INTERNATIONAL JOURNAL OF
NUMERICAL ANALYSIS AND MODELING
Volume 18, Number 5, Pages 712–721

EXTRAPOLATING THE ARNOLDI ALGORITHM TO IMPROVE EIGENVECTOR CONVERGENCE

SARA POLLOCK AND L. RIDGWAY SCOTT

Abstract. We consider extrapolation of the Arnoldi algorithm to accelerate computation of the dominant eigenvalue/eigenvector pair. The basic algorithm uses sequences of Krylov vectors to form a small eigenproblem which is solved exactly. The two dominant eigenvectors output from consecutive Arnoldi steps are then recombined to form an extrapolated iterate, and this accelerated iterate is used to restart the next Arnoldi process. We present numerical results testing the algorithm on a variety of cases and find on most examples it substantially improves the performance of restarted Arnoldi. The extrapolation is a simple post-processing step which has minimal computational cost.

Key words. Eigenvalue computation, extrapolation, Arnoldi algorithm.

1. Introduction

There are many applications in which the smallest eigenvalues of large systems must be computed, e.g., in stability analysis of numerical schemes [6] and in stability analysis of partial differential equations [16]. The (inverse) power method is often preferred due to its ease of implementation and the limited amount of storage required. In many cases [16], all that is required to implement the inverse power method is to solve the associated system of equations repeatedly. Thus it is easy to modify a code for a solver to become a code for the inverse power method.

Extrapolation has been shown [11] to provide an effective way to improve the power method. Here we consider a different approach in which the power method is first generalized. We then examine extrapolation of the method.

One generalization of the power method is to use a small number $k > 1$ of approximation vectors and to project the eigenproblem onto the corresponding k-dimensional space. One then solves the projected k-dimensional eigenproblem and extracts the eigenvector corresponding to the extreme eigenvalue as the next iterate. This can lead to faster convergence with a controlled increase in storage. A natural set of vectors to use is a Krylov basis, and we dub this approach the k-step Krylov method. We show that the popular LOBPCG method is of this form. There could of course be other ways of generating appropriate vectors at each step, e.g., a random process.

We show that the Arnoldi algorithm provides a very stable implementation of the k-step Krylov method. We further demonstrate how a simple extrapolation technique, which takes a combination of the two latest Arnoldi outputs as the next approximation, can be used to further enhance the rate of convergence. Since the basic k-step method is Arnoldi, it is remarkable that this algorithm can be improved by extrapolation.

Received by the editors January 28, 2021 and, in revised form, July 5, 2021.
2000 *Mathematics Subject Classification.* 65B05,65F15.

2. k-step Krylov methods

We can define general k-step Krylov methods as follows. We start with a vector $\mathbf{y}_1$ and define $\mathbf{y}_{j+1} = A\mathbf{y}_j$, for $j = 1, \ldots, k-1$. We seek to find coefficients $a_1, \ldots, a_k$ such that

$$A \sum_j a_j \mathbf{y}_j = \lambda \sum_j a_j \mathbf{y}_j. \tag{1}$$

Taking dot products, we see that this is equivalent to

$$K\mathbf{a} = \lambda M\mathbf{a},$$

where

$$K_{ij} = \mathbf{y}_i^t A\mathbf{y}_j, \qquad M_{ij} = \mathbf{y}_i^t \mathbf{y}_j.$$

Thus we can determine $\mathbf{a}$ by solving the eigenproblem

$$M^{-1}K\mathbf{a} = \lambda\mathbf{a}. \tag{2}$$

Due to the close connection with Ritz methods, $\mathbf{a}$ is called the Ritz vector and λ is the Ritz value. A great deal is known about how these approximate eigenvectors and eigenvalues for A as k increases [14].

We define the output of the k-step method as $\lambda_1, \ldots \lambda_m$ $(m < k)$ and

$$\mathbf{y} = \sum_{i=1}^{k} a_i \mathbf{y}_i,$$

where $\mathbf{a}$ is the eigenvector corresponding to the extreme eigenvalue λ_1 for (2), and $\lambda_2, \ldots \lambda_m$ are the remaining eigenvalues in descending order.

The eigenproblem (2) can be solved analytically for $k \leq 4$. In [13], the case $k = 2$ is described in detail.

