Filtered Conjugate Residual-type Algorithms with Applications *

Yousef Saad †

May 21, 2005

Abstract

In a number of applications, certain computations to be done with a given matrix are performed by replacing this matrix by its best low rank approximation. This has the effect of yielding the most relevant part of the desired solution while discarding noise and redundancies. One such application is that of regularization where the right-hand side of the original linear system is noisy or inaccurate while the coefficient matrix is very ill-conditioned. Solving such linear systems accurately is counter-productive as the noise tends to be amplified. A common remedy is to compute the pseudo-inverse solution in which the inverses of the smallest singular values are replaced by zeros or small quantities. A similar procedure is also used in methods related to Principal Component Analysis, such as in Latent Semantic Indexing in information retrieval. Here the low-rank approximation to the original matrix is used to analyze similarities with a given query vector. This paper presents a few conjugate-gradient like methods to provide solutions to these two types of problems by iterative procedures which utilize only matrix-vector products.

Keywords: Conjugate Residual, Conjugate Gradient, Polynomial Filtering, Principal Component Analysis, LSI, Regularized solutions, Tychonov, SVD, TSVD.

1 Introduction

The approximate solution of the linear system

$$Ax = b \tag{1}$$

given at the k-th step of a Krylov subspace method, takes the form

$$x_k = x_0 + s_{k-1}(A)r_0$$

*Work supported by NSF grants ACI-0305120, DMR-0325218, and INT-0003274, by DOE under Grant DE-FG02-03ER25585, and by the Minnesota Supercomputing Institute

†Computer Science & Engineering, University of Minnesota, Twin Cities. saad@cs.umn.edu

where s_{k-1} is a polynomial of degree $\leq k-1$. The solution polynomial s_{k-1} is typically obtained from exploiting optimality. Thus, if x_* is the exact solution, the conjugate gradient method computes s_{k-1} so that the A-norm of the error $x_* - x_k$ is minimized while GMRES [25] computes s_{k-1} so that the 2-norm of the residual $b - Ax_k = A(x_* - x_k)$ is minimized. While this approach works well in standard cases, there are many instances where it is unsatisfactory. The best known of these situations is when A is very ill-conditioned and the right-hand side b is perturbed with noise. Solving such systems accurately will have the effect of amplifying the noise and will yield a solution that is useless in general. Examples of typical applications of this nature are in discrete inverse problems for image recovery and in tomography. Regularization methods deal with this issue by computing a *filtered solution*, i.e., by solving the system accurately only in the space associated with the large singular values. Two prototypes of these methods are the Truncated Singular Value (TSVD) technique, and Tychonov regularization [14].

A second type of problems which require filtering arises when computing the vector $A_k b$, where A_k is a k-rank 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 of A. These methods include all the techniques based on Principal Component Analysis, such as for example Latent Semantic Indexing (LSI, [5]).

In the simplest situation when A is square, symmetric, and semi-definite both classes of methods amount to to computing an approximation to the problem in the form

$$s(A)b \tag{2}$$

where s is a certain desired polynomial. In the case of regularization what is wanted is that $\lambda s(\lambda)$ be close to one for the largest eigenvalues and to zero for the smallest ones. Thus, we want $\lambda s(\lambda)$ to approximate a function ϕ such as the one shown in Figure 1. This means that the polynomial s should approximate $1/\lambda$ for the largest eigenvalues only. For PCA, the requirement is that the polynomial s, instead of $\lambda s(\lambda)$, approximates the filter ϕ.

Classical methods based on the Singular Value Decomposition, whether for regularization or for PCA, consist of approximating $A^\dagger$ (regularization) or A (PCA), 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 PCA methods 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. In regularization, $A^\dagger = V\Sigma^\dagger U^T$ is replaced by $A_k^\dagger = V\Sigma_k^\dagger U^T$. This Truncated SVD (TSVD) method for regularization amounts to replacing $A^\dagger b$ by an approximation of the form (2) where s is such that $\lambda s(\lambda)$ is exactly the step function shown in Figure 1. Similarly, SVD-based PCA is equivalent to replacing Ab by $s(A)b$ where $s(\lambda)$ equals the step function of Figure 1. An obvious limitation of the SVD-based approach is its excessive computational cost for large matrices. Indeed, this approach would entail a (complete or partial) SVD factorization of A.

Alternatives have been developed which are based on Krylov subspace methods. For example, in Tychonov regularization [27, 26, 14] the original system is replaced by the regularized system

$$\left(A^T A + \rho^2 I\right) x = A^T b$$

which is typically solved by the conjugate gradient algorithm. Because the matrix $A^T A$ is shifted, the number of CG steps that are required is likely to be moderate. In the symmetric

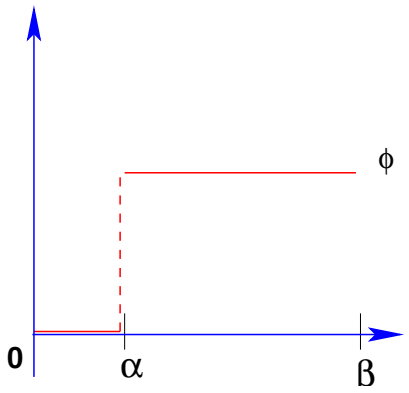

Figure 1: Step-function filter

case, the (converged) solution x_ρ of the above system is given by

$$x_\rho = \left(A^2 + \rho^2 I\right)^{-1} Ab \equiv s_\rho(A)b \ . \tag{3}$$

So, $x_\rho = s_\rho(A)b$, where $\lambda s_\rho(\lambda) \equiv \phi_\rho(\lambda)$ is the Tychonov filter function

$$\phi_\rho(\lambda) = \frac{\lambda^2}{\lambda^2 + \rho^2},$$

which can be viewed as a parameterized approximation to the step function ϕ of Figure 1.

A related method in regularization that is quite common in the case of low noise, is simply to use the conjugate gradient method for solving $A^TAx = A^Tb$ and stop the process prematurely, i.e., well before convergence of the approximate solution [14].

The algorithms to be described in the next sections are based on polynomial filtering. 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 or Conjugate Residual algorithms. The algorithms to be described can be used to compute solutions for both the problem of regularization and that of PCA within the same framework. In addition, they can easily be extended to other applications. As an example, by selecting the filter function to be the exponential function, the algorithms will yield a method for computing approximations to $exp(A)b$.

2 Polynomial Filtering

Two distinct problems which have important applications are addressed in this paper. The first can be formulated as the computation of the matrix vector product

$$x = Ab, \tag{4}$$

where A is an $n \times n$ matrix, and b is a vector. This computation in itself is trivial, except that it is the goal of many techniques to only extract the vector $\tilde{x}$ obtained from a low rank

approximation of the matrix A, typically the one associated with the largest singular values. The second problem is to compute the pseudo-inverse solution

$$x = A^{\dagger} b \ . \tag{5}$$

Again this computation is to be done approximately in the subspace associated with a few of the dominant singular values of A.

A representative case for (4) is in information retrieval, see, e.g., [5]. There, b represents a query vector and s the similarity vector, which is (after rescaling) the vector of cosines of all rows of A with the query b (typically this is formulated as $x = A^T b$ instead of (4) as a query is closer to a document - column of A in the usual notation used in LSI.). However, it has been observed that literal matching based on using these cosines directly faces many difficulties due to word usage. Latent Semantic Indexing solves this problem by computing Ab by $A_k b$ where A_k is obtained by only keeping the k largest singular values in the SVD of A, replacing the others with zeros.

