*, Phys. Rev. B, 79 (2009), 115112.
[22] T. J. Rivlin, *An Introduction to the Approximation of Functions*, Dover, New York, 2010.
[23] Y. Saad, *Practical use of polynomial preconditionings for the conjugate gradient method*, SIAM J. Sci. Statist Comput., 6 (1985), pp. 865–881.
[24] Y. Saad, *Iterative Methods for Sparse Linear Systems*, 2nd ed., SIAM. Philadelphia, 2003.
[25] Y. Saad, *Filtered conjugate residual-type algorithms with applications*, SIAM J. Matrix Anal. Appl., 28 (2006), pp. 845–870.
[26] Y. Saad, *Numerical Methods for Large Eigenvalue Problems*, Classics in Appl. Math. 66, SIAM, Philadelphia, 2011.
[27] Y. Saad, *Analysis of subspace iteration for eigenvalue problems with evolving matrices*, SIAM J. Matrix Anal. Appl., 37 (2016), pp. 103–122.
[28] Y. Saad, J. R. Chelikowsky, and S. M. Shontz, *Numerical methods for electronic structure calculations of materials*, SIAM Rev., 52 (2010), pp. 3–54.
[29] T. Sakurai and H. Sugiura, *A projection method for generalized eigenvalue problems using numerical integration*, J. Comput. Appl. Math., 159 (2003), pp. 119–128.
[30] H. D. Simon, *The Lanczos algorithm with partial reorthogonalization*, Math. Comp., 42 (1984), pp. 115–142.
[31] A. Stathopoulos and J. R. McCombs, *PRIMME: Preconditioned iterative multimethod eigensolver: Methods and software description*, ACM Trans. Math. Software, 37 (2010), pp. 21:1–21:30.
[32] A. Stathopoulos, Y. Saad, and K. Wu, *Dynamic thick restarting of the Davidson, and the implicitly restarted Arnoldi methods*, SIAM J. Sci. Comput., 19 (1998), pp. 227–245.
[33] E. L. Stiefel, *Kernel polynomials in linear algebra and their numerical applications*, U.S. Nat. Bur. Standards Appl. Math. Ser., 49 (1958), pp. 1–22.
[34] E. Vecharynski, C. Yang, and J. E. Pask, *A projected preconditioned conjugate gradient algorithm for computing many extreme eigenpairs of a hermitian matrix*, J. Comput. Phys., 290 (2015), pp. 73–89.
[35] K. Wu and H. Simon, *Thick-restart Lanczos method for large symmetric eigenvalue problems*, SIAM J. Matrix Anal. Appl., 22 (2000), pp. 602–616.
[36] Y. Xi and Y. Saad, *Least-Squares Rational Filters for the Solution of Interior Eigenvalue Problems*, Tech. report, Department of Computer Science and Engineering, University of Minnesota, 2015.
[37] I. Yamazaki, Z. Bai, H. Simon, L.-W. Wang, and K. Wu, *Adaptive projection subspace dimension for the thick-restart Lanczos method*, ACM Trans. Math. Software, 37 (2010), pp. 27:1–27:18.
[38] Y. Zhou, Y. Saad, M. L. Tiago, and J. R. Chelikowsky, *Self-consistent-field calculation using Chebyshev-filtered subspace iteration*, J. Comput. Phys., 219 (2006), pp. 172–184.