SIAM J. MATRIX ANAL. APPL.
Vol. 28, No. 3, pp. 845–870

FILTERED CONJUGATE RESIDUAL-TYPE ALGORITHMS WITH APPLICATIONS[*]

YOUSEF SAAD[†]

Abstract. It is often necessary to filter out an eigenspace of a given matrix A before performing certain computations with it. The eigenspace usually corresponds to undesired eigenvalues in the underlying application. One such application is in information retrieval, where the method of latent semantic indexing replaces the original matrix with a lower-rank one using tools based on the singular value decomposition. Here the low-rank approximation to the original matrix is used to analyze similarities with a given query vector. Filtering has the effect of yielding the most relevant part of the desired solution while discarding noise and redundancies in the underlying problem. Another common application is to compute an invariant subspace of a symmetric matrix associated with eigenvalues in a given interval. In this case, it is necessary to filter out eigenvalues that are not in the interval of the wanted eigenvalues. This paper presents a few conjugate gradient–like methods to provide solutions to these types of problems by iterative procedures which utilize only matrix-vector products.

Key words. conjugate residual, conjugate gradient, polynomial filtering, principal component analysis, interior eigenvalues

AMS subject classifications. 65F10, 65F20, 65F50

DOI. 10.1137/060648945

1. Introduction.

A number of applications in science and engineering require filtering, a process by which a matrix A is replaced by a function $\phi(A)$, where the filter function ϕ has the desirable property of filtering out certain unwanted eigenvalues. For example, this arises when computing the vector $A_k b$, where A_k is a rank-k approximation to A, and b a certain vector. Typically, A_k is the rank-k approximation that is the closest to A in the 2-norm sense, and it can be obtained from the singular value decomposition (SVD) of A. These methods include the techniques based on principal component analysis (PCA), such as, for example, latent semantic indexing (LSI); see [6].

Classical methods based on the SVD consist of approximating A by a rank-k matrix obtained by retaining only the k largest singular values in the SVD. For example, if $A = U\Sigma V^T$ is the SVD of A, where U and V are unitary and Σ is diagonal, then methods based on PCA replace A by $A_k = U\Sigma_k V^T$, where Σ_k is obtained from Σ by setting all singular values $\sigma_i < \sigma_k$ to zero. This truncated SVD (TSVD) technique amounts to replacing Ab by $s(A)b$, where $s(\lambda)$ is a step function that has value 1 for $\lambda \geq \sigma_k$ and zero for $\lambda < \sigma_k$. An obvious limitation of the SVD-based approach is its excessive computational cost for large matrices since in principle, at least, a complete SVD factorization of A is required.

Another important use of filtering is when computing large invariant subspaces. Here, one can think of a Lanczos-type procedure applied to the matrix $p(A)$ instead of A, where p is a low-degree polynomial. This approach has been successfully used

[*]Received by the editors January 3, 2006; accepted for publication (in revised form) by L. Reichel March 6, 2006; published electronically October 16, 2006. This work was supported by NSF grants ACI-0305120, DMR-0325218, and DMS 0510131; by the DOE under grant DE-FG02-03ER25585; and by the Minnesota Supercomputing Institute.
http://www.siam.org/journals/simax/28-3/64894.html

[†]Department of Computer Science and Engineering, University of Minnesota, 200 Union Street S.E., Minneapolis, MN 55455 (saad@cs.umn.edu).

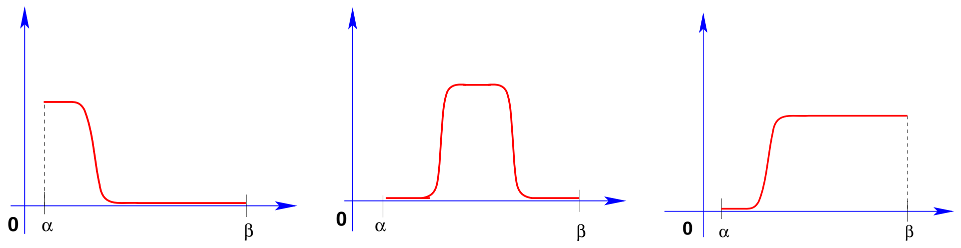

FIG. 1. *Low-pass (left), middle-pass (center), and high-pass (right) filter functions.*

in an application related to quantum mechanics [4], where large invariant subspaces associated with the lowest part of the spectrum are required. A rationale and some details on this approach are given in section 3. An emphasis is placed on invariant subspaces associated with middle eigenvalues, as this was not covered in [4].

The algorithms to be described in the next sections are based on polynomial filtering. Typically, a smooth "base filter" ϕ is selected, and then a sequence of least-squares polynomial approximations to this base function is constructed, from which a sequence of approximate solutions is extracted. One of the main goals of this paper is to express these approximations in a form which resembles the well-known conjugate gradient (CG) or conjugate residual (CR) algorithms. The algorithms to be described can be used to compute solutions of various problems in numerical linear algebra. As an example, by selecting the filter function ϕ to be the exponential function, the algorithms will yield a method for computing approximations to $\exp(A)b$.

2. Polynomial filtering.

Given a filter function ϕ, a symmetric matrix A, and a vector b, the problem addressed in this paper is to formulate algorithms for computing approximations of the form $p(A)b$ to the filtered vector $\phi(A)b$, where p is a polynomial. Specifically, we are interested in CG-type algorithms for finding "best" approximations to $\phi(A)b$.

This paper considers only three cases for the filter function ϕ: (a) A high-pass filter function, (b) a low-pass filter, and (c) a middle-pass filter. These three cases are illustrated in Figure 1. A high-pass (low-pass) filter function is one that is close to 1 for large (resp., small) eigenvalues and close to zero for small (resp., large) eigenvalues. A middle-pass filter is close to 1 for eigenvalues in a certain interval of the spectrum and close to zero elsewhere.

Consider a low-pass filter ϕ. We seek to approximate $\phi(A)b$ by vectors of the form $\rho(A)b$, where ρ is a polynomial. We will assume that $\phi(0) = 1$ so that ρ can be sought to also satisfy $\rho(0) = 1$. Thus the polynomial is selected to be in the form

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

In fact, ρ has the form of residual polynomials used in standard iterative methods such as the CG iteration. We will still often use the term "residual" polynomial for ρ, noting that there is not really a linear system to solve. We would like to minimize, for a certain norm $\|.\|_w$, the difference

$$\|\phi(A)b - \rho(A)b\|_w \tag{2}$$

over all polynomials ρ of degree $\leq k$, such as $\rho(0) = 1$.

The solution corresponding to the case of a high-pass filter can be trivially obtained from that of a low-pass filter. When ψ is a high-pass filter, then we can minimize $\|(1-\psi)-\rho(\lambda)\|_w$, where now $\phi \equiv 1-\psi$ is a low-pass filter. The best approximation to ψ is $1-\rho(\lambda)=\lambda s(\lambda)$, and therefore the vector $As(A)b$, where ρ minimizes (2), will be the desired solution in this case.

In summary, given a low-pass filter function ϕ, we seek s so that $\|\phi(\lambda)-(1-\lambda\ s(\lambda))\|_w$ is small—as measured by a certain norm w. The polynomial $\rho(\lambda)=1-\lambda s(\lambda)$ approximates the filter ϕ. This will also yield the minimum for $\|(1-\phi)-\lambda\ s(\lambda)\|_w$ with respect to $\lambda s(\lambda)$. The focus is now on the $\lambda s(\lambda)$, and the same process makes this polynomial close to the function

$$\psi \equiv 1-\phi \tag{3}$$

a high-pass filter, which we will refer to as the *dual filter to* ϕ. The case of the middle-pass filter, which will be addressed in detail in section 3.1, resembles the case of the high-pass filter since the filter function is such that $\phi(0)=0$.

There are many applications of filtering. Low-pass filters are of interest when computing invariant subspaces associated with all eigenvalues $\leq \alpha$. For example, the Lanczos algorithm can be used on the matrix $\rho(A)$, where ρ is a low-degree polynomial with the property that $\rho(\lambda)$ is small for $\lambda > \alpha$. This can be generalized to other situations, and thus in fact eigenspaces associated with eigenvalues located in arbitrary sub-intervals of the spectrum can be computed with the help of filtering. Filtering can also be used to solve highly ill-conditioned linear systems by regularization, but this is not considered in this paper.

An important application is in information retrieval, where one seeks to compute a matrix-vector product $A_k b$, where A_k is a low-rank approximation to A. Here A is not necessarily a square matrix, and we wish to find an approximation of Ab which is accurate in the dominant singular space. This is the situation in LSI, apart from the fact that it is the transpose of A that is considered instead of A. If $A=U\Sigma V^T$ is the SVD of A, then calculating a solution of the form $A\phi(A^TA)b$, we find that

$$\begin{aligned} A\phi(A^TA)b &= (U\Sigma V^T)V\phi(\Sigma^T\Sigma)V^Tb \\ &= U\Sigma\phi(\Sigma^T\Sigma)V^Tb. \end{aligned}$$

The requirement is that $\Sigma\phi(\Sigma^T\Sigma)$ be close to Σ for large σ_i's and close to zero for small σ_i's. So this situation can be handled by a low-pass filter. In [19], a method of this type was used by exploiting expansions of the desired polynomial in a basis of orthogonal polynomials.

2.1. Polynomial filters. In this section, we focus on the problem of filtered iterations. We begin with some notation as well as a rationale for the approach to be taken. We consider a CG-like (actually CR-like) method which uses an arbitrary inner product of functions. The main reason why we seek to write the solution algorithm by exploiting the CR/CG framework is that we already know some of the good algorithmic properties of these methods. In particular, the solution and residual vectors are available at each step, and the solution vector at step k is easily updated from the solution vector at step $k-1$. The numerical properties of the algorithms are also well understood, both in practice and in theory.

Recall that the approximate solution vector obtained at the jth step of a Krylov subspace method is of the form $x_0+s_j(A)r_0$, where s_j is a polynomial of degree $\leq j$. The corresponding residual vector is $\rho_{j+1}(\lambda)=1-\lambda s_j(\lambda)$. This polynomial is of degree $j+1$. It has value 1 at $\lambda=0$, and it approximates a low-pass filter function ψ.

2.2. CR algorithms in polynomial spaces. In the standard CR algorithm, the solution polynomial s_j minimizes $\|(I - As(A))r_0\|_2$ over all polynomials s of degree $\leq j$. This is nothing but a discrete least-squares norm when expressed in the eigenbasis. Indeed, if the eigenexpansion of r_0 is $r_0 \equiv \sum_{i=1}^{n} \omega_i u_i$, then

$$\|(I - As(A))r_0\|_2 = \left[\sum_{i=1}^{n} \omega_i^2 (1 - \lambda_i s(\lambda_i))^2\right]^{1/2} \equiv \|1 - \lambda s(\lambda)\|_w .$$

It is possible to write a CR-like algorithm which minimizes $\|1 - \lambda s(\lambda)\|_w$ for any 2-norm associated with a (proper) inner product over polynomial spaces:

$$\langle p, q \rangle_w \ .$$

The generic algorithm is given below for reference.