Similarly, a representative for problem (5) would be any of the "regularization" methods [14]. These methods attempt to compute filtered pseudo-inverse solutions to (5) whereby the matrix is first approximated by its best low rank approximationA_k before computing x. In other words one strives to compute $\tilde{x}_k = A_k^{\dagger} b$ instead of the exact pseudo-inverse solution x in (5).

Consider first the problem of computing a regularized solution for a least-squares problem where A is not necessarily square. A polynomial filter approach to the problem consists of computing an approximation to $A^{\dagger} b$ in the form

$$A^{\dagger} b \approx x_s \equiv A^T s(AA^T) b,$$

where s is a certain polynomial. Note that the residual vector satisfies

$$b - Ax_s = \rho(A^T A) b \quad \text{with} \quad \rho(\lambda) \equiv 1 - \lambda \ s(\lambda) \ .$$

If $A = U \Sigma V$ is the SVD of A, then we have

$$b - Ax_s = V \rho(\Sigma^T \Sigma) V^T b = \sum_{j=1}^{m} \rho(\sigma_j^2) v_j (v_j^T b).$$

What is typically wanted is that the residual polynomial be small for the larger singular values. The requirements for the small singular values vary but it is typical to require that $s(\sigma_j) \approx 0$ for $\sigma_j \approx 0$. In this paper we will seek residual polynomials ρ which are such that

$$\rho(\sigma_j^2) \approx \begin{cases} 0 & \text{for large} \quad \sigma_j \\ 1 & \text{for small} \quad \sigma_j \end{cases} \ . \tag{6}$$

In other words, the residual polynomial ρ has characteristics opposite to those of the step function shown in Figure 1: it is a low-pass filter instead of a high-pass filter. The solution polynomial $s(\lambda)$ is close to the function zero in the neighborhood of zero and $1/\lambda$ for the larger singular values (second interval).

Consider now Problem (4). It is interesting to notice that for a residual polynomial ρ which satisfies the conditions (6), the polynomial $1 - \rho(\lambda) = \lambda s(\lambda)$ has the desired characteristics of the polynomial that is needed for Problem (4), namely it is close to one for the small singular values and to zero for the larger ones. For this reason, it is possible to only consider one of the two problems with this formalism as the solution to both problems will be provided by the same algorithm. This viewpoint will be exploited throughout the paper.

In summary, given a high pass filter function ϕ (close to one for large values, to zero for small values), we seek s so that $\|\phi(\lambda) - \lambda\ s(\lambda)\|_w$ is small - as measured by a certain norm w. Thus, the polynomial $\lambda s(\lambda)$ approximates the filter ϕ instead of the constant 1, as is done in standard methods. Equivalently, we can say that the goal is to minimize $\|(1 - \phi) - (1 - \lambda\ s(\lambda))\|_w$ over all polynomials s of degree $\leq k$. The focus is now on the residual polynomial $1 - \lambda s(\lambda)$ and the goal is to make this polynomial close to the function $1 - \phi$, a low-pass filter. While the vector $s(A)b$ provides a filtered solution to the system (5), the vector $b - As(A)b = \rho(A)b$ provides a filtered matrix-vector product which approximates (4).

2.1 Polynomial filters

In this section, we focus on the problem of filtered iterations for regularization. We begin with some notation as well as a rationale for the approach to be taken. The approximate solution vector obtained from Krylov subspaces is of the form $s_j(A)r_0$ 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 $j + 1$ and has value one at $\lambda = 0$.

We examine two ways to filter a solution. The first one uses a filter function ϕ explicitly: As discussed above, it will select $\lambda s_j(\lambda)$ so that it is close to close to $\phi(\lambda)$ on the spectrum. The second does not use a filter function explicitly. Instead, it makes the function $\lambda s_j(\lambda)$ close to one on the spectrum, but changes the L_2 inner product. Specifically, this approach combines a discrete and a continuous inner products. Both approaches will be based on formal Conjugate Gradient or Conjugate Residual algorithms described in polynomial spaces, and this is discussed next.

2.2 Conjugate Residual algorithms in polynomial spaces

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.

In the standard CR algorithm, the solution polynomial s_j minimizes $\|(I - As(A))r_0\|_2$ which is nothing but a discrete least-squares norm when expressed in the eigenbasis:

$$\|(I - As(A))r_0\|_2 = \left[\sum_1^N (1 - \lambda_i s(\lambda_i))^2\right]^{1/2} \equiv \|1 - \lambda s(\lambda)\|_D\ .$$

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

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

The generic algorithm is given below for reference.

ALGORITHM **2.1** *Formal Conjugate Residual Algorithm*

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

It is easy to show that the residual polynomial ρ_j generated by this algorithm minimize $\|\rho(\lambda)\|_w$ 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.

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

$$s_j(\lambda) = \alpha_0\pi_0(\lambda) + \cdots + \alpha_j\pi_j(\lambda) \ . \tag{7}$$

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

$$\langle \lambda\pi_j(\lambda), \lambda\pi_i(\lambda)\rangle_w = \langle \lambda\rho_j(\lambda), \rho_i(\lambda)\rangle_w = 0 \quad \textit{for} \quad i \neq j. \tag{8}$$

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 $\leq j$, then 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:

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

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 scaler 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 $\phi(\lambda) - \lambda s(\lambda)$ 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 conjugate gradient 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 Conjugate Residual 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 $\psi \equiv 1 - \phi$, in the least-squares sense. The function ψ is a low-pass filter whose value is close to one near zero and near zero for large eigenvalues. To make the computation tractable, the function ϕ 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

$$\|\phi(\lambda) - \lambda s_j(\lambda)\|_w = \min_{s \in \mathcal{P}_j} \|\phi(\lambda) - \lambda s(\lambda)\|_w \ . \tag{9}$$

Here 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 (9) is that

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

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 3-term recurrence and the approximation can be directly expressed in this basis. This was the approach taken in [10, 19].

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

$$s_j(\lambda) = s_{j-1}(\lambda) + \alpha_j \pi_j(\lambda) \ . \tag{10}$$

The scalar α_j can be obtained by expressing the condition that $\phi(\lambda) - \lambda s_j(\lambda)$ is orthogonal to $\lambda\pi_j(\lambda)$, or $\langle \phi(\lambda) - \lambda s_j(\lambda), \lambda\pi_j(\lambda)\rangle_w = 0$, which, with the use of (10), leads to

$$\alpha_j = \frac{\langle \phi(\lambda) - \lambda s_{j-1}(\lambda), \lambda\pi_j(\lambda)\rangle_w}{\langle \lambda\pi_j(\lambda), \lambda\pi_j(\lambda)\rangle_w} \ . \tag{11}$$

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

$$\alpha_j = \frac{\langle \phi(\lambda), \lambda\pi_j(\lambda)\rangle_w}{\langle \lambda\pi_j(\lambda), \lambda\pi_j(\lambda)\rangle_w} \ . \tag{12}$$

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

$$\pi_{j+1}(\lambda) \ = \ -\alpha_j \left[\lambda\pi_j(\lambda) - \eta_j \pi_j(\lambda) - \beta_j \pi_{j-1}(\lambda)\right] \ . \tag{13}$$

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 $i \leq j$. 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 pseudo-residual 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. $\alpha_j := \frac{\langle \phi, \lambda\pi_j\rangle_w}{\langle \lambda\pi_j, \lambda\pi_j\rangle_w}$
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 [10, 19]. The main difference, is that the algorithms in [10, 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 $\phi(\lambda) \equiv 1$, 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 $\langle \phi - \lambda s_j(\lambda), \lambda\pi_i \rangle_w = 0$ for $i \leq j$, so we have

$$\langle (1-\phi) - \rho_{j+1}, \lambda\pi_i \rangle_w \;=\; 0 \;, \quad i \leq j \;.$$