2.1. Orthogonalization of Krylov vectors. Unfortunately, the naive approach (2) to the k-step method fails for larger k, due to ill conditioning, as described in [13]. Thus we consider orthogonalization of the Krylov vectors.

Now we modify the steps leading to (2). We start with a vector $\hat{\mathbf{y}}_1$ and define Krylov vectors $\hat{\mathbf{y}}_{j+1} = A\hat{\mathbf{y}}_j$, for $j = 1, \ldots, k$. Then we orthogonalize to get $\mathbf{y}_1, \ldots, \mathbf{y}_k$ by the modified Gram–Schmidt algorithm [2, 12]. We provide the details in [13]. In this setting, the matrix M is the identity.

The use of orthogonal Krylov vectors allows extension to more steps k, but for slightly larger k the algorithm still fails due to the increasing condition number of K, as indicated in [13].

2.2. Arnoldi algorithm. The Arnoldi algorithm makes a small change in the order of orthogonalization and multiplication by the matrix A. Instead of first creating the Krylov vectors all at once, we multiply by A only after orthogonalization. Thus $\mathbf{y}_1 = \|\hat{\mathbf{y}}_1\|^{-1}\hat{\mathbf{y}}_1$. Then for $n = 1, 2, \ldots, k-1$, define

$$\begin{gathered} \widetilde{\mathbf{y}}_n = A\mathbf{y}_n - \sum_{j=1}^{n} h_{j,n}\mathbf{y}_j\,, \qquad h_{j,n} = \mathbf{y}_j^{\,t} A\mathbf{y}_n\,, \quad j = 1, \ldots n, \\ h_{n+1,n} = \|\widetilde{\mathbf{y}}_n\|, \qquad \mathbf{y}_{n+1} = h_{n+1,n}^{-1}\widetilde{\mathbf{y}}_n. \end{gathered} \tag{3}$$

For $n = k$, we compute $h_{j,k} = \mathbf{y}_j^{\,t} A\mathbf{y}_k$ for $j = 1, \ldots k$, but we do not perform the orthogonalization steps in the first line of (3) for $n = k$.

One can show by induction that the vectors $\mathbf{y}_j$ are orthogonal, so that, in exact arithmetic, $H = K$. But this subtle change makes the algorithm far more robust, as shown in Figure 1(a). The k-step algorithm approximates accurately many of the

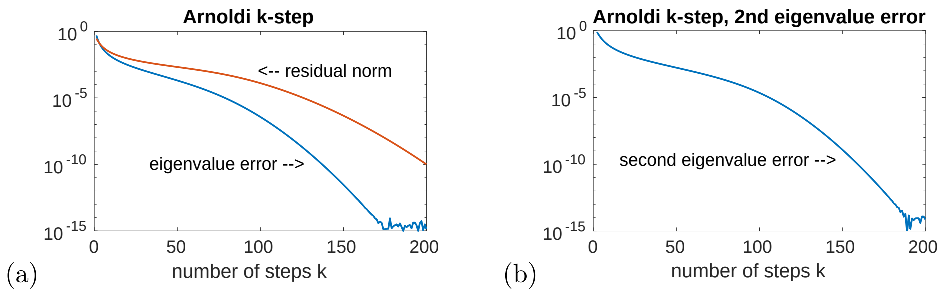

FIGURE 1. The Arnoldi k-step method for the $n \times n$ diagonal matrix A where $a_{ii} = n/i$, $i = 1, \ldots, n$, for $n = 1000$. (a) Errors for the largest eigenvalue (which is 1) and corresponding eigenvector residual norm. (b) Errors for the second-largest eigenvalue (which is $1 - 1/n$).

largest eigenvalues. Figure 1(b) plots the error for the second-largest eigenvalue. Figure 1(a) further displays the expected [3] behavior that the eigenvalue error is proportional to the square of the eigenvector error.

2.3. Restarted Arnoldi algorithm. Our k-step algorithm described above is a restarted Arnoldi algorithm [17]. What is different is that here we propose the restart for purely algorithmic purposes (e.g., minimizing storage) as opposed to deflation or stability considerations [10].