Algorithm 2.1. *Formal conjugate residual algorithm.*

```
0. Compute $r_0 := b - Ax_0$, $p_0 := r_0$        $\pi_0 = \rho_0 = 1$
1.                                                Compute $\lambda\pi_0$
2. For $j = 0, 1, \ldots,$ until convergence Do:
3.     $\alpha_j := \langle \rho_j, \lambda\rho_j \rangle_w / \langle \lambda\pi_j, \lambda\pi_j \rangle_w$
4.     $x_{j+1} := x_j + \alpha_j p_j$
5.     $r_{j+1} := r_j - \alpha_j A p_j$              $\rho_{j+1} = \rho_j - \alpha_j \lambda\pi_j$
6.     $\beta_j := \langle \rho_{j+1}, \lambda\rho_{j+1} \rangle_w / \langle \rho_j, \lambda\rho_j \rangle_w$
7.     $p_{j+1} := r_{j+1} + \beta_j p_j$             $\pi_{j+1} := \rho_{j+1} + \beta_j \pi_j$
8.                                                Compute $\lambda\pi_{j+1}$
9. EndDo
```

It is easy to show that the residual polynomial ρ_j generated by this algorithm minimizes $\|\rho(\lambda)\|_w$ among all polynomials of the form $\rho(\lambda) = 1 - \lambda s(\lambda)$, where s is any polynomial of degree $\leq j - 1$. In other words, ρ_j minimizes $\|\rho(\lambda)\|_w$ among all polynomials ρ of degree $\leq j$ such that $\rho(0) = 1$. It is also easy to show that the polynomials $\lambda\pi_j$ are orthogonal to each other; i.e., $\langle \pi_i, \pi_j \rangle = 0$ for $i \neq j$.

Proposition 2.1. *The solution vector x_{j+1} computed at the jth step of Algorithm* 2.1 *is of the form $x_{j+1} = x_0 + s_j(A)r_0$, where s_j is the jth degree polynomial*

$$s_j(\lambda) = \alpha_0 \pi_0(\lambda) + \cdots + \alpha_j \pi_j(\lambda). \tag{4}$$

The polynomials π_j and the residual polynomials $\rho_{j+1}(\lambda)$ satisfy the following orthogonality relations:

$$\langle \lambda\pi_j(\lambda), \lambda\pi_i(\lambda) \rangle_w = \langle \lambda\rho_j(\lambda), \rho_i(\lambda) \rangle_w = 0 \quad \textit{for} \quad i \neq j. \tag{5}$$

In addition, the residual polynomial $\rho_{j+1} = 1 - \lambda s_j(\lambda)$ minimizes $\|1 - \lambda s(\lambda)\|_w$ among all polynomials s of degree $\leq j$.

A formal proof is not necessary, but one can exploit the analogy with the usual CR algorithm. In CR (see, e.g., [24]), it is known that the vectors Ap_j are orthogonal to each other. Writing a member of the affine Krylov subspace $x_0 + K_j$ as $x = x_0 + s(A)r_0$, where the degree of s is $\leq j$, the vectors r_{j+1} minimize the 2-norm of all residuals $b - Ax = r_0 - As(A)r_0$ for x in $x_0 + K_j$.

It is useful to comment on implementation aspects. In the usual CR algorithm (see [24]) we would compute Ap_{j+1} in line 8 using the relation which follows from line 7,

$$Ap_{j+1} = Ar_{j+1} + \beta_j A p_j ,$$

in order to avoid an additional matrix-vector product. The vector Ar_{j+1} is computed after line 5 (and saved for the next step to get α_{j+1}), and Ap_{j+1} is then obtained from it using the above formula. Generally, this needs to be done in the situation when the computation of the scalar α_j in line 3 requires the vector Ap_j as well as the vector Ar_j. In the very first step, p and r are the same, so computing Ap_0 in line 1 will suffice. Thereafter, it is necessary to compute Ar_j (before line 3) and update Ap_{j+1}, as was just explained. This strategy is not necessary here because the updates and computations of polynomials require relatively few operations.

We would like to modify the algorithm shown above in order to incorporate filtering. As it is written the algorithm does not lend itself to filtering. Indeed, filtering amounts to minimizing some norm of $\phi(\lambda)-(1-\lambda s(\lambda))$, where ϕ is the filter function, and one must remember that $\phi(A)v$ may be practically difficult to evaluate for a given vector v. In particular, $\phi(A)r_0$ may not be available.

We omit the discussion of CG-type iterations—but it is clear that a CG algorithm in polynomial space can also be written. The residual polynomials will be orthogonal, while the π_js will be conjugate ($\langle\lambda\pi_j,\pi_i\rangle_w = 0$ for $i \neq j$).

2.3. Filtered CR polynomial iterations. Given a certain filter function ϕ, the method to be described in this section consists of finding an approximate solution x_j whose residual polynomial $\rho_j(\lambda)$ approximates the function ϕ, in the least-squares sense. Throughout this section, we consider the dual viewpoint, which is that we are given a *high-pass* filter ψ which is close to zero for λ near zero and close to 1 for large eigenvalues. To make the computation tractable, the function ψ will be chosen to be a piecewise continuous function, though this is not an essential requirement. This will be discussed in more detail in section 2.5. In mathematical terms, we seek a polynomial $s_j(\lambda)$ such that

$$\|\psi(\lambda)-\lambda s_j(\lambda)\|_w = \min_{s\,\in\,\mathcal{P}_j}\|\psi(\lambda)-\lambda s(\lambda)\|_w. \tag{6}$$

Here $\mathcal{P}_j$ represents the space of polynomials of degree $\leq j$, and the w-norm is associated with an inner product of the form

$$\langle p,q\rangle_w = \int_0^\beta p(\lambda)q(\lambda)w(\lambda)d\lambda.$$

Note that the left bound of the interval is taken to be zero without loss of generality. For the sake of clarity, the discussion of the choice of the weight function is deferred to a later section. For now, all that needs to be said is that w is selected primarily to enable an easy computation of an inner product of any two functions involved in the algorithms, without resorting to numerical integration.

The condition for the polynomial s_j to be the solution to (6) is that

$$\langle\psi(\lambda)-\lambda s_j(\lambda),\lambda q(\lambda)\rangle_w = 0 \quad \forall q\in\mathcal{P}_j.$$

In order to construct the sequence of approximate solutions, we can generate the sequence of polynomials of the form $\lambda\pi_j$ which are orthogonal. The sequence satisfies a three-term recurrence, and the approximation can be directly expressed in this basis. This was the approach taken in [11, 19].

As a slight alternative, we can try to proceed as in the CR algorithm by updating s_j from s_{j-1} as

$$s_j(\lambda) = s_{j-1}(\lambda)+\alpha_j\pi_j(\lambda). \tag{7}$$

The scalar α_j can be obtained by expressing the condition that $\psi(\lambda) - \lambda s_j(\lambda)$ is orthogonal to $\lambda\pi_j(\lambda)$, or $\langle \psi(\lambda) - \lambda s_j(\lambda), \lambda\pi_j(\lambda)\rangle_w = 0$, which, with the use of (7), leads to

$$\alpha_j = \frac{\langle \psi(\lambda) - \lambda s_{j-1}(\lambda), \lambda\pi_j(\lambda)\rangle_w}{\langle \lambda\pi_j(\lambda), \lambda\pi_j(\lambda)\rangle_w} \ . \tag{8}$$

The orthogonality of the set $\{\lambda\pi_i\}$ can be exploited to observe that $\lambda s_{j-1}(\lambda)$ is orthogonal to $\lambda\pi_j$. In the end the above expression simplifies to

$$\alpha_j = \frac{\langle \psi(\lambda), \lambda\pi_j(\lambda)\rangle_w}{\langle \lambda\pi_j(\lambda), \lambda\pi_j(\lambda)\rangle_w} \ . \tag{9}$$

This is a different expression from that obtained from the usual CR algorithm. However, it is possible to express it differently, and this will be explored later for a different algorithm.

After α_j is computed in this manner, we proceed to update the solution x_j and the residual vector r_{j+1} as in steps 4 and 5 of Algorithm 2.1. The polynomial ρ_{j+1} is also updated accordingly. Next, we must compute π_{j+1}. In the usual CG and CR algorithms, π_{j+1} is computed in the form $\pi_{j+1}(\lambda) = \rho_{j+1}(\lambda) + \beta_j\pi_j(\lambda)$, but this will not work here because such an expression exploits the orthogonality of ρ_{j+1} against all $\lambda\pi_i$'s with $i \le j$, which is no longer satisfied. Instead, we could just use a Stieljes-type procedure of the form

$$\beta_{j+1}\pi_{j+1}(\lambda) = \lambda\pi_j(\lambda) - \eta_j\pi_j(\lambda) - \beta_j\pi_j(\lambda).$$

Note that $-\alpha_j\lambda\pi_j(\lambda) = \rho_{j+1}(\lambda) - \rho_j(\lambda)$, and so, if we need the leading coefficients of π_{j+1} and ρ_{j+1} to be the same, we can use the formula

$$\pi_{j+1}(\lambda) = -\alpha_j\left[\lambda\pi_j(\lambda) - \eta_j\pi_j(\lambda) - \beta_j\pi_{j-1}(\lambda)\right] \tag{10}$$

and select the scalars η_j and β_j to make $\lambda\pi_{j+1}$ orthogonal to both $\lambda\pi_j$ and $\lambda\pi_{j-1}$. Assume by induction that the $\lambda\pi_i(\lambda)$'s are orthogonal for $i \le j$. Then, we find that

$$\eta_j = \frac{\langle \lambda^2\pi_j, \lambda\pi_j\rangle_w}{\langle \lambda\pi_j, \lambda\pi_j\rangle_w} \quad \text{and} \quad \beta_j = \frac{\langle \lambda^2\pi_j, \lambda\pi_{j-1}\rangle_w}{\langle \lambda\pi_{j-1}, \lambda\pi_{j-1}\rangle_w} \ .$$

ALGORITHM 2.2. *Minimal pseudoresidual algorithm.*

```
0. Compute $r_0 := b - Ax_0$, $p_0 := r_0$          $\pi_0 = \rho_0 = 1$
1.                                               Compute $\lambda\pi_0$, $\lambda^2\pi_0$
2. For $j = 0, 1, \ldots,$ until convergence Do:
3.     $\alpha_j := \frac{\langle\psi,\lambda\pi_j\rangle_w}{\langle\lambda\pi_j,\lambda\pi_j\rangle_w}$
4.     $x_{j+1} := x_j + \alpha_j p_j$
5.     $r_{j+1} := r_j - \alpha_j A p_j$                 $\rho_{j+1} = \rho_j - \alpha_j\lambda\pi_j$
6.     $\eta_j := \frac{\langle\lambda^2\pi_j,\lambda\pi_j\rangle_w}{\langle\lambda\pi_j,\lambda\pi_j\rangle_w}$   $\beta_j := \frac{\langle\lambda^2\pi_j,\lambda\pi_{j-1}\rangle_w}{\langle\lambda\pi_{j-1},\lambda\pi_{j-1}\rangle_w}$
7.     $p_{j+1} := -\alpha_j[Ap_j - \eta_j p_j - \beta_j p_{j-1}]$  $\pi_{j+1} := -\alpha_j\left[\lambda\pi_j - \eta_j\pi_j - \beta_j\pi_{j-1}\right]$
8.                                               Compute $\lambda\pi_{j+1}, \lambda^2\pi_{j+1}$
9. EndDo
```

This approach is a slight variation of the one presented in [11, 19]. The main difference is that the algorithms in [11, 19] focus on the solution polynomial instead of the residual polynomial; i.e., they do not explicitly compute or exploit residual polynomials. However, the two algorithms are mathematically equivalent. Note that when $\psi(\lambda) \equiv 1$, the algorithm should give the same iterates (and same auxiliary vectors) as those of Algorithm 2.1 in exact arithmetic.