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 $[\phi(A) - As(A)]b$ but this is not practically computable. Therefore, we can generate instead of r_j 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. $\quad \tilde{\alpha}_j := \langle \tilde{\rho}_j, \lambda\tilde{\rho}_j \rangle_w / \langle \lambda\pi_j, \lambda\pi_j \rangle_w$
4. $\quad \alpha_j := \langle \phi, \lambda\pi_j \rangle_w / \langle \lambda\pi_j, \lambda\pi_j \rangle_w$
5. $\quad x_{j+1} := x_j + \alpha_j p_j$
6. $\quad \tilde{r}_{j+1} := \tilde{r}_j - \tilde{\alpha}_j A p_j$ — $\tilde{\rho}_{j+1} = \tilde{\rho}_j - \tilde{\alpha}_j \lambda\pi_j$
7. $\quad \beta_j := \langle \tilde{\rho}_{j+1}, \lambda\tilde{\rho}_{j+1} \rangle_w / \langle \tilde{\rho}_j, \lambda\tilde{\rho}_j \rangle_w$
8. $\quad p_{j+1} := \tilde{r}_{j+1} + \beta_j p_j$ — $\pi_{j+1} := \tilde{\rho}_{j+1} + \beta_j \pi_j$
9. — *Compute* $\lambda\pi_{j+1}$
10. *EndDo*

It is remarkable that the only difference with a generic Conjugate Residual-type algorithm (see, e.g. Algorithm 2.1) is that the updates to x_{j+1} use a different coefficient α_j from the one 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 or may not be necessary to include these computations.

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

$$s_j(\lambda) = \alpha_0\pi_0(\lambda) + \cdots + \alpha_j\pi_j(\lambda) \ . \tag{14}$$

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

$$\langle \lambda\pi_j(\lambda), \lambda\pi_i(\lambda)\rangle_w = \langle \lambda\tilde{\rho}_j(\lambda), \tilde{\rho}_i(\lambda)\rangle_w = 0 \quad \textit{for} \quad i \neq j \ . \tag{15}$$

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

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 (15) is trivially satisfied. The relation (7) 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 $\phi - \lambda s_j(\lambda) = \phi - \sum_{i=0}^{j} \alpha_i \lambda\pi_i(\lambda)$. From the optimality condition, the best polynomial is given when the scalars α_i satisfy the relation $\langle \phi - \lambda s_j(\lambda), \lambda\pi_i(\lambda)\rangle_w = 0$, which, by exploiting (7) and the orthogonality of the system $\{\lambda\pi_i\}_{i=0,\ldots,j}$, yields,

$$\alpha_j = \langle \phi, \lambda\pi_j\rangle_w \ / \ \langle \lambda\pi_j, \lambda\pi_j\rangle_w \ .$$

■

It is worth exploring the formula (12), which defines the scalars α_j, a little further. In the standard CR algorithm, the expression (11) 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 (12) uses the scalars α_i's (formula (14)), 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 $i \leq j-1$. Then, we can rewrite the numerator of (12) as

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

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:

$$\alpha_j = \tilde{\alpha}_j - \frac{\langle 1 - \phi, \lambda\pi_j\rangle_w}{\langle \lambda\pi_j, \lambda\pi_j\rangle_w} \ . \tag{16}$$

The only merit of this expression, as a substitute to (12), is that it clearly establishes Algorithm 2.3 as a 'corrected version' of the standard Algorithm 2.1. In the special situation when $\phi \equiv 1$, $\alpha_i = \tilde{\alpha}_i$, and the two algorithms coincide as expected.

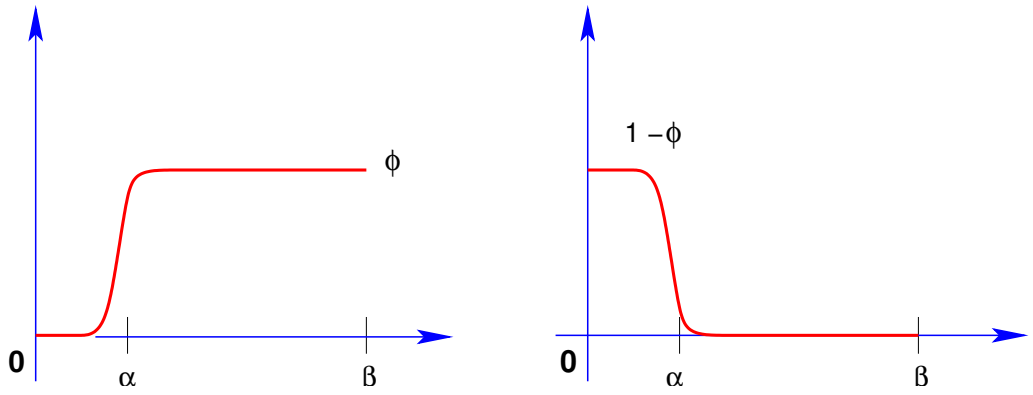

Figure 2: A typical filter function ϕ and its dual filter $\psi \equiv 1 - \phi$

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. As was already mentioned it is generally not a good idea to use as ϕ the step function shown in Figure 1. 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 show in Figure 2.

The filter function in Figure 2 can be a piecewise polynomial consisting of two parts: A function which increases smoothly from zero to one when λ increases from 0 to α, and the constant function unity in the interval $[\alpha, \beta]$. Alternatively, the function can consist of three parts, one on each of the intervals $[0,\ \alpha_0]$, $[\alpha_0,\ \alpha_1]$ and $[\alpha_1,\ \beta]$, with $0 < \alpha_0 < \alpha_1 < \beta$. It will begin with the constant value zero in the interval $[0\ \alpha_0]$, then increase smoothly from 0 to one in a second interval $[\alpha_0\ \alpha_1]$, and finally take the constant value one in $[\alpha_1,\ \beta]$. This second part of the function (the first part for the first scenario) bridges the values zero and one by a smooth function and was termed a "bridge function" in [10]. In what follows we focus on obtaining bridge functions for the generic case, i.e., for an interval $[\alpha_0,\ \alpha_1]$.

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:

$$\begin{aligned} &\Theta_{[m_0,m_1]}(\alpha_0) = 0; \quad \Theta'_{[m_0,m_1]}(\alpha_0) = \cdots = \quad \Theta^{(m_0)}_{[m_0,m_1]}(\alpha_0) = 0 \\ &\Theta_{[m_0,m_1]}(\alpha_1) = 1; \quad \Theta'_{[m_0,m_1]}(\alpha_1) = \cdots = \quad \Theta^{(m_1)}_{[m_0,m_1]}(\alpha_1) = 0. \end{aligned} \tag{17}$$

Thus, $\Theta_{[m_0,m_1]}$ has degree $m_0 + m_1 + 1$ and m_0, m_1 define the degree of smoothness at the points 0 and α 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 translate everything for the variable in the interval $[-1, 1]$, and shift down the function by $1/2$. If the corresponding function is denoted

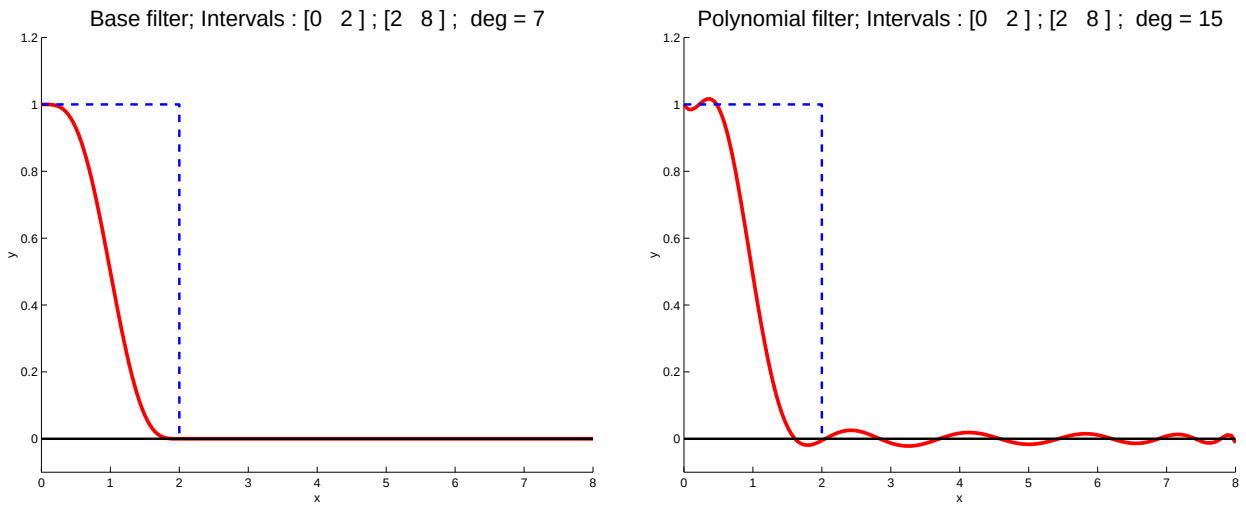

Figure 3: Left: Dual base filter ψ defined on two intervals: $1 - \Theta_{[4,4]}$ in [0, 2] and zero in [2, 8]; Right: its polynomial approximation of degree 15.

by η, then the above conditions become

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

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 $\eta(t)$:

$$\eta(t) = -\frac{1}{2} \ + \ \frac{\int_{-1}^{t} (1-s)^{m_1}(1+s)^{m_0}\ ds}{\int_{-1}^{1} (1-s)^{m_1}(1+s)^{m_0}\ ds} \tag{18}$$

The first and second derivatives of η are

$$\eta'(t) = \frac{(1-t)^{m_1}(1+t)^{m_0}}{\int_{-1}^{1} (1-s)^{m_1}(1+s)^{m_0}\ ds}\ ; \qquad \eta''(t) = \left[\frac{m_0}{1+t} - \frac{m_1}{1-t}\right] \eta'(t)\ . \tag{19}$$

So there is an inflexion point at

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

Since the maximum value of the derivative is required for the convergence analysis, it will be useful to determine it. The derivative increases from its value at the point -1, to a certain peak, reached at the inflexion point and then it decreases 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 2.1 *The maximum value of the derivative of the function η in the interval $[-1, 1]$ is given by*

$$\eta'_{max} = \frac{m_0 + m_1 + 1}{2}\ \frac{m_1^{m_1}\ m_0^{m_0}}{(m_0+m_1)^{m_0+m_1} \times \binom{m_0}{m_0+m_1}}\ . \tag{20}$$

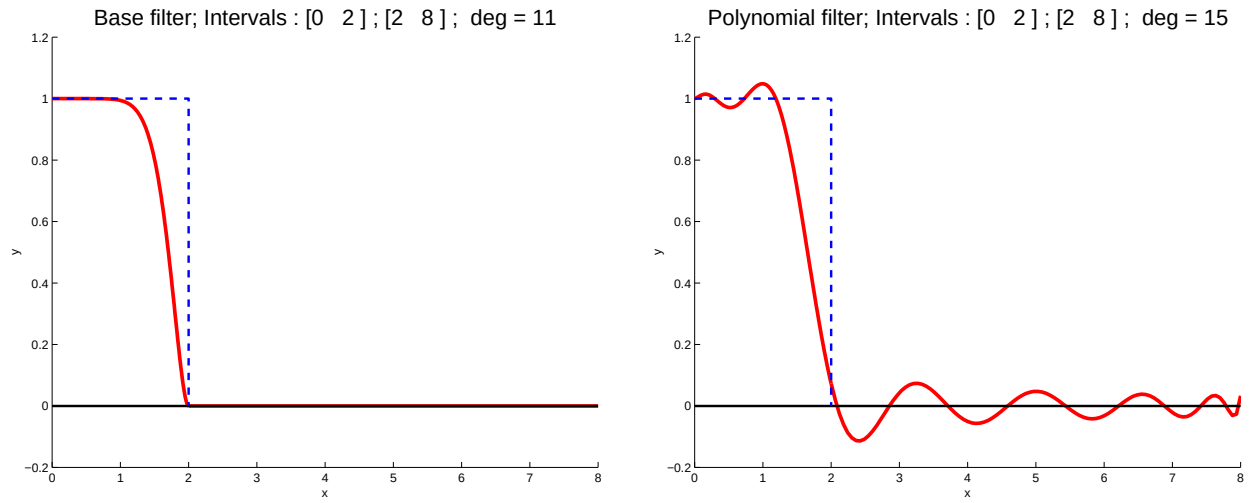

Figure 4: Left: Dual base filter ψ defined on two intervals: $1 - \Theta_{[10,2]}$ in $[0,\ 2]$ and zero in $[2,\ 8]$; Right: its polynomial approximation of degree 15.

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

$$\eta'_{max} \approx \frac{m_0 + m_1}{2\sqrt{2\pi}} \sqrt{\frac{1}{m_0} + \frac{1}{m_1}} \ . \tag{21}$$

Proof. The integral in the denominator of η in (19) 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 derivative η' 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 (21) after simplifications. ∎

This result must now be translated into the original interval $[\alpha_0,\ \alpha_1]$. The function Θ (indices m_0, m_1 are omitted) and its derivative in terms of η and η' are

$$\Theta(\lambda) = \frac{1}{2} + \eta\left(2\frac{\lambda-\alpha_0}{\alpha_1-\alpha_0} - 1\right) \quad \Theta'(\lambda) = \frac{2}{\alpha_1-\alpha_0}\eta'\left(2\frac{\lambda-\alpha_0}{\alpha_1-\alpha_0} - 1\right) \ .$$

and so,

$$\Theta'_{max} = \frac{2}{\alpha_1 - \alpha_0}\eta'_{max} \ .$$

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

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

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 $m_0 = m_1 = 3$:

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

As was seen, the ratio $\frac{m_1}{m_0}$ determines the localization of the inflexion point. Making the polynomial increase rapidly from 0 to 1 in a small interval, can be achieved 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 (dual) filter functions are shown in Figures 3 and 4. A third example shows a situation where 3 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 $1 - \Theta_{[5,5]}$ in [1.72.3]. This time we plot a higher degree polynomial approximation to ψ 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 polynomials using Legendre weights in each interval instead of Chebyshev weights and, in all cases, there was no significant difference.