The polynomials $\lambda\pi_j$ are orthogonal by construction. On the other hand, the residual polynomials ρ_j do not satisfy any orthogonality relation, but optimality implies that $\langle \psi - \lambda s_j(\lambda), \lambda\pi_i\rangle_w = 0$ for $i \le j$, so we have (recall that $\phi \equiv 1 - \psi$)

$$\langle \phi - \rho_{j+1}, \lambda\pi_i\rangle_w \;=\; 0, \qquad i \le j.$$

2.4. Corrected CR algorithm. We now consider an alternative implementation of the above algorithm, which can be viewed as a corrected version of the standard CR algorithm. The derivation is based on the following observation. After line 5 of Algorithm 2.2, the residual vector r_{j+1} is no longer used. This particular residual vector is not all that useful since a convergence test cannot employ it. It would have been more meaningful to compute $[\psi(A) - As(A)]b$, but this is not practically computable. Therefore, instead of r_j we can generate another residual polynomial which will help obtain the p_i's: *the one that would be obtained from the actual CR algorithm*, i.e., the same r vectors as those of Algorithm 2.1. It is interesting to note that with this sequence of residual vectors, which will be denoted by $\tilde{r}_j$, it is easy to generate the directions p_i, *which are the same* for both algorithms. So the idea is straightforward: obtain the auxiliary residual polynomials $\tilde{\rho}_j$ that are those associated with the *standard* CR algorithm and exploit them to obtain the π_i's in the same way as in the CR algorithm. The polynomials $\lambda\pi_j$ are orthogonal, and therefore the expression of the desired approximation is the same. The algorithm is described next where now $\tilde{\rho}_j$ is the polynomial associated with the auxiliary sequence $\tilde{r}_j$.

ALGORITHM 2.3. *Filtered conjugate residual polynomials algorithm.*

```
0.  Compute $\tilde{r}_0 := b - Ax_0$, $p_0 := \tilde{r}_0$          $\pi_0 = \tilde{\rho}_0 = 1$
1.                                              Compute $\lambda\pi_0$
2.  For $j = 0, 1, \ldots,$ until convergence Do:
3.      $\tilde{\alpha}_j := \langle \tilde{\rho}_j, \lambda\tilde{\rho}_j\rangle_w / \langle \lambda\pi_j, \lambda\pi_j\rangle_w$
4.      $\alpha_j := \langle \psi, \lambda\pi_j\rangle_w / \langle \lambda\pi_j, \lambda\pi_j\rangle_w$
5.      $x_{j+1} := x_j + \alpha_j p_j$
6.      $\tilde{r}_{j+1} := \tilde{r}_j - \tilde{\alpha}_j A p_j$                 $\tilde{\rho}_{j+1} = \tilde{\rho}_j - \tilde{\alpha}_j \lambda\pi_j$
7.      $\beta_j := \langle \tilde{\rho}_{j+1}, \lambda\tilde{\rho}_{j+1}\rangle_w / \langle \tilde{\rho}_j, \lambda\tilde{\rho}_j\rangle_w$
8.      $p_{j+1} := \tilde{r}_{j+1} + \beta_j p_j$                 $\pi_{j+1} := \tilde{\rho}_{j+1} + \beta_j \pi_j$
9.                                              Compute $\lambda\pi_{j+1}$
10. EndDo
```

It is remarkable that the only difference between this and generic CR-type algorithm (see, e.g., Algorithm 2.1) is that the updates to x_{j+1} use a coefficient α_j different from that of the update to the vectors $\tilde{r}_{j+1}$. Observe that the residual vectors $\tilde{r}_j$ obtained by the algorithm are just auxiliary vectors that do not correspond to the original residuals $r_j = b - Ax_j$. Needless to say, these residuals, the r_j's, can also be generated after line 5 (or 6) from $r_{j+1} = r_j - \alpha_j A p_j$. Depending on the application, it may be necessary to include these computations.

PROPOSITION 2.2. *The solution vector x_{j+1} computed at the jth step of Algorithm 2.3 is of the form $x_{j+1} = x_0 + s_j(A) r_0$, where s_j is the jth degree polynomial:*

$$s_j(\lambda) = \alpha_0\pi_0(\lambda) + \cdots + \alpha_j\pi_j(\lambda). \tag{11}$$

The polynomials π_j and the auxiliary polynomials $\tilde{\rho}_j(\lambda)$ satisfy the orthogonality re-

lations,

$$\langle \lambda\pi_j(\lambda), \lambda\pi_i(\lambda)\rangle_w = \langle \lambda\tilde{\rho}_j(\lambda), \tilde{\rho}_i(\lambda)\rangle_w = 0 \quad \textit{for} \quad i \neq j. \tag{12}$$

In addition, the filtered residual polynomial $\psi - \lambda s_j(\lambda)$ *minimizes* $\|\psi - \lambda s(\lambda)\|_w$ *among all polynomials* s *of degree* $\le j-1$.

Proof. The first observation is that the polynomials $\tilde{\rho}_j$ and π_j are identical with the polynomials ρ_j and π_j of Algorithm 2.1, so the orthogonality property (12) is trivially satisfied. The relation (4) uses scalars α_j that are different from those denoted by $\tilde{\alpha}_j$ of the sequence $\tilde{\rho}_j$. From this relation, we have that $\psi - \lambda s_j(\lambda) = \psi - \sum_{i=0}^{j} \alpha_i \lambda\pi_i(\lambda)$. By the optimality condition, the best polynomial is obtained when the scalars α_i satisfy the relation $\langle \psi - \lambda s_j(\lambda), \lambda\pi_i(\lambda)\rangle_w = 0$, for $i = 1, \dots, j$. Exploiting (11) and the orthogonality of the system $\{\lambda\pi_i\}_{i=0,\dots,j}$, this yields

$$\alpha_j = \langle \psi, \lambda\pi_j\rangle_w \ / \ \langle \lambda\pi_j, \lambda\pi_j\rangle_w. \qquad \square$$

It is worth exploring the formula (9), which defines the scalars α_j, a little further. In the standard CR algorithm, the expression (8) is modified by exploiting orthogonality relations to lead to the standard expression of line 3 of Algorithm 2.1. However, this is no longer possible here, essentially because the polynomial s_{j-1} in (9) uses the scalar α_i's (formula (11)), and there are no orthogonality relations satisfied with the corresponding residual polynomials ρ_j. It is, however, possible to express the scalar α_j as a modification to the scalar $\tilde{\alpha}_j$. Indeed, define $\tilde{s}_j \equiv \sum_{i=0}^{j} \tilde{\alpha}_i \pi_i$, which is the solution polynomial of Algorithm 2.1, and observe that $\langle \lambda\tilde{s}_{j-1}, \lambda\pi_j\rangle_w = 0$, because $\lambda\pi_j$ is orthogonal to all polynomials λq_i for polynomials q_i of degree $i \le j-1$. Then, we can rewrite the numerator of (9) as

$$\langle \psi, \lambda\pi_j\rangle_w = \langle \psi - \lambda\tilde{s}_{j-1}, \lambda\pi_j\rangle_w = \langle (\psi-1) + 1 - \lambda\tilde{s}_{j-1}, \lambda\pi_j\rangle_w = \langle \tilde{\rho}_j, \lambda\pi_j\rangle_w - \langle 1-\psi, \lambda\pi_j\rangle_w \,.$$

Since $\tilde{\rho}_j$ and π_j have the same leading coefficient, by exploiting orthogonality we readily obtain the relation $\langle \tilde{\rho}_j, \lambda\pi_j\rangle_w = \langle \tilde{\rho}_j, \lambda\tilde{\rho}_j\rangle_w$, which yields the following alternative formula for α_j:

$$\alpha_j = \tilde{\alpha}_j - \frac{\langle 1-\psi, \lambda\pi_j\rangle_w}{\langle \lambda\pi_j, \lambda\pi_j\rangle_w} \,. \tag{13}$$

The only merit of this expression, as a substitute for (9), is that it clearly establishes Algorithm 2.3 as a "corrected version" of the standard Algorithm 2.1. In the special situation when $\psi \equiv 1$, $\alpha_i = \tilde{\alpha}_i$, and the two algorithms coincide as expected.

2.5. The base filter function. The solutions computed by the algorithms just seen are based on generating polynomial approximations to a certain base filter function ϕ. In the following, we will consider a low-pass filter ϕ. As was already mentioned, it is generally not a good idea to use as ϕ the step function

$$\phi(t) = \begin{cases} 1, & t < \tau_0, \\ 0, & t \ge \tau_0. \end{cases}$$

This is because this function is discontinuous, and approximations to it by high-degree polynomials will exhibit very wide oscillations, known as Gibbs oscillations. It is preferable to take as a "base" filter, i.e., the filter which is ultimately approximated by polynomials, a smooth function such as the one on the left side of Figure 1.

The base filter function can be a piecewise polynomial consisting of two parts: a function which decreases smoothly from 1 to 0 when λ increases from 0 to τ_0, and the constant function zero in the interval $[\tau_0, \beta]$. Alternatively, the function can consist of three parts, one on each of the intervals $[0,\ \tau_0]$, $[\tau_0,\ \tau_1]$, and $[\tau_1,\ \beta]$, with $0 < \tau_0 < \tau_1 < \beta$. It will begin with the constant value 1 in the interval $[0,\ \tau_0]$, then decrease smoothly from 1 to 0 in the second interval $[\tau_0\ \tau_1]$, and finally take the constant value 0 in $[\tau_1,\ \beta]$. The second part of the function (the first part for the first scenario) bridges the values 0 and 1 by a smooth function and was termed a "bridge function" in [11]. In what follows we focus on obtaining bridge functions for the generic case, i.e., for an interval $[\tau_0,\ \tau_1]$.

A systematic way of generating base filter functions is to use bridge functions obtained from Hermite interpolation. The bridge function is an interpolating polynomial (in the Hermite sense) depending on two integer parameters m_0, m_1, and denoted by $\Theta_{[m_0,m_1]}$, which satisfies the following conditions:

$$\begin{array}{llll} \Theta_{[m_0,m_1]}(\tau_0) = 1, & \Theta'_{[m_0,m_1]}(\tau_0) = \cdots = & \Theta^{(m_0)}_{[m_0,m_1]}(\tau_0) = 0, \\ \Theta_{[m_0,m_1]}(\tau_1) = 0, & \Theta'_{[m_0,m_1]}(\tau_1) = \cdots = & \Theta^{(m_1)}_{[m_0,m_1]}(\tau_1) = 0. \end{array} \tag{14}$$

Thus, $\Theta_{[m_0,m_1]}$ has degree $m_0 + m_1 + 1$, m_0, and m_1 define the degree of smoothness at the points τ_0 and τ_1, respectively.

Such polynomials can be easily determined by the usual finite difference tables in the Hermite sense. To find a closed form for the polynomials $\Theta_{[m_0,m_1]}$ it is useful to change variables in order to exploit symmetry. We map the variable onto the interval $[-1, 1]$ and shift the function down by 1/2. If the corresponding function is denoted by η, then the above conditions become

$$\begin{array}{lclllcll} \eta(-1) & = & 1/2, & & \eta(+1) & = & -1/2, & \\ \eta^{(i)}(-1) & = & 0 & \text{for } i = 1, \ldots, m_0, & \eta^{(i)}(+1) & = & 0 & \text{for } i = 1, \ldots, m_1\ . \end{array}$$

The derivative function η' can be expressed as $\eta'(t) = c\ (1-t)^{m_1}(1+t)^{m_0}$, and as a result we have a closed form expression of $\eta(t)$:

$$\eta(t) = \frac{1}{2}\ -\ \frac{\int_{-1}^{t}\ (1-s)^{m_1}(1+s)^{m_0}\ ds}{\int_{-1}^{1}\ (1-s)^{m_1}(1+s)^{m_0}\ ds}. \tag{15}$$

The first and second derivatives of η are

$$\eta'(t) = -\frac{(1-t)^{m_1}(1+t)^{m_0}}{\int_{-1}^{1}\ (1-s)^{m_1}(1+s)^{m_0}\ ds}\ , \qquad \eta''(t) = \left[\frac{m_1}{1-t} - \frac{m_0}{1+t}\right]\eta'(t). \tag{16}$$

Thus there is an inflexion point at

$$t = \frac{m_0 - m_1}{m_0 + m_1}.$$

Since the maximum absolute value of the derivative is required for the convergence analysis, it will be useful to determine it. The derivative is negative and decreases from its value at the point -1 to a certain minimum, reached at the inflexion point, and then it increases from there to its final value at the point 1. The peak value and an approximation to it are given by the following lemma.