2.6 The weight function w

Denoting the l sub-intervals of $[0, \ \beta]$, by $[\alpha_{i-1}, \alpha_i]$ $i = 1, \ldots, l$, we define the inner-product on each sub-interval (α_{l-1}, α_l) using Chebyshev weights,

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

Then the inner product on the interval $[0, \beta] \equiv [\alpha_0, \ \alpha_1] \cup [\alpha_1, \ \alpha_2] \cdots \cup [\alpha_{l-1}, \ \alpha_l]$, is defined as a weighted sum of the inner products on the smaller intervals,

$$\langle \psi_1, \psi_2 \rangle_w = \sum_{i=1}^{l} \ \mu_i \int_{\alpha_{i-1}}^{\alpha_i} \frac{\psi_1(t)\psi_2(t)}{\sqrt{(t - \alpha_{i-1})(\alpha_i - t)}} \, dt. \tag{22}$$

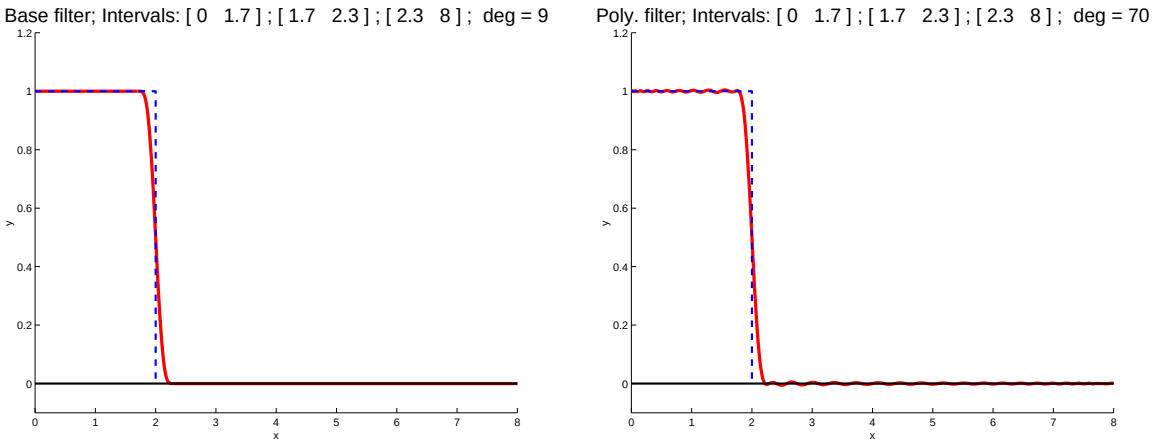

Figure 5: Left: Dual base filter ψ defined on three intervals: 1 in [0, 1.7], $1 - \Theta_{[5,5]}$ in [1.7, 2.3] and 0 in [2.3, 8]; Right: its polynomial approximation of degree 70.

For example for two intervals the weight function is defined as

$$\langle \psi_1, \psi_2 \rangle_w = \mu_1 \int_0^\alpha \frac{\psi_1(t)\psi_2(t)}{\sqrt{t(\alpha - t)}}\, dt + \mu_2 \int_a^\beta \frac{\psi_1(t)\psi_2(t)}{\sqrt{(t-\alpha)(\beta - t)}}\, dt. \tag{23}$$

The μ_i's can be chosen to emphasize or de-emphasize specific sub-intervals. In most of our tests we took the μ_i to be either equal to the constant one 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 [11] 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 $[\alpha_{l-1}, \alpha_l]$ into $[-1, 1]$:

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

Denote by C_i the i-th 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 $\left(C_k^{(l)}\right)_{k\in\mathbb{N}}$ constitute an orthogonal basis since,

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

As a result, if two polynomials ψ_1, ψ_2 are expanded in the above Chebyshev bases for each interval, the inner products (22) 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). Multiplying a polynomial ψ expanded in the Chebyshev bases by the variable λ, can be easily done by exploiting the following relations:

$$\begin{aligned}
\lambda\, C_i^{(l)}(\lambda) &= \frac{\alpha_l - \alpha_{l-1}}{4} C_{i+1}^{(l)}(\lambda) + \frac{\alpha_l + \alpha_{l-1}}{2} C_i^{(l)}(\lambda) + \frac{\alpha_l - \alpha_{l-1}}{4} C_{i-1}^{(l)}(\lambda) \quad i \geq 1 \\
\lambda\, C_0^{(l)}(\lambda) &= \frac{\alpha_l - \alpha_{l-1}}{2} C_1^{(l)}(\lambda) + \frac{\alpha_l + \alpha_{l-1}}{2} C_0^{(l)}(\lambda)\ .
\end{aligned}$$

These 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 $1-\phi$. Convergence results of this type utilize uniform norm results. We will restrict ourselves to a simple result derived from the Jackson theorems, see [8]. 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 [8]. In particular this means that $\lim_{n\rightarrow\infty} E_n(f) = 0$. 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

$$\omega_f(\delta) = \sup_{|t_1 - t_2|\ \leq \delta} |f(t_1) - f(t_2)|\ . \tag{24}$$

Theorem 2.1 (Jackson's theorem III) *For all functions* $f\ \in C[0\ 2\pi]$,

$$E_n(f) \leq \omega_f\left(\frac{\pi}{n+1}\right)\ . \tag{25}$$

See [8] for proofs and additional details. For an arbitrary interval $[\alpha, \beta]$ the above theorem translates into

$$E_n(f, [\alpha, \beta]) \leq \omega_f\left(\frac{\beta - \alpha}{2(n+1)}\right)\ . \tag{26}$$

Applying the above result to base filter functions is easy.

Lemma 2.2 *Let the base filter function ϕ be the spline function constructed as*

$$\phi(t) = \begin{cases} 0 & for \quad t \in [0, \alpha_0) \\ \Theta_{[m_0,m_1]} & for \quad t \in [\alpha_0, \alpha_1) \\ 1 & for \quad t \in [\alpha_1, \beta] \end{cases} .$$

Then,

$$\omega_\phi(\delta) \leq \frac{2\eta'_{max}}{\alpha_1 - \alpha_0}\delta,$$

where η'_{max} is given by (20), and is approximated by (21) for large values of m_0, m_1.

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

Proposition 2.3 *Let ϕ the base filter function defined in Lemma 2.2. Then,*

$$E_n(\phi) \ \leq \frac{\beta \ \eta'_{max}}{(n+1)(\alpha_1 - \alpha_0)} \ . \tag{27}$$

where η'_{max} is given by (20), and is approximated by (21) 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 one in (22) then $K = \sqrt{l \ \pi}$.

2.8 Hybrid dot products

Returning to Section 2.2, we recall that the inner product $\langle ., . \rangle_w$ can be essentially any (non-degenerate) L_2 dot product, whether discrete or continuous. As is well-known, in the Hermitian case, standard Krylov subspace methods such as the CG algorithm, amount to minimizing a certain discrete norm of the error or the residual vector, taken in the eigenbasis. In contrast, methods such as the Chebyshev algorithm use a purely continuous weight function to achieve a certain minimization of the error or the residual. It is also possible to have a hybrid method which mixes the two weights though this does not seem to have been considered so far in the literature.

We now consider a filtering technique which works by altering the discrete inner product used by these algorithms. Specifically, we can consider mixing a discrete and continuous inner products by defining

$$\langle p, q \rangle_w = (1 - \mu) \ (p(A)r_0, q(A)r_0) + \mu \ \int_\alpha^\beta p(\lambda)q(\lambda)w(\lambda)d\lambda \ . \tag{28}$$

Although not clear from the above definition it is possible to filter a solution by carefully selecting the continuous weigtht function. The rationale for this choice is that we sometimes need the procedure to be biased toward the eigenvalues in the interval $[\alpha, \ \beta]$ without sacrificing completely the accuracy in the interval $[0, \ \alpha]$. When $\mu = 0$ we recover the usual CR

algorithm, while $\mu = 1$ yields a pure (continuous) least-squares approach which will tend to make residual components small in the interval $[\alpha,\ \beta]$.