LEMMA 2.3. *The maximum absolute value of the derivative of the function η in the interval $[-1, 1]$ is given by*

$$\eta'_{max} = \frac{m_0 + m_1 + 1}{2} \; \frac{m_1^{m_1} \; m_0^{m_0}}{(m_0 + m_1)^{m_0+m_1} \times \binom{m_0}{m_0+m_1}} \; . \tag{17}$$

For large values of m_0 and m_1, the maximum derivative is approximately

$$\eta'_{max} \approx \frac{m_0 + m_1}{2\sqrt{2\pi}} \sqrt{\frac{1}{m_0} + \frac{1}{m_1}} \; . \tag{18}$$

Proof. The integral in the denominator of η in (16) can be computed by successive integration by parts to be

$$\int_{-1}^{1} (1-s)^{m_1}(1+s)^{m_0} \; ds = \frac{m_1! m_0!}{(m_0+m_1)!} \times \frac{2^{m_0+m_1+1}}{m_0+m_1+1} \; .$$

Evaluating the negative derivative $-\eta'$ at the inflexion point yields

$$\eta'_{max} = \frac{\frac{(2m_1)^{m_1} \; (2m_0)^{m_0}}{(m_0+m_1)^{m_0+m_1}}}{\frac{m_1! m_0!}{(m_0+m_1)!} \times \frac{2^{m_0+m_1+1}}{m_0+m_1+1}} = \frac{m_0 + m_1 + 1}{2} \; \frac{m_1^{m_1} \; m_0^{m_0}}{(m_0 + m_1)^{m_0+m_1} \times \binom{m_0}{m_0+m_1}},$$

which is the first result. This can be rewritten as

$$\eta'_{max} \;=\; \frac{m_0 + m_1 + 1}{2} \; \frac{\frac{m_0^{m_0}}{m_0!} \times \frac{m_1^{m_1}}{m_1!}}{\frac{(m_0+m_1)^{m_0+m_1}}{(m_0+m_1)!}} \; .$$

Using Sterling's formula $m! \approx \sqrt{2\pi m} \; (m/e)^m$ yields (18), after simplifications. ∎

This result must now be translated into the original interval $[\tau_0, \; \tau_1]$. The function Θ (indices m_0, m_1 are omitted) and its derivative in terms of η and η' are

$$\Theta(\lambda) = \frac{1}{2} + \eta\left(2\frac{\lambda - \tau_0}{\tau_1 - \tau_0} - 1\right), \qquad \Theta'(\lambda) = \frac{2}{\tau_1 - \tau_0}\eta'\left(2\frac{\lambda - \tau_0}{\tau_1 - \tau_0} - 1\right),$$

and so

$$\Theta'_{max} = \frac{2}{\tau_1 - \tau_0}\eta'_{max} \; .$$

As an example of a bridge function, the case when $m_0 = m_1 = 2$ yields

$$\eta(t) = \frac{-15}{16} \times \left(t - 2\frac{t^3}{3} + \frac{t^5}{5}\right),$$

which, for the interval $[0, \alpha]$, translates into the function

$$\Theta_{[2,2]}(t) = \frac{1}{2} - \frac{15}{16} \; \left(2\frac{t}{\alpha} - 1\right) + \frac{5}{8} \; \left(2\frac{t}{\alpha} - 1\right)^3 - \frac{3}{16}\left(2\frac{t}{\alpha} - 1\right)^5.$$

Similarly, for $m_0 = m_1 = 3$ we find

$$\Theta_{[3,3]}(t) = \frac{1}{2} - \frac{35}{32}\left(2\frac{t}{\alpha} - 1\right) + \frac{35}{32}\left(2\frac{t}{\alpha} - 1\right)^3 - \frac{21}{32}\left(2\frac{t}{\alpha} - 1\right)^5 + \frac{5}{32}\left(2\frac{t}{\alpha} - 1\right)^7.$$

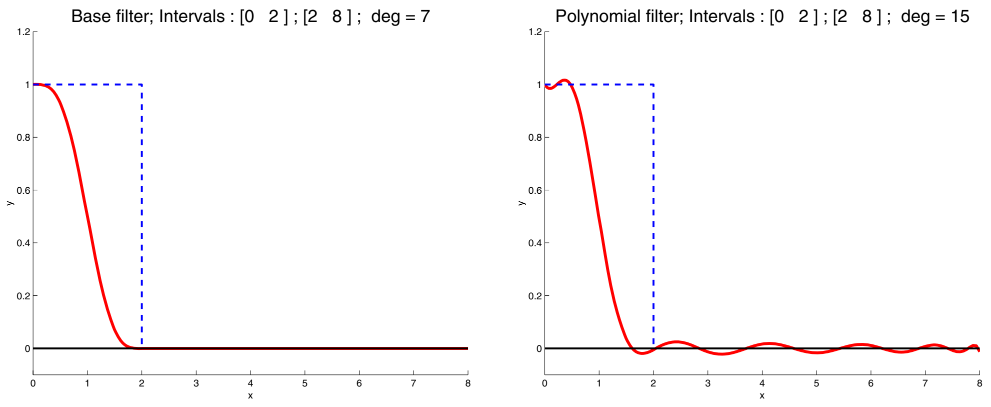

FIG. 2. *Left: Base filter ϕ defined on two intervals: $\Theta_{[4,4]}$ in $[0, 2]$ and zero in $[2, 8]$; Right: its polynomial approximation of degree* 15.

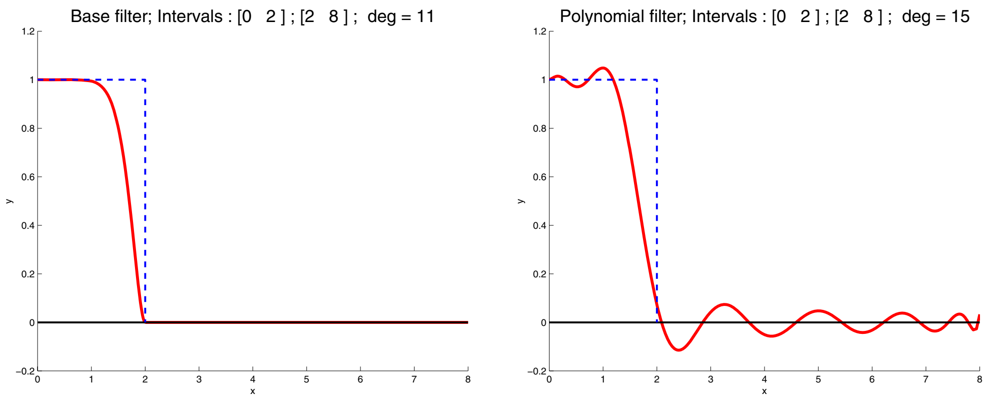

FIG. 3. *Left: Base filter ϕ defined on two intervals: $\Theta_{[10,2]}$ in $[0, 2]$ and zero in $[2, 8]$. Right: Its polynomial approximation of degree* 15.

As was seen, the ratio $\frac{m_1}{m_0}$ determines the localization of the inflexion point. The polynomial can be made to decrease rapidly from one to zero in a small interval by taking high-degree polynomials, but this has the effect of slowing down convergence toward the desired filter, as it tends to cause undesired oscillations.

Two examples of filter functions are shown in Figures 2 and 3. A third example, shown in Figure 4, shows a situation where three intervals are used. In the first interval [0, 1.7] and third interval [2.3, 8], the filter takes the constant values 1 and 0, respectively. In the middle interval [1.7, 2.3], ϕ is defined by the Hermite polynomial $\Theta_{[5,5]}$ in [1.72.3]. This time we plot a higher-degree polynomial approximation to ϕ to show the quality of the resulting polynomial. For higher-degree polynomials (say 80) there is no visible difference between the base filter ϕ and its polynomial approximation. We also computed many other polynomials using Legendre weights in each interval instead of Chebyshev weights and, in all cases, saw no significant difference.

2.6. The weight function w. Denoting the l subintervals of $[0,\ \beta]$ by $[\tau_{i-1}, \tau_i]$, $i = 1, \ldots, l$, we define the inner product on each subinterval (τ_{l-1}, τ_l), using Chebyshev

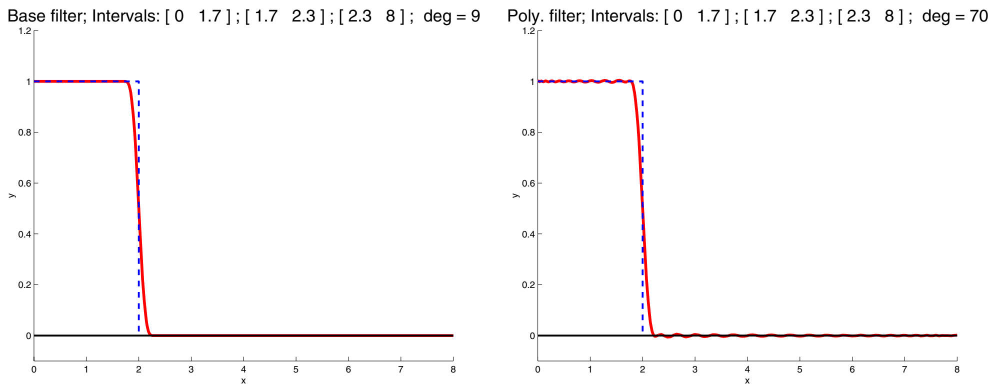

FIG. 4. *Left: Dual base filter* ϕ *defined on three intervals:* 1 *in* $[0, 1.7]$, $\Theta_{[5,5]}$ *in* $[1.7, 2.3]$, *and* 0 *in* $[2.3, 8]$. *Right: Its polynomial approximation of degree* 70.

weights:

$$\langle \psi_1, \psi_2 \rangle_{\tau_{l-1},\tau_l} = \int_{\tau_{l-1}}^{\tau_l} \frac{\psi_1(t)\psi_2(t)}{\sqrt{(t-\tau_{l-1})(\tau_l - t)}}\, dt.$$

Then the inner product on the interval $[0,\ \beta] \equiv [\tau_0,\ \tau_1] \cup [\tau_1,\ \tau_2] \cdots \cup [\tau_{l-1},\ \tau_l]$ is defined as a weighted sum of the inner products on the smaller intervals,

$$\langle \psi_1, \psi_2 \rangle_w = \sum_{i=1}^{l} \mu_i \int_{\tau_{i-1}}^{\tau_i} \frac{\psi_1(t)\psi_2(t)}{\sqrt{(t-\tau_{i-1})(\tau_i - t)}}\, dt. \tag{19}$$

For example, for two intervals the weight function is defined as

$$\langle \psi_1, \psi_2 \rangle_w = \mu_1 \int_0^{\alpha} \frac{\psi_1(t)\psi_2(t)}{\sqrt{t(\alpha - t)}}\, dt + \mu_2 \int_a^{\beta} \frac{\psi_1(t)\psi_2(t)}{\sqrt{(t-\alpha)(\beta - t)}}\, dt. \tag{20}$$

The μ_i's can be chosen to emphasize or deemphasize specific subintervals. In most of our tests we took the μ_i to be either equal to the constant 1 or to the inverse of the width of each subinterval. Note that we can also use Legendre polynomials, or indeed any other orthogonal polynomials, instead of Chebyshev polynomials. We found very little difference in performance (convergence) between Legendre and Chebyshev polynomials.

The issue of obtaining orthogonal polynomials from sequences of orthogonal polynomials on other intervals was addressed in [12] and [22]. One of the main problems is to avoid numerical integration. In [22] this was achieved by expanding the desired functions in a basis of Chebyshev polynomials on each of the subintervals. Note that the expansions are redundant—but cost is not a major issue. Let $\varsigma^{(l)}$ be the mapping which transforms the interval $[\tau_{l-1}, \tau_l]$ into $[-1, 1]$:

$$\varsigma^{(l)}(\lambda) = \frac{2}{\tau_l - \tau_{l-1}}\lambda - \frac{\tau_l + \tau_{l-1}}{\tau_l - \tau_{l-1}}\ .$$

Denote by C_i the ith degree Chebyshev polynomial of the first kind on $[-1, 1]$, and define

$$C_i^{(l)}(\lambda) = C_i\left(\varsigma^{(l)}(\lambda)\right), \quad i \geq 0.$$

When all polynomials are expanded in the above Chebyshev bases on each interval, then all operations involved in Algorithms 2.1, 2.2, and 2.3 are easily performed with the expansion coefficients. Thus, adding and scaling two expanded polynomials is a trivial operation. Consider now inner products of two polynomials. Recall that on each interval the scaled and shifted Chebyshev polynomials $(C_k^{(l)})_{k\in\mathbb{N}}$ constitute an orthogonal basis since

$$\langle C_i^{(l)}, C_j^{(l)}\rangle_{\tau_{l-1},\tau_l} = \begin{cases} 0 & \text{if } i \neq j, \\ \pi & \text{if } i = j = 0, \\ \frac{\pi}{2} & \text{if } i = j \neq 0. \end{cases}$$

As a result, if two polynomials ψ_1, ψ_2 are expanded in the above Chebyshev bases for each interval, the inner products (19) of these polynomials are trivially obtained from their expansion coefficients in the bases.

The only remaining operation to consider is that of multiplying a polynomial by λ (e.g., line 9 of Algorithm 2.3). A polynomial ψ expanded in the Chebyshev bases can easily be multiplied by the variable λ, by exploiting the following relations:

$$\lambda\, C_i^{(l)}(\lambda) = \frac{\tau_l - \tau_{l-1}}{4} C_{i+1}^{(l)}(\lambda) + \frac{\tau_l + \tau_{l-1}}{2} C_i^{(l)}(\lambda) + \frac{\tau_l - \tau_{l-1}}{4} C_{i-1}^{(l)}(\lambda), \quad i \geq 1,$$
$$\lambda\, C_0^{(l)}(\lambda) = \frac{\tau_l - \tau_{l-1}}{2} C_1^{(l)}(\lambda) + \frac{\tau_l + \tau_{l-1}}{2} C_0^{(l)}(\lambda).$$

These formulations come from the recurrences obeyed by Chebyshev polynomials: $2tC_i(t) = C_{i+1}(t) + C_{i-1}(t)$ for $i > 0$, and $tC_0(t) = C_1(t)$.

2.7. Convergence. It is desirable to know how fast the polynomial ρ_j converges to the low-pass filter function ϕ. Convergence results of this type utilize uniform norm results. We will restrict ourselves to a simple result derived from the Jackson theorems; see [7]. A common notation adopted in the theory of approximation of functions is the following. For a given continuous function f, define the *degree of approximation* of f by

$$E_n(f) = \min_{p\, \in\, \mathcal{P}_n} \|f - p\|_\infty,$$

where $\|g\|_\infty$ is the infinity norm of a continuous function g, on the interval $[\alpha, \beta]$,

$$\|g\|_\infty = \max_{t\in\ [\alpha\ \beta]} |g(t)|\ .$$

The Weierstrass theorem states that any continuous function f can be uniformly approximated by polynomials [7]. In particular this means that $\lim_{n\to\infty} E_n(f) = 0$. In the early 1900s, Jackson proved a number of theorems which give further information on this convergence. The following is the third of the Jackson theorems. Another definition is needed before stating the theorem: The *modulus of continuity* of a bounded function f on an interval $[\alpha,\ \beta]$ is defined as

$$\omega_f(\delta) = \sup_{|t_1 - t_2|\ \leq \delta} |f(t_1) - f(t_2)|. \tag{21}$$

THEOREM 2.4 (Jackson's theorem III). *For all functions $f\ \in C[0\ 2\pi]$,*

$$E_n(f) \leq \omega_f\left(\frac{\pi}{n+1}\right)\ . \tag{22}$$

See [7] for proofs and additional details. For an arbitrary interval $[\alpha, \beta]$ the above theorem translates into

$$E_n(f, [\alpha, \beta]) \leq \omega_f \left(\frac{\beta - \alpha}{2(n+1)} \right) . \tag{23}$$

Applying the above result to base filter functions is easy.

LEMMA 2.5. *Let the base filter function ϕ be the spline function constructed as*

$$\phi(t) = \begin{cases} 1 & \textit{for} \quad t \in \ [0, \tau_0), \\ \Theta_{[m_0, m_1]} & \textit{for} \quad t \in \ [\tau_0, \tau_1), \\ 0 & \textit{for} \quad t \in \ [\tau_1, \beta]. \end{cases}$$

Then,

$$\omega_\phi(\delta) \leq \frac{2\eta'_{max}}{\tau_1 - \tau_0} \delta,$$

where η'_{max} is given by (17) *and is approximated by* (18) *for large values of m_0, m_1.*

Substituting this result into Jackson's theorem, we obtain the following bound.

PROPOSITION 2.6. *Let ϕ be the base filter function defined in Lemma* 2.5. *Then,*

$$E_n(\phi) \ \leq \frac{\beta \ \eta'_{max}}{(n+1)(\tau_1 - \tau_0)}, \tag{24}$$

where η'_{max} is given by (17) *and is approximated by* (18) *for large values of m_0, m_1.*

The above result is about convergence in the ∞-norm. Obtaining a result for the L-2-norm with the weight function w is straightforward and standard because the norms are related to each other in a simple way. Specifically, the following is easily shown:

$$\|g\|_w \leq K \|g\|_\infty \quad \text{with} \quad K = \|1\|_w.$$

For example, if we have l intervals and the μ_i's are equal to 1 in (19), then $K = \sqrt{l \ \pi}$.

3. Applications and extensions.

Polynomial filtering has many applications in numerical linear algebra and related areas. In fact, we can argue that the number of these applications is likely to increase because of the growing need to solve problems in reduced dimensions and to apply various forms of PCA. In [19], we have considered the use of polynomial filters in information retrieval. The paper [18] exploits similar ideas for the problem of eigenfaces. Here we examine a few other applications which may also benefit from polynomial filtering. Though we will show a few supporting experiments shortly, the ideas are exposed here only to describe the rationale and the concepts, and some of these ideas will be further explored in forthcoming articles.

3.1. Computing a large invariant subspace.

In this section we show how polynomial filtering can be used to compute large invariant subspaces of symmetric real (or Hermitian complex) matrices. Specifically, the following problem is addressed: *Compute all eigenvalues of A located in a certain subinterval of the spectrum along with associated eigenvectors.*

The simplest form of this problem is to compute all eigenvalues of A that are $\leq \tau$. It can be assumed that an upper bound β for the spectrum is available, and, without loss of generality, that all eigenvalues are ≥ 0. Consider this case first. One solution to the problem is to use the Lanczos algorithm for the matrix $q(A)$, where q is a low-pass filter polynomial such that $q(\lambda) \approx 1$ for $0 \leq \lambda \leq \tau$ and $q(\lambda) \approx 0$ for $\tau < \lambda \leq \beta$. To reduce cost, the polynomial should not be of high degree. What might happen with this approach is that the Lanczos procedure will quickly produce a good invariant subspace associated with the largest eigenvalues of $q(A)$. If enough steps are taken, then clearly this subspace should include the desired subspace, which could be easily extracted by a simple Rayleigh–Ritz projection. The main point is that a shorter basis is required because the Lanczos algorithm will converge faster, and this will lead to a much lower cost due to much less expensive orthogonalization steps. Indeed, it was observed in [4] that, for large invariant subspaces, the high cost of orthogonalization far outweighs the additional cost of the matrix-vector products with $p(A)$. This comes with the added benefit of using less memory.

The procedure described above can be enhanced by filtering the initial vector of the Lanczos procedure. The reason why this could be useful is the observation that if v has a zero component with respect to $\lambda_i > \tau$, then since $q(\lambda_i)$ is close to zero, the components of the Lanczos vectors will also remain close to zero throughout the algorithm. We can use a high-degree polynomial to filter the initial vector and then a low-degree polynomial for the inner loop of the Lanczos procedure. Initial results show that this process works as predicted and may lead to good savings in time when compared with standard approaches.

Next we provide a motivation for this approach based on an application from quantum mechanics (for details, see [4]) and then explore in detail the case of interior eigenvalues.

3.2. Motivation. In electronic structures calculations one is faced with the problem of computing an orthogonal basis of the invariant subspace associated with the k lowest eigenvalues of a Hamiltonian matrix. This particular problem was the original motivation for this work. The Hamiltonian is (real) symmetric. A major difficulty with these calculations is that the dimension k of the subspace can be quite large. A typical example would be that $k = 1,000$ and that n, the dimension of the matrix, is $n \approx 1,000,000$. Methods based on standard restarted Lanczos procedures tend to suffer from the need to save a very large set of basis vectors as well as from the need for a very large number of costly restarts and reorthogonalizations. An alternative considered recently is to forego the restarts and not focus on individual eigenvectors; see, e.g., [4]. This approach is usually faster than the implicit restarted version of Lanczos, but it may require the use of secondary storage as the Lanczos basis can be quite large.

As an illustration consider a hypothetical situation where, for example, $m = 2000$ Lanczos vectors are required by a standard Lanczos procedure to compute a subspace of dimension $k = 100$. The cost of orthogonalization will be $0.5m^2 \times n$, which is $2 \times 10^6 \times n$ operations. In contrast, if polynomial filtering is used in the manner described earlier and if only 200 vectors are needed, the new cost will $10^4 \times n$ plus the additional cost of matrix-vector products. If degree 10 polynomials are used and the matrix has, say, 13 nonzero entries per row, then this additional cost is roughly $200 * 10 * 13n = 26000n$. So the total adds up to $\approx 36,000n$ operations versus $2,000,000n$. Of course this example is hypothetical and somewhat extreme, but it underscores the unacceptable cost of orthogonalization for large bases. One may argue that a much smaller basis might be needed for the restarted Lanczos method. Though the situation is generally difficult to analyze, the point remains that restarting is expensive because eigenvectors are repeatedly (implicitly) computed. It is not the

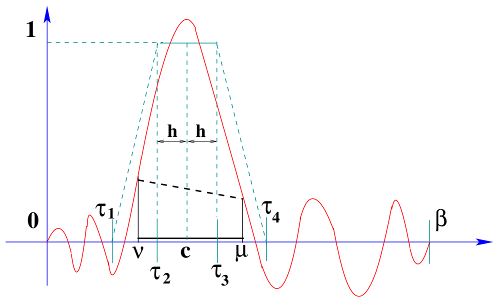

FIG. 5. *Illustration of the procedure to set up the intervals and the base filter function ψ. The polynomial shown is the polynomial which results from approximating ψ with the choice of $\tau_1, \ldots, \tau_4$.*

goal of this paper to compare these approaches. This is done in another article [4], where these comparisons are undertaken for realistic problems arising from electronic structures calculations.

3.2.1. Interior invariant subspaces. A case not considered in [4] is the situation of interior eigenvalues. Though the overall scheme is not too different from that of the computation of the smallest or largest eigenvalues, a few difficulties arise which make the scheme somewhat more complex. One of the main difficulties lies in the selection of the (dual) base filter ψ. Suppose we want all eigenvalues in the subinterval $[\eta,\ \mu] \subset [0\ \beta]$. To construct the base filter ψ we will need to subdivide the interval $[0,\ \beta]$ into five subintervals, $[0,\ \beta] \equiv [\tau_0,\ \tau_1] \cup [\tau_1,\ \tau_2] \cup \cdots \cup [\tau_4,\ \tau_5]$, with $0 = \tau_0 < \tau_1 < \tau_2 < \tau_3 < \tau_4 < \tau_5 = \beta$. In the intervals $[\tau_0\ \tau_1]$ and $[\tau_4\ \tau_5]$, the function ψ takes the value 0. In the central interval, $[\tau_2,\ \tau_3]$, the function ψ has value 1. The other two intervals bridge the values 0 and 1, and so the global function ψ is continuous and sufficiently smooth.