That the above constitutes a non-degenerate inner product "in general" is a consequence of the fact that, under some mild conditions on r_0 and the degree j, both parts of (28) are proper inner products, when considered individually.

3 Applications and extensions

Polynomial filtering has many applications in numerical linear algebra and related areas. In fact, we may state that the number of these applications is likely to increase because of the growing need to solve regularized least-squares problems as well as to apply various forms of Principal Component Analysis. 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 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 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 [2]. 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.

In this section we show how polynomial filtering can be used to compute very large invariant subspaces of symmetric real (or Hermitian complex) matrices. Specifically, the following problem is addressed: *Compute an orthonormal basis of the invariant subspace associated with all eigenvalues below a bound* α. It is assumed that an upper bound β for the spectrum is available, and, without loss of generality, that all eigenvalues are ≥ 0.

The simplest 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 $q(\lambda)$ is large for $0 \leq \lambda \leq \alpha$ and small for $\alpha < \lambda \leq \beta$. 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 will be required because the Lanczos algorithm will converge faster, and this will lead to a much lower cost due to much less expensive orthogonalization steps. In case of large k, the additional cost

of matrix vector products (now replaced by a product with $q(A)$ at each step) is outweighed by the gain from these other computations.

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. If in contrast only 200 vectors are required, 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 counter argument to this is that a much smaller basis can also be used for the restarted Lanczos method. Though this situation is much harder to analyze, it is important to realize that restarting is expensive because eigenvectors are repeatedly (implicitly) computed. It is not the goal of this paper to compare these approaches. This will be done in a forthcoming article where these comparisons will be undertaken for realistic problems arising from electronic structures calculations [3].

The procedure just described 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 $\lambda_i > \alpha$ then since $q(\lambda_i)$ is close to zero, then 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 leads to substantial savings in time when compared with standard approaches.

3.2 Computing $f(A)v$

The procedures described earlier compute approximations to $\phi(A)v$ where ϕ is a specific spline function on two or three intervals. There is of course no reason why one should be limited a to spline function which approximates a step function. In fact the approach can extended to many 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., [28, 23, 4, 16, 15], though the attention was primarily focussed on the the case when f is analytic (e.g., $f(t) = \exp(t)$). Problems which involve non continous 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, although we do not know of specific practical application where general splines other than the ones used in this paper are required. However, 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 polynonial approximation. The framework developed in this paper is ideally suited for handling this situation. The only extensions required are that the filter has several intervals now instead of 2 or three, and the polynomials in each interval are those of the spline function.

Another interesting application is when ϕ is the sign function. Computing the sign function of a matrix has important applications in QMC (quantum chromo dynamics), see, e.g. [12]. In order to solve the linear system associated with the matrix $s(A)$, we can use the same approach as that of the regularized solutions, except that the base filter is now

a function which approximates the sign function instead of the step function. In this case it is important to use 3 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_+.

3.3 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 [13]. However, for large matrices this is not always practically feasible or it may be too expensive.

It is sometines useful to obtain just 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 $[0, \ \alpha]$ must be found which contains these k eigenvalues. A guess for α can be given and then refined by answering the question: How many eigenvalues are located on the left of α?

An easy solution to this can be given by the Lanczos procedure. One can simply run the Lanczos algorithm with partial reorthogonalization and record the number n_α of all eigenvalues below α of the tridiagonal matrix T_m obtained from the Lanczos algorithm. When this number stabilizes (i.e., all eigenvalues below α converge) then n_α will represent the desired number. The problem with this approach is that it may be very expensive when the number n_α is large.

A rough approximation of n_α 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, 20, 1]. Consider a low-pass polynomial filter such as the one shown in Figure 5, and an arbitrary vector v of 2-norm unity. Expand the vector v in the eigenbasis as,

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

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

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

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

$$n_\alpha \approx n \times (v, p(A)v) \ . \tag{29}$$

Of course, a unique sample may not be good enough and several trials should be taken and averaged in some way. The numerical experiments sections explores 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_α 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 will be reported in a forthcoming article [3]. The goals of the tests discussed in this section are (1) to examine the convergence of the process; (2) to show and compare the techniques discussed earlier for the problem of obtaining regularized solutions of linear systems; and (3) to demonstrate the use of polynomial filtering for approximating inertia of shifted matrices (see Section 3.3).

All tests were performed with matlab version 6.5 on a Linux computer (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 Laplacean using centered differences on a 20×15 mesh. We then compute the vector v which has all components equal to one in the eigenbasis, i.e., v is the sum of all the (normalized) eigenvectors. This vector is then filtered with a chosen low-pass base filter $\psi = 1 - \phi$ and we plot $\|\phi(A)v - As_{k-1}(A)v\|_2$ for $k = 1, \ldots, 200$. This is referred to as the 'filtered residual'. The low-pass filter $\psi = 1 - \phi$ is selected as follows:

$$\psi(t) = \begin{cases} 1 & \text{for} \quad t \in [0, \alpha_0) \\ 1 - \Theta_{[m_0, m_1]} & \text{for} \quad t \in [\alpha_0, \alpha_1) \\ 0 & \text{for} \quad t \in [\alpha_1, \beta] \ . \end{cases} \tag{30}$$

A first run used the values : $m_0 = m_1 = 10$, $\beta = 8$, $\alpha_0 = 1.9, \alpha_1 = 2.1$ and the second used the same values for m_0, m_1, and β, and changed α_0, α_1 to $\alpha_0 = 1.8, \alpha_1 = 2.2$. 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 (27) corresponding to the first test case ($m_0 = m_1 = 10$, $\beta = 8$, $\alpha_0 = 1.9$). Here, η'_{max} is estimated by (21), 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 are important to make. The first is that for the second run, the behavior is not at all predicted by the bounds. It has an exponential character which is not at all reflected by 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.

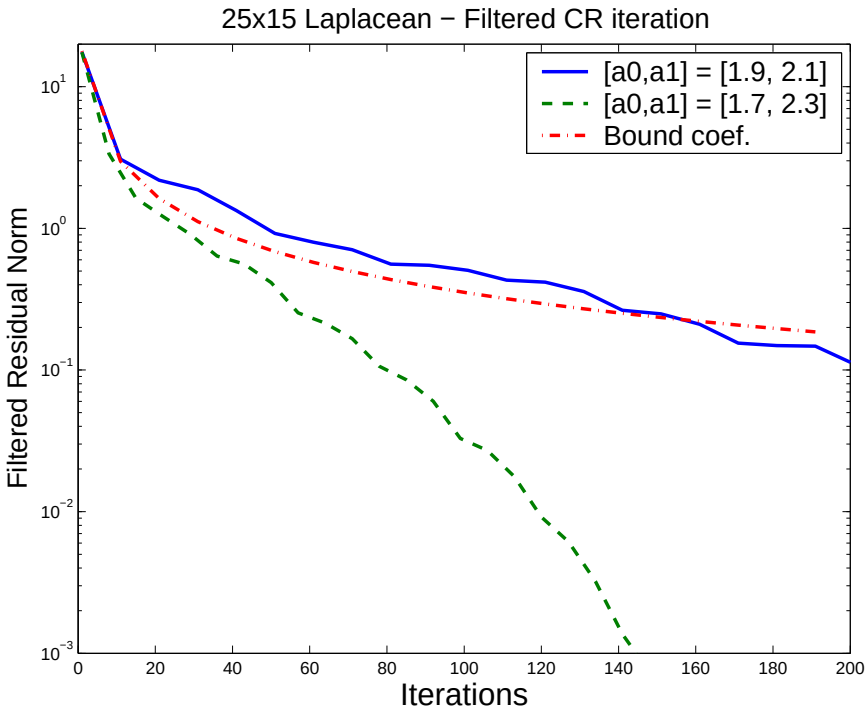

Figure 6: Convergence of filtered Polynomial CR algorithm for two different cases and comparison with the coefficient of the bound (27).