We would like to use the Lanczos algorithm on the matrix $p(A)$, where $p(\lambda) = \lambda s(\lambda)$ is the polynomial approximation to the filter ψ. In order not to miss eigenvalues in the desired interval it is essential that $p(\lambda_i)$ be larger than $p(\lambda_k)$ for each $\lambda_i \in [\eta,\ \mu]$ and $\lambda_k \notin [\eta\ \mu]$. Because of the likely imbalance between the left and right branches of the polynomial, it is not easy to guarantee this without an iterative process for selecting the intervals and ψ. The goal of the iterative process is to guarantee that $p(\mu) = p(\eta)$ and that all eigenvalues inside the interval $[\eta,\ \mu]$ will be mapped to the largest eigenvalues of $p(A)$. To achieve this, a bisection algorithm is applied, whereby τ_2, τ_3 are changed until the relation $p(\mu) = p(\eta)$ is approximately satisfied. A few details on this procedure follow. The discussion is illustrated in Figure 5.

Initially, the values of τ_1 and τ_4 are fixed so that $\tau_1 = \nu - \delta$ and $\tau_4 = \mu + \delta$ for a certain δ (our code uses $\delta = 0.05 * (\mu - \nu)$). Then what is left is to determine τ_2, τ_3. These are set to be of the form $\tau_2 = c - h$ and $\tau_3 = c + h$, where h is a small fraction of the interval width (our code uses $h = (\tau_4 - \tau_1)/10$). This means that the desired "plateau" interval for p is chosen to be of the form $[\tau_2,\ \tau_3] = [c - h,\ c + h]$, where h is fixed and c is to be found. Now the only unknown left is c, which is determined by bisection so that the resulting p satisfies $p(\eta) = p(\mu)$.

The figure reveals another potential problem. Recall that the goal is to use the Lanczos algorithm with the matrix $p(A)$. In order to be able to stop the iteration, it is necessary to know whether the required eigenvalues have converged. Following [4], this is done without computing eigenvectors, but by only considering the tridiagonal matrix T_m generated from the Lanczos iteration. If the sum of those eigenvalues of T_m which correspond to the wanted eigenvalues of A has converged, then the process is stopped. These eigenvalues are $p(\lambda_i)$ for $\lambda_i \in [\nu, \mu]$. This stopping criterion is modeled after the one in [4], where the restricted trace (sum of desired eigenvalues) has an important physical meaning (total energy) and is therefore the proper quantity to monitor for convergence. In the smallest / largest eigenvalues case, the situation is simple: the eigenvalues $p(\lambda_i)$ of T_m corresponding to the desired λ_i are (in general) the largest eigenvalues of T_m. This facilitates the test. For the case of interior eigenvalues and middle-pass filters, the situation is not as straightforward. Figure 5 illustrates this, since there are points λ at the right of τ_4 whose values $p(\lambda)$ are larger than some values $p(\lambda)$ for λ inside the interval $[\nu,\ \mu]$. We handle this by a heuristic iterative procedure. Once the value of c has been obtained by bisection, it is necessary to check whether the following condition is satisfied:

$$\sup_{\lambda\in[0,\ \nu)\cup(\mu,\ \beta]} |p(\lambda)| \ < \ \min_{\lambda\in[\nu,\ \mu]} |p(\lambda)| \ .$$

If this is not satisfied, then the interval $[\tau_1, \tau_4]$ is expanded by doubling the value of δ. At the same time h is halved, leading to a shrinking of the plateau interval $[\tau_2, \tau_3]$. The process is then repeated until a satisfactory interval is found. In addition to this, the function which determines the interval also returns the maximum value, say γ, of $p(\lambda)$ outside the interval $[\nu,\ \mu]$ so as to recognize which eigenvalues λ of A do belong to the desired interval $[\nu,\ \mu]$: If the eigenvalue $p(\lambda)$ is $> \gamma$, then λ must belong to $[\nu,\ \mu]$. Note that λ is not available; only $p(\lambda)$ is.

The main point of the above discussion is that the polynomials are easy to use, and it is inexpensive to work with them, so some careful preprocessing can be done to ensure that the correct eigenspace is computed.

In the following algorithm, p_0 denotes the "prefilter" polynomial, while p is the filter polynomial used in the iteration. Typically, the prefilter polynomial is of high degree (e.g., 200), while the internal polynomial is of low degree (e.g., 20).

Algorithm 3.1. *Filtered Lanczos.*

```
Input:   Matrix $A \in \mathbb{R}^{n\times n}$, starting vector $q_1$, $\|q_1\|_2 = 1$, scalar $m$,
         Interval of desired eigenvalues $[\nu, \mu]$
         Degrees of prefilter and filter polynomials
Output: Eigenvalues of $A$ in interval $[\nu, \mu]$ + eigenvector basis.
  0.  Call getIntv to obtain good intervals and base filter.
  1.  Set $\beta_1 = 0$, $q_0 = 0$
  2.  If prefilter degree is $> 0$ then prefilter initial vector $q_1 := p_0(A)q_1$, $q_1 = q_1/\|q_1\|_2$
  3.  for $i = 1, \ldots, m$
  4.      $w_i = p(A)q_i - \beta_i q_{i-1}$
  5.      $\alpha_i = < w_i, q_i >$
  6.      $w_i = w_i - \alpha_i q_i$
  7.      $\beta_{i+1} = \|w_i\|_2$
  8.      if $(\beta_{i+1} == 0)$ then stop
  9.      $q_{i+1} = w_i/\beta_{i+1}$
 10.      Let $T_i = \text{tridiag}(\beta_i, \alpha_i, \beta_{i+1})$.
 11.      Compute eigenvalues and eigenvectors of $T_i$. Sort decreasingly.
 12.      Let $n_{ev}$ = number of eigenvalues $\lambda_j^\star$ of $T_i$ such that $\lambda_j^\star > \gamma$.
 13.      Compute $s_i = \sum_{\lambda_j^\star > \gamma} \lambda_j^\star$
 14.      if $(|s_i - s_{i-1}| < |s_{i-1}| * \texttt{tol})$ then break
```

```
15. end
16. For $j = 1 : n_{ev+2}$ do :
17.     compute Ritz vectors $z_j$ of $T_i$.
18.     If $\tilde{\lambda}_j = (Az_j, z_j) \notin [\nu,\ \mu]$ reject $\tilde{\lambda}_j, z_j$
19. end
```

A few comments are in order. The function `getIntv` referenced in line 0 is the heuristic procedure discussed above, which carefully determines the five subintervals of $[0, \beta]$ and the initial filter ψ. It returns in particular the scalar γ used in line 12, which is such that if $p(\lambda) > \gamma$, then $\lambda \in [\nu,\ \mu]$. In line 16, we compute n_{ev+2} Ritz pairs instead of n_{ev} as a safeguard only. The test in the next lines will keep only the required eigenvalues. The eigenvalues $\lambda_j^\star$ are eigenvalues of T_i, and they approximate eigenvalues of $p(A)$. The convergence test in lines 13–14 need not be executed at each step (this is an $O(i^2)$ process); we can instead perform it at regular intervals. Finally, it is clear that it is essential to include some form or reorthogonalization. In [4] we used partial reorthogonalization.

3.3. Computing $\boldsymbol{f(A)v}$. The procedures described earlier compute approximations to $\phi(A)v$, where ϕ is a specific spline function on up to five intervals. There is, of course, no reason why one should be limited to spline functions which approximate filters. The approach can be extended to other situations where a vector of the form $f(A)v$ is to be computed. The problem of approximating $f(A)v$ has been extensively studied (see, e.g., [25, 23, 5, 16, 15]), though the attention was primarily focussed on the case when f is analytic (e.g., $f(t) = \exp(t)$). Problems which involve noncontinuous functions, such as the step function or the sign function, can also be important. The approach described in this paper can be trivially extended to the case where ϕ is a general spline function. One can certainly imagine situations where a certain vector $f(A)v$ is to be evaluated, where f is some complex function known through an accurate piecewise polynomial approximation. The framework developed in this paper is ideally suited for handling this situation. The only extensions required are to increase the number of intervals (which is now ≤ 5) and to define ψ in each of these intervals by the polynomials of the spline function.

Another interesting application is when approximating $\psi(A)$, where ψ is the sign function. Computing the sign function of a matrix has important applications in QMC (quantum chromo dynamics); see, e.g., [13]. In this case we need to use three intervals, for example, $[a_-\ d_-], [d_-\ d_+], [d_+\ a_+]$, where d_-, a_- are negative and d_+, a_+ are positive. The difficulty here is to compute estimates for the interior values d_- and d_+.

3.4. Estimating the number of eigenvalues in an interval. The most common way to compute the number of eigenvalues inside an interval is to exploit the Sylvester inertia theorem and the LDL^T factorization [14]. However, for large matrices this is not always practically feasible, or it may be too expensive.

It is sometimes useful to obtain a rough idea of the number of eigenvalues of a Hermitian matrix that are located inside a given interval. This information can be used, for example, for the case when the smallest k eigenvalues of A must be computed by using a form of polynomial filtering. In this situation an interval $[0,\ \tau]$ must be found which contains these k eigenvalues. A guess for τ can be given and then refined by answering the question: How many eigenvalues are located on the left of τ?

One possible solution to this can be provided by the Lanczos procedure. One can simply run the Lanczos algorithm without reorthogonalization (the Cullum–Willoughby algorithm; see [8]) or with partial reorthogonalization and record the number n_τ of all eigenvalues below τ of the tridiagonal matrix T_m obtained from the Lanczos algorithm. When this number stabilizes (i.e., all eigenvalues below τ converge), then n_τ will represent the desired number. The problem with this approach is that it may be very expensive when the number n_τ is large.

A rough approximation of n_τ can be easily obtained from statistical arguments, using polynomial filtering. This technique is an adaptation of methods described elsewhere for estimating the trace of certain operators; see, for example, [17, 21, 3]. Consider a low-pass polynomial filter such as the one shown in Figure 4, and an arbitrary vector v of 2-norm unity. Expand the vector v in the eigenbasis as

$$v = \sum_{i=1}^{n} \xi_i u_i,$$

and consider the inner product of v with $p(A)v$:

$$(v, p(A)v) = \sum_{i=1}^{n_\tau} \xi_i^2 p(\lambda_i) + \sum_{i=n_v+1}^{n} \xi_i^2 p(\lambda_i).$$

If the polynomial p is selected so that it is close to 1 on $[0,\ \tau]$ and to 0 in $(\tau,\ \beta]$, then clearly the second sum in the above expression should be close to zero, and the first close to the sum $\sum_{i=1}^{n_\tau} \xi_i^2$. If the vector v is a random vector, then the ξ_i's are unbiased, and therefore the ratio $\sum_{i=1}^{n_\tau} \xi_i^2 / \sum_{i=1}^{n} \xi_i^2$ should be close to n_τ/n. In the end we can estimate n_τ by

$$n_\tau \approx n \times (v, p(A)v). \tag{25}$$

Of course, a unique sample may not be good enough, and several trials should be taken and the results averaged. The numerical experiments sections explore this approach a little further. It should be emphasized that, as is always the case, it is expensive to obtain an accurate answer by statistical methods in general. Accordingly, this approach may be useful only when a rough estimate of n_τ is wanted and other methods cannot be considered. Two appealing features of the method are its exclusive reliance on matrix-vector products and its highly parallel nature.

4. Numerical tests.

Applications of filtered polynomial iterations to information retrieval and face recognition have been reported elsewhere [18, 19]. In addition, the use of these ideas for computing large eigenspaces has recently been successfully exploited; see [4]. Section 4.2 explores this further.

The goals of the tests discussed in this section are (a) to examine the convergence of the process, (b) to show and compare a few of the techniques discussed earlier for computing invariant subspaces, and (c) to demonstrate the use of polynomial filtering for approximating inertia of shifted matrices (see section 3.4).

All tests were performed with Matlab on a Linux workstation (running Debian) and equipped with two 1.7 GHz Xeon processors (with 256kB cache) and 1 GB of main memory.

4.1. Convergence.

In this test we generate a matrix obtained from the discretization of a Laplacian using centered differences on a 25×15 mesh. We then compute the vector v, which has all components equal to 1 in the eigenbasis; i.e., v is the sum of all the (normalized) eigenvectors. This vector is then filtered with a chosen

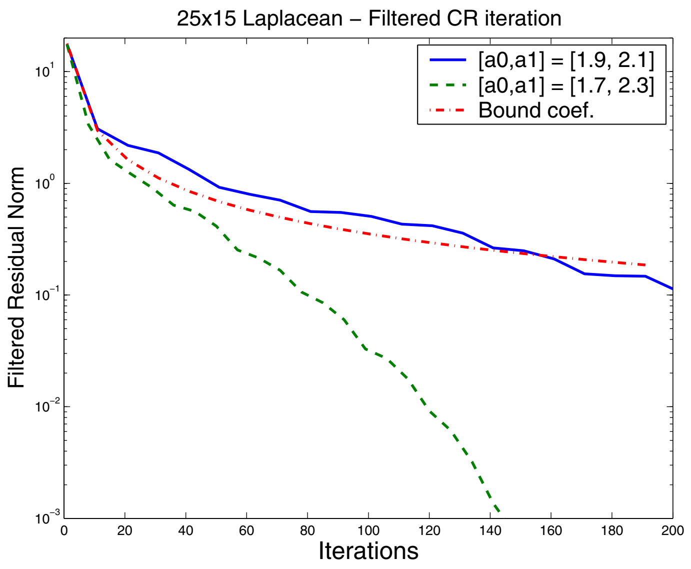

FIG. 6. *Convergence of filtered polynomial CR algorithm for two different cases, and comparison with the coefficient of the bound* (24).

low-pass base filter ϕ, and we plot $\|\phi(A)v - (I - As_{k-1}(A))v\|_2$ for $k = 1, \ldots, 200$. This is referred to as the "filtered residual." The low-pass filter is selected as follows:

$$\phi(t) = \begin{cases} 1 & \text{for} \quad t \in \ [0, \tau_0), \\ \Theta_{[m_0,m_1]} & \text{for} \quad t \in \ [\tau_0, \tau_1), \\ 0 & \text{for} \quad t \in \ [\tau_1, \beta]. \end{cases} \tag{26}$$

A first run used the values $m_0 = m_1 = 10$, $\beta = 8$, $\tau_0 = 1.9, \tau_1 = 2.1$, and the second used the same values for m_0, m_1, and β, and changed τ_0, τ_1 to $\tau_0 = 1.8, \tau_1 = 2.2$. The plot in Figure 6 shows three curves. The first two show the progress of the filtered residual norm for the two runs (solid line and dashed line, respectively). The third one (dash-dot) shows the coefficient in the right-hand side of (24) corresponding to the first test case ($m_0 = m_1 = 10$, $\beta = 8$, $\tau_0 = 1.9$). Here, η'_{max} is estimated by (18), where for $m_0 = m_1 = 10$ we find that $\eta'_{max} \approx \sqrt{m_0/\pi}$. So the third curve shows exactly the sequence

$$\frac{8\sqrt{10/\pi}}{0.4 * (i+1)}, \quad i = 1, \ldots, 200.$$

Two observations can be made. The first is that for the second run, the behavior is not at all predicted by the bounds. It has an exponential character not seen in the bounds obtained in section 2.7. The second observation is the big difference in convergence between two seemingly close situations. If the middle interval increases in width, we can get very fast convergence. However, note that taking a wide middle interval may yield a function that is not desirable from other viewpoints; i.e., there may be situations when this interval must be taken to be small. In information retrieval this is not critical [19]. When computing invariant subspaces, on the other hand, it is undesirable to have a wide gap since it will include eigenvalues that need to be eliminated by some other means.

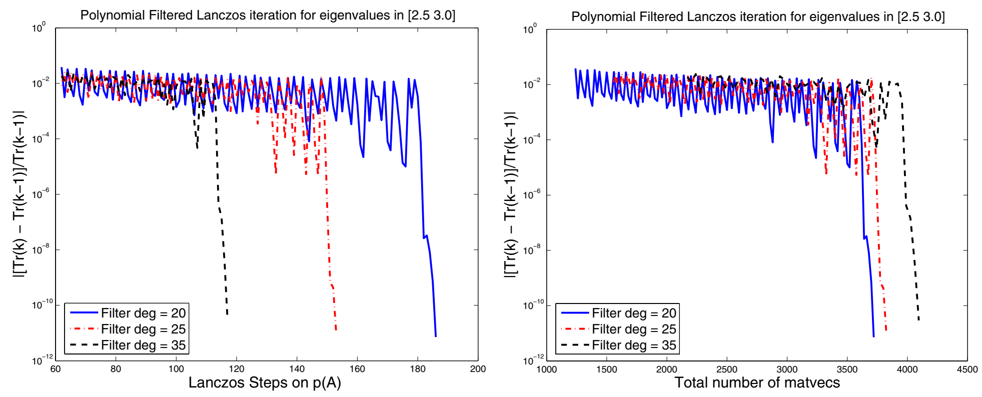

FIG. 7. *Convergence of a filtered Lanczos iteration for computing all eigenvalues of a Laplacian of dimension* 891*, located in the interval* [2.5, 3].

4.2. Computing invariant subspaces. Polynomial filtering can be helpful in the situation when a large invariant subspace associated with all eigenvalues in a given interval is to be computed. In [4] we have shown how the method can be applied to realistic problems arising from electronic structures. The problem there is that of computing an invariant subspace associated with the smallest eigenvalues of a large symmetric real matrix. The ideas used in [4] follow closely those sketched in section 3.1. For matrices such as those that arise in electronic structures, the method works well because the invariant subspace is quite large, and this causes methods which rely too much on orthogonalization to become excessively expensive. This little fact, which has not been adequately addressed by researchers in numerical methods, is well-known to researchers in the physics community. By reducing the frequency as well as the cost of orthogonalization, one can reduce the overall cost dramatically. Thus, a strategy based on polynomial filtering combined with the inexpensive "partial reorthogonalization" resulted in gains close to a factor of 12 in some cases (see [4]) relative to standard existing codes such as ARPACK [20].

Since the case corresponding to the smallest (or largest) eigenvalues has already been covered in detail in [4], we will illustrate next the case of interior eigenvalues. This case is just as important, because there is currently a lack of good algorithms for dealing with it.

In the experiments which follow, we consider a model problem arising from a Laplacian matrix. The matrix corresponds to the discretization of the Laplacian on an $(nx + 2) \times (ny + 2)$ grid including boundary points. After applying zero Dirichlet boundary conditions, we obtain a matrix of dimension $n = n_x n_y$.

In the first test, we take $n_x = 27, n_y = 33$. This results in a matrix of dimension $n = 891$. We would like to compute all eigenvalues of A in the interval $[\nu,\ \mu] = [2.5,\ 3]$. As it turns out, there are 60 eigenvalues in this interval. We ran three tests with a different degree of the filter polynomial: degree 20, 25, and 35. We did not apply prefiltering. The base filter function uses bridge functions of the form $\Theta_{[10,10]}$. The boundaries for the various intervals defining the base filter function were set up as described in section 3.1. The same initial vector for the Lanczos iteration was used for all three runs, and it was generated randomly. Algorithm 3.1 was run with tol= $1.e-10$. The plots in Figure 7 show the error measure $|s_i - s_{i-1}|/|s_{i-1}|$ used in lines 13–14 of Algorithm 3.1 to test convergence. These error rates are plotted against the

TABLE 1

Number of Lanczos steps and sum of final eigenvalue errors as expressed by (27) *for the filtered Lanczos procedure using three different filtering polynomials..*

Degrees	20	25	35
Lancz. steps	190	157	120
Error-sums	6.77e-12	4.631e-12	5.570e-11

TABLE 2

Number of Lanczos steps and sum of final eigenvalue errors as expressed by (27) *for the filtered Lanczos procedure using two different filtering polynomials. The matrix is a* 3*-D Laplacian of dimension* $n = 10,051$.

Degrees	75	80
Lancz. steps	364	270
Error-sums	5.684e-14	1.430e-13

number of Lanczos steps (left figure) and against the total number of matrix-vector products (matvecs; right side).

As expected, the number of Lanczos steps decreases as the degree of the polynomial increases. In the case of degree 35, the procedure requires 120 Lanczos steps to compute all 60 eigenvalues. This good performance comes at the cost of 35 matvecs with A per Lanczos step. This amounts to $175n$ operations per Lanczos steps for the matvec, and the total number of matvecs with A is 4200. For the lower degree of 20, we now need 190 steps, so we need a total of 3800 matvecs. Though this is lower than with the degree 35, the comparison favors the higher degree if the cost of orthogonalization is taken into account (a full reorthogonalization is performed). The total number of matvecs may appear to be quite high. However, the alternative of running the Lanczos algorithm to compute eigenvalues from the first to the last one in the interval may be much more costly for realistic cases because of the cost of orthogonalization. This was demonstrated for the computation of smallest eigenvalues in a realistic computation in [4].

In order to verify that the code run does not miss eigenvalues we printed the errors

$$\sum_{\lambda_i \ \in \ [\nu, \ \mu]} \min_j |\tilde{\lambda}_j - \lambda_i|, \tag{27}$$

where the $\tilde{\lambda}_j$ are the approximate Ritz values computed in lines 16–18 of Algorithm 3.1. These are printed in Table 1 along with the number of Lanczos steps required for convergence.

The next test proceeds along the same lines but considers a more difficult problem. We take a three-dimensional (3-D) Laplacian with $n_x = 23$, $n_y = 23$, and $n_z = 19$, leading to a problem of size $n = 10,051$. The eigenvalues of A are located in the interval $[0, \ 12]$, so to make the problem more challenging we try to compute eigenvalues around the middle of the interval. Specifically, we seek to compute all eigenvalues in the interval $[\nu, \ \mu] = [6.25, \ 6.30]$. There are 53 eigenvalues of A in this interval. This particular example will require higher-degree polynomials than the smaller example seen above to reach convergence in a small number of Lanczos steps. We take polynomials of degrees 75 and 100. For the filter function the bridge functions are of the form $\Theta_{[25,15]}$. Results similar to the ones seen above are shown in Figure 8 and Table 2.

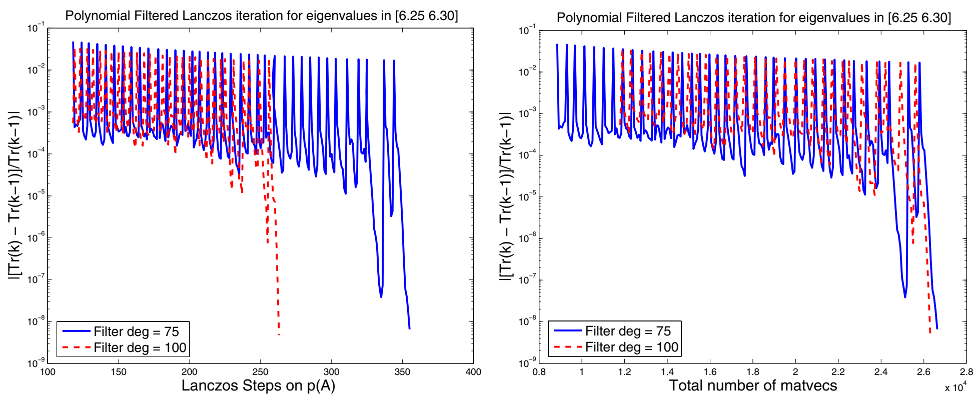

FIG. 8. *Convergence of a filtered Lanczos iteration for computing all eigenvalues of a Laplacian of dimension* 10, 051 *located in the interval* [6.25, 6.30].