4.2 Tests with regularization

In this test we construct a linear system that is ill-conditioned and whose exact solution is known. We then perturb the right-hand side and solve the linear system with five techniques and compare the resulting norms of the errors. The matrix is generated from a discretization of a Laplacean on a 35×45 rectangular mesh. This matrix, call it B, is then shifted and squared to yield the desired ill-conditioned coefficient matrix A:

$$A = (B - 0.01)^2 \ .$$

The smallest eigenvalue of B is ≈ 0.0123, so the the smallest eigenvalue of A is close to $0.0023^2 = 5.167e-06$. Since the largest eigenvalue is close to 63.64 this yields a condition number for A of $\approx 1.2e+07$.

The right-hand side is generated so that the solution is known. Specifically the solution is taken to be the discretization of the function $f(x_1, x_2) = x_1(1-x_1)x_2(1-x_2)$. The discrete

version of the above exact solution is normalized to have inf-norm equal to one. The resulting vector is denoted by x_*. This vector is then multiplied by A to obtain an unperturbed right-hand side b_*. This right hand-side is now perturbed by normally distributed pseudo-random numbers multiplied by 0.05 (using the mat lab function `randn(n,1)`). This is a relatively large perturbation – the largest entry of the perturbing vector is about 0.05*2.7= 0.135. The exact solution of the system will be dominated by errors because A is quite ill-conditioned. Solving the system exactly yielded an error norm of $\|x - A^{-1}b\|_2 = 18.67$.

Figure 7 shows the behavior of $\|x_k - x_*\|_\infty$ for 5 different iterative schemes. Here k denotes the iteration number and x_* is the original solution of the unperturbed system. A few details are given for the methods used in this test. The simplest methods to use are simply the Conjugate Gradient or the Conjugate Residual algorithms. For perturbed and ill-conditioned systems, it is known that these algorithms should not be iterated to convergence, since this would otherwise introduce the part of the solution in the subspace associated with the smallest eigenvalues, which leads to amplifying the noise. This is verified in the plots of the CG and CR error curves. One remedy is to use Tychonov regularization as discussed in the introduction. In Tychonov regularization, the CG algorithm is used to solve the system $(A^2 + \sigma I)x = Ab$. We have selected to take $\sigma = 0.1$. The CR filtered polynomial approach is based on two intervals, with the function Θ determined by $m_0 = 5, m_1 = 10$ on the first interval.

The method labeled Hybrid uses hybrid inner products as described in Section 2.8. The coefficient μ used in the inner product (28) had to be set relatively large to obtain a behavior that shows a noticeable difference with the standard CG. Specifically, we took $\mu = 1.e + 08$. So when norms are considered, then the continuous part weighs 4 orders of magnitude more than the discrete part. This is to be expected because the first part is not scaled by the number of points, which is $n = 1,575$.

Notice that all methods except the ones based on polynomial filtering and Tychonov regularization end up with increasing values of the error norm after a certain iteration. For CG the errors jump back rather steeply after the bottom, more so than with CR. This underscores the difficulty in trying to use the standard CG (or CR) algorithms, but exploit an early termination. The Hybrid algorithm seems to be a compromise between the regularized approaches (filtered iterations and Tychonov) and the non-regularized iterations (CG and CR). Tychonov and filtered CR reach about the same level of error, though the error with the filtered CR algorithm decreases more rapidly at the start. Interestingly, the minimum error reached during all steps for each of the 5 methods is quite close: they are all in the range 0.77 – 0.79.

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 3.3 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 $nnz = 6,511$ entries. The sparsity pattern of the matrix is shown on the left side of Figure 8. This matrix has all its eigenvalues

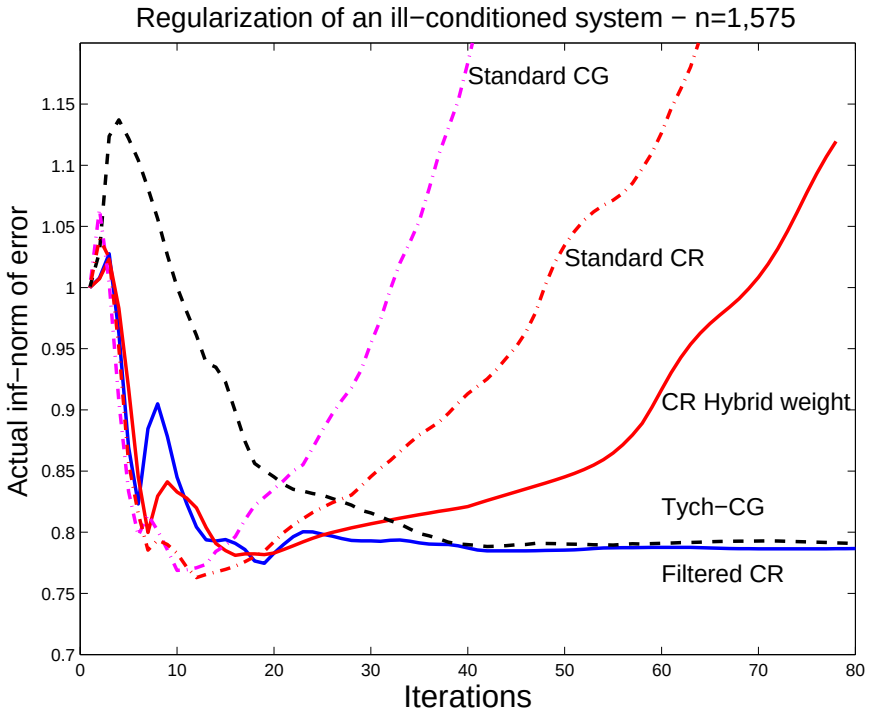

Figure 7: Behavior of five different techniques for solving a perturbed ill-conditioned linear system.

in the interval [-3.117.., 5.971..]. 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 3 intervals, defined as in Equation (30), was used with the parameters: $m_0 = m_1 = 10$. The interval bounds given were $0, \alpha_0 = 3.15, \alpha_1 = 3.25, \beta = 6$. The degree of the polynomial used was $m = 20$.

The right side of Figure 8 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 (29) 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.