It is interesting to ask the question: What are the alternatives to this approach for solving this problem? In this case, the best known method is to use a shift-and-invert technique, whereby the Lanczos procedure is applied to $(A - \sigma I)^{-1}$. However, factoring the matrix $A - \sigma I$ can be quite expensive, especially for 3-D problems of this type and when considering the fact that the matrix is highly indefinite. For very large 3-D problems factorization may not even be a feasible option. The other alternative is to employ a Lanczos-type procedure to compute a large number of eigenvalues until those of interest are reached. Orthogonalization and the need to keep a large basis will be two serious problems for large matrices. Polynomial filtering can be attempted in such cases. An approach of this type was suggested in [2, 1] in an algorithm which exploits implicit restarts. The IRBL code (Matlab) presented in [2] uses Leja points for the purpose of acceleration, instead of the least-squares polynomials used in this paper. The other major difference is that IRBL is a block algorithm.

Another idea that is similar in spirit to the one described in this paper is presented in [10]. There, a polynomial is constructed by compounding a quadratic polynomial with a higher degree Chebyshev polynomial in order to obtain a desired filter. In fact, the paper [10] explores several other methods for computing interior eigenvalues. For their problem, called the Anderson model of localization, the authors found that the best approach is the Cullum–Willoughby [8] technique based on the Lanczos algorithm without reorthogonalization. The problem in [10] is somewhat different from the one addressed here in that the number of eigenvalues/ eigenvectors computed is relatively small (all tests were with five eigenpairs).

As pointed out in [10] and elsewhere (see, e.g., [26]), the potential difficulty with any polynomial filtered approach is the high cost of the procedure if large-degree polynomials are required. Though we do not offer comparisons with competing methods, we can say that polynomial filtered Lanczos procedures are likely to be superior to competing techniques in some situations. Specifically, they may offer the best alternative in situations when (a) a large number of eigenvalues and eigenvectors must be computed, (b) matrix-vector products are not expensive, and (c) there are not too many eigenvalues around the interval boundaries ν, μ. Condition (a) is based on the observation that when the subspace is large the cost of the eigenvalue calculation is dominated by orthogonalization. The result is that a big part of this cost can be traded off with filtering, which leads to fewer steps in the Lanczos algorithm. Condi-

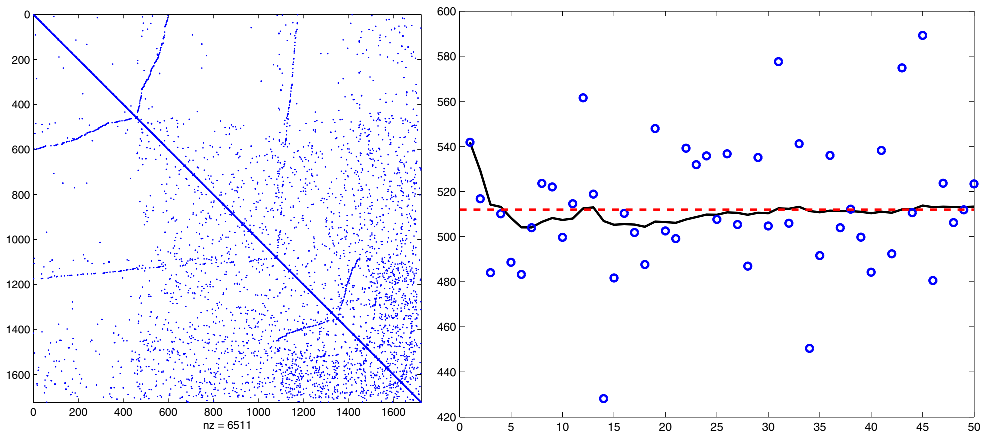

FIG. 9. *Pattern of matrix* `bcspwr09` *(Left) and stochastic estimate of its number of negative eigenvalues (right).*

tion (b) will ensure that convergence will be reached without resorting to a polynomial of too high a degree.

4.3. Estimating the number of eigenvalues in an interval. This section reports on a test with the stochastic estimator of the inertia of a shifted matrix, i.e., the number of eigenvalues of a matrix that are below a certain number α. Section 3.4 suggested a simple algorithm for this calculation for the case when a rough estimate of this number is wanted.

For this test we took a matrix from the Harwell–Boeing collection [9], namely the matrix `bcspwr09`. This matrix is of size $n = 1,723$ and has $nnz = 6,511$ nonzero entries. The sparsity pattern of the matrix is shown on the left side of Figure 9. This matrix has all its eigenvalues in the interval $[-3.117\ldots, 5.971\ldots]$. The question one may ask is: How many eigenvalues are negative? The correct answer is 512. We shifted everything by 3.2 (so A becomes $A + 3.2I$) and we sought the number of eigenvalues of the shifted matrix that are below $\alpha = 3.2$. A dual filter ψ using three intervals, defined as in (26), was used with the parameters: $m_0 = m_1 = 10$. The interval bounds given were $0, \tau_0 = 3.15, \tau_1 = 3.25, \beta = 6$. The degree of the polynomial used was $m = 20$.

The right side of Figure 9 shows a test with 50 runs (each using a polynomial of degree 20 and a different unit random vector v). The number n_α reported for given k in the x-axis is simply the average of the numbers given by formula (25) over all previous k tests:

$$n_\alpha(k) = \frac{n}{k} \times \sum_{i=1}^{k} (v_i, p(A)v_i).$$

The small circles in the figure are the values of $n \times (v_i, p(A)v_i)$ obtained from each (single) sample. The dashed horizontal line represents the correct answer, which is 512. Notice that there are a few outliers, e.g., the smallest single estimate obtained was close to 428 and the largest close to 590, but the average over several runs quickly converges to a reasonable estimate. So after 30 runs (a total of 600 matrix-vector products), a fairly good estimate is reached.

5. Conclusion.

Polynomial filtering is a useful and versatile tool in computational linear algebra. It is most appealing in situations where rough solutions to various matrix problems are sought. We have shown a few such applications, and hinted at others, where approximations to the matrix problem are sought which are restricted to be in a small space.

Apart from the methods related to low-rank approximations mentioned above, polynomial filtering has also been tried in the past with limited success in the more traditional areas of matrix computations, for example for the problem of preconditioning. Polynomial filtering is not a panacea, but it can play a significant role in specific cases. Perhaps the most important of these is the computation of large invariant subspaces. A successful use of polynomial filters in a realistic application has already been reported elsewhere [4].

There are many other potential uses of polynomial filtering in numerical linear algebra which remain to be explored. Many computations require the solution of least-squares systems with regularization. We also hinted at the problem of computing $f(A)b$ when f is a spline function, which can itself be an approximation to an arbitrary function.

REFERENCES

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

[2] J. Baglama, D. Calvetti, L. Reichel, and A. Ruttan, *Computation of a few close eigenvalues of a large matrix with application to liquid crystal modeling*, J. Comput. Phys., 146 (1998), pp. 203–226.

[3] C. Bekas, E. Kokiopoulou, and Y. Saad, *An estimator for the diagonal of a matrix*, Appl. Numer. Math., (2007), to appear.

[4] C. Bekas, E. Kokiopoulou, and Y. Saad, *Polynomial Filtered Lanczos Iterations with Applications in Density Functional Theory*, Technical Report umsi-2005-117, Minnesota Supercomputer Institute, University of Minnesota, Minneapolis, MN, 2005; SIAM J. Sci. Comput., submitted.

[5] L. Bergamaschi, M. Caliari, and M. Vianello, *Efficient computation of the exponential operator for discrete* 2*d advection-diffusion equations*, Numer. Linear Algebra Appl., 10 (2003), pp. 271–289.

[6] M. W. Berry and M. Browne, *Understanding Search Engines: Mathematical Modeling and Text Retrieval*, 2nd ed., Software Environ. Tools 17, SIAM, Philadelphia, 2005.

[7] C. C. Cheney, *Introduction to Approximation Theory*, McGraw-Hill, New York, 1966.

[8] J. K. Cullum and R. A. Willoughby, *Lanczos Algorithms for Large Symmetric Eigenvalue Computations: Vol.* I: *Theory*, Classics in Appl. Math. 41, SIAM, Philadelphia, 2002.

[9] I. S. Duff, R. G. Grimes, and J. G. Lewis, *Sparse matrix test problems*, ACM Trans. Math. Software, 15 (1989), pp. 1–14.

[10] U. Elsner, V. Mehrmann, F. Milde, R. A. Römer, and M. Schreiber, *The Anderson model of localization: A challenge for modern eigenvalue methods*, SIAM J. Sci. Comput., 20 (1999), pp. 2089–2102.

[11] J. Erhel, F. Guyomarc, and Y. Saad, *Least-Squares Polynomial Filters for Ill-Conditioned Linear Systems*, Technical Report umsi-2001-32, Minnesota Supercomputer Institute, University of Minnesota, Minneapolis, MN, 2001.

[12] B. Fischer and G. H. Golub, *How to generate unknown orthogonal polynomials out of known orthogonal polynomials*, J. Comput. Appl. Math., 43 (1992), pp. 99–115.

[13] A. Frommer, T. Lippert, B. Medeke, and K. Shilings, *Numerical Challenges in Lattice Quantum Chromodynamics*, Lectures Notes in Comput. Sci. 15, Springer-Verlag, Berlin, 1999.

[14] G. H. Golub and C. F. Van Loan, *Matrix Computations*, 3rd ed., Johns Hopkins University Press, Baltimore, MD, 1996.

[15] M. Hochbruck and C. Lubich, *On Krylov subspace approximations to the matrix exponential operator*, SIAM J. Numer. Anal., 34 (1997), pp. 1911–1925.

[16] M. Hochbruck, C. Lubich, and H. Selhofer, *Exponential integrators for large systems of differential equations*, SIAM J. Sci. Comput., 19 (1998), pp. 1552–1574.

[17] M. F. Hutchinson, *A stochastic estimator of the trace of the influence matrix for Laplacian smoothing splines*, Commun. Statist. Simula., 19 (1990), pp. 433–450.

[18] E. Kokiopoulou and Y. Saad, *PCA without eigenvalue calculations: A case study on face recognition*, in Proceedings of the Fifth SIAM International Conference on Data Mining, Newport, CA, 2005, SIAM, Philadelphia, 2005.

[19] E. Kokiopoulou and Y. Saad, *Polynomial filtering in latent semantic indexing for information retrieval*, in Proceedings of the ACM-SIGIR Conference on Research and Development in Information Retrieval, Sheffield, UK, 2004, ACM, New York, 2004.

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

[21] G. A. Parket, W. Zhu, Y. Huang, D. K. Hoffman, and D. J. Kouri, *Matrix pseudo-spectroscopy: Iterative calculation of matrix eigenvalues and eigenvectors of large matrices using a polynomial expansion of the dirac delta function*, Comput. Phys. Comm., 96 (1996), pp. 27–35.

[22] Y. Saad, *Iterative solution of indefinite symmetric linear systems by methods using orthogonal polynomials over two disjoint intervals*, SIAM J. Numer. Anal., 20 (1983), pp. 784–811.

[23] Y. Saad, *Analysis of some Krylov subspace approximations to the matrix exponential operator*, SIAM J. Numer. Anal., 29 (1992), pp. 209–228.

[24] Y. Saad, *Iterative Methods for Sparse Linear Systems*, 2nd ed., SIAM, Philadelphia, 2003.

[25] H. A. van der Vorst, *An iterative solution method for solving $f(A)x = b$, using Krylov subspace information obtained for the symmetric positive definite matrix A*, J. Comput. Appl. Math., 18 (1987), pp. 249–263.

[26] K. Wu, *Preconditioned Techniques for Large Eigenvalue Problems*, Ph.D. thesis, Department of Computer Science, University of Minnesota, Twin Cities, Minneapolis, MN, 1997.