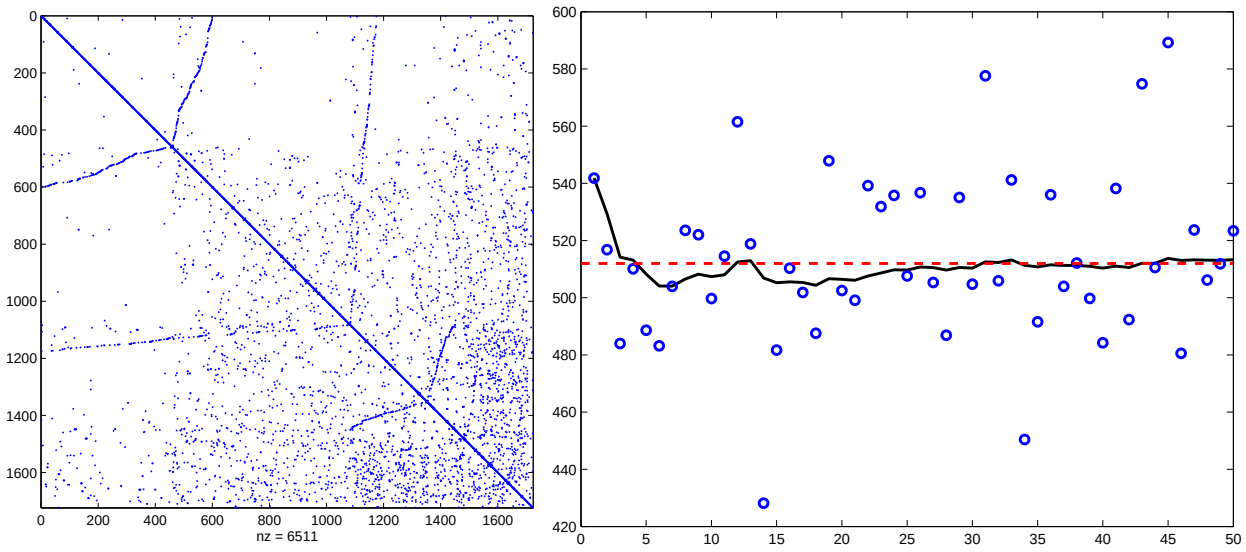

Figure 8: Pattern of matrix `bcspwr09` (Left) and stochastic estimate of its number of negative eigenvalues (right)

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 the 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. This will be reported elsewhere [3] along with a software release.

There are many other potential applications of polynomial filtering in numerical linear algebra beyond those discussed here. Many computations require the solution of least-squares systems with regularization. In some cases these problems come with a set of constraints and it would be quite useful to extend the ideas of this paper to this situation. As an example, a primary technique used in adaptive airborne radar and called Space-Time Adaptive Processing (STAP), finds weights which minimize $\|Xw\|_2$ subject to the constraint $S_T^T w = 1$, where both X (space-time data snapshot) and S_T (space-time steering vector) are given [7, 21, 6]. The problem is to solve this constrained least-squares system in the subspace corresponding to the largest singular values (called the 'interference space'). The techniques described in this paper can be extended to handle constraints by simply using a penalty technique.

References

[1] C. Bekas, E. Kokiopoulou, and Y. Saad. An estimator for the diagonal of a matrix. Technical Report umsi-2005-xxx, Minnesota Supercomputer Institute, University of Minnesota, Minneapolis, MN, 2005.

[2] C. Bekas, Y. Saad, M. L. Tiago, and J. R. Chelikowsky. Computing charge densities with partially reorthogonalized lanczos. Technical Report umsi-2005-029, Minnesota Supercomputer Institute, University of Minnesota, Minneapolis, MN, 2005.

[3] K. Bekas, E. Kokiopoulou, and Y. Saad. Polynomial filtered lanczos iterations in density functional theory. Technical report, Minnesota Supercomputer Institute, University of Minnesota, Minneapolis, MN, 2005. In preparation.

[4] L. Bergamaschi, M. Caliari, and M. Vianello. Efficient computation of the exponential operator for discrete 2d advection-diffusion equations. *Numer. Lin. Alg. Appl.*, 10(3):271–289, 2003.

[5] M. Berry and M. Browne. *Understanding search engines*. SIAM, 1999.

[6] Bojanczyk, A. W., and Lebak J. Design and perofrmance evaluation of a portable parallel library for STAP. *IEEE Transactions on Parallel and Distributed Systems*, 2000.

[7] L. E. Brennan and I. S. Reed. Theory of adaptive radar. *IEEE trans. AES*, 9(2):237–252, 1973.

[8] C. C. Cheney. *Introduction to Approximation Theory*. McGraw Hill, NY, 1966.

[9] I. S. Duff, R. G. Grimes, and J. G. Lewis. Sparse matrix test problems. *ACM Transactions on Mathematical Software*, 15:1–14, 1989.

[10] 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.

[11] B. Fischer and G. H. Golub. How to generate unknown orthogonal polynomials out of known orthogonal polynomials. *Journal of Computational and Applied Mathematics*, 43:99–115, 1992.

[12] A. Frommer, T. Lippert, B. Medeke, and K. Shilings. *Numerical challenges in Lattice Quantum Chromodynamics*. Springer Verlag, Berlin, 1999. Lectures Notes in Computational Science and Engineering, vol. 15.

[13] G. H. Golub and C. Van Loan. *Matrix Computations, 3rd edn.* The John Hopkins University Press, Baltimore, 1996.

[14] M. Hanke. *Conjugate gradient type methods for ill-posed problems*. Longman Scientific & Technical, Harlow, 1995.

[15] M. Hochbruck and C. Lubich. On Krylov subspace approximations to the matrix exponential operator. *SIAM Journal on Numerical Analysis*, 34:1911–1925, 1997.

[16] M. Hochbruck, C. Lubich, and H. Selhofer. Exponential integrators for large systems of differential equations. *SIAM Journal on Scientific Computing*, 19:1552–1574, 1998.

[17] M. F. Hutchinson. A stochastic estimator of the trace of the influence matrix for Laplacian smoothing splines. *Commun. Statist. Simula.*, 19(2):433–450, 1990.

[18] E. Kokiopoulou and Y. Saad. PCA and kernel PCA using polynomial filtering: a case study on face recognition. Technical Report umsi-2004-213, Minnesota Supercomputer Institute, University of Minnesota, Minneapolis, MN, 2004. submitted.

[19] E. Kokiopoulou and Y. Saad. Polynomial Filtering in Latent Semantic Indexing for Information Retrieval. In *ACM-SIGIR Conference on research and development in information retrieval*, Sheffield, UK, July 25th-29th 2004.

[20] G. A. Parket, W. Zhu, Y. Huang, D. K. Hoffman, and D. J. Kouri. Matrix pseudospectroscopy: iterative calculation of matrix eigenvalues and eigenvectors of large matrices using a polynomial expansion of the dirac delta function. *Comp. Phys. Comm.*, 96:27–35, 1996.

[21] I. S. Reed, J. D. Mallet, and L. E. Brennan. Rapid convergence rate in adaptive arrays. *IEEE trans. AES*, 10(6):853–863, 1974.

[22] Y. Saad. Iterative solution of indefinite symmetric systems by methods using orthogonal polynomials over two disjoint intervals. *SIAM Journal on Numerical Analysis*, 20:784–811, 1983.

[23] Y. Saad. Analysis of some Krylov subspace approximations to the matrix exponential operator. *SIAM Journal on Numerical Analysis*, 29:209–228, 1992.

[24] Y. Saad. *Iterative Methods for Sparse Linear Systems, 2nd edition*. SIAM, Philadelpha, PA, 2003.

[25] Y. Saad and M. H. Schultz. GMRES: a generalized minimal residual algorithm for solving nonsymmetric linear systems. *SIAM Journal on Scientific and Statistical Computing*, 7:856–869, 1986.

[26] A. N. Tikhonov. Regularisation of incorrectly posed problems. *Soviet. Math. Dokl.*, 4:1624–1627, 1963.

[27] A. N. Tikhonov. Solution of incorrectly formulated problems and the regularisation method. *Soviet. Math. Dokl.*, 4:1036–1038, 1963.

[28] 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. Comp. and Appl. Math.*, 18:249–263, 1987.