2.4. Rationale for k-step algorithm. Assessments of efficiency are problem dependent, and so we look in detail at one application. Suppose that the matrix A is sparse and of size $N \times N$. We quantify the sparseness by assuming that a matrix-vector multiplication $A\mathbf{y}$ costs $N\varkappa$ operations. Vector dot products cost N operations. Thus the Arnoldi method requires

$$kN\varkappa + k^2N$$

operations. In many cases, multiplication by A actually requires solving a system of equations [16], and so $\varkappa$ might be as large as 1000 or larger, and perhaps increasing as N increases for solvers that are not optimal order. And in many physical applications, only a few eigenvalues are of interest.

One limit on k is the required kN storage of the Krylov vectors. Thus there may be a need for several iterations of a k-step method for a fixed k, even though one iteration for a much larger k might be more efficient.

The restarted algorithm requires solving a $k \times k$ eigenproblem at each step. Let us assume that the reduced eigenproblem takes on the order of k^3 operations. Then if $k << \min\{\varkappa, N\}$, the cost of the reduced eigensolve is negligible.

3. Accelerating the k-step method with extrapolation

A depth-1 extrapolation can be applied to the k-step or restarted Arnoldi method as a simple low-cost post-processing procedure. As discussed and shown below, this can be advantageous particularly for smaller values of k. Here we discuss the extrapolation with iteration j parameter γ_j as summarized in the following algorithm. In order for the first two approximate eigenvalues $\lambda_1^{(j)}$ and $\lambda_2^{(j)}$ to provide meaningful information, generally $k \geq 4$ makes sense. However, if the second approximate eigenvalue is not being used to define the extrapolation parameter γ_j, then $k \geq 2$ makes sense.

Algorithm 3.1 (Extrapolated k-step Arnoldi)**.** *Choose $y^{(0)}$ and $k \geq 2$.*

```
Compute $[y^{(1)}, \lambda_1^{(0)}, \lambda_2^{(0)}] = Arnoldi(y^{(0)}, A, k)$. Set $u^{(1)} = y^{(1)}$.
for $j = 1, 2, \ldots$
    a. Compute $[y^{(j+1)}, \lambda_1^{(j)}, \lambda_2^{(j)}] = Arnoldi(u^{(j)}, A, k)$
    b. Set $\gamma_j$
    c. Set $u^{(j+1)} = (1-\gamma_j)y^{(j+1)} + \gamma_j y^{(j)}$
end
```

In the tests of subsections 3.2 and 3.3, the condition to exit the loop on convergence is $\|Ay^{(j+1)} - \lambda_1^{(j)} y^{(j+1)}\| <$ `tol`, for a given tolerance `tol`.

The choice of extrapolation parameter γ_j is the key to a successful extrapolation. In [11], where the power iteration is accelerated by extrapolation, the parameter γ_j which gives asymptotically exponential convergence for positive semidefinite problems is an approximation of $-(\lambda_2/\lambda_1)^j$, where the eigenvalues of A are labeled in descending magnitude. Here we will see the approximation of $-(\lambda_2/\lambda_1)^j$ produced by the k-step Arnoldi method gives an effective acceleration of the k-step method; but, it is not clear that this is necessarily the best choice.

The complication in setting the extrapolation parameter lies in understanding the approximate eigenvector $y^{(j+1)}$ produced from Arnoldi$(u^{(j)}, A, k)$ as an expansion in the eigenbasis of A; in contrast to the power iteration, this expansion is not available in closed form for the k-step method for general values of k. For concreteness, suppose A is diagonalizable with a basis of orthonormal eigenvectors $\{v_i\}_{i=1}^n$, corresponding to eigenvalues $\{\lambda_i\}_{i=1}^n$, labeled with decreasing magnitude.

3.1. Analysis of extrapolation. By construction, the first generated iterate $y^{(1)} \in \mathcal{K}_k(y^{(0)}) = \text{span}\{y^{(0)}, Ay^{(0)}, \ldots A^{k-1}y^{(0)}\}$, the k-dimensional Krylov space generated by A applied to $y^{(0)}$. The next iterate $y^{(2)} \in \mathcal{K}_k(y^{(1)}) \subsetneq \mathcal{K}_{2k-1}(y^{(0)})$, and in general $y^{j+1} \in \mathcal{K}_k(y^{(k)}) \subsetneq \mathcal{K}_{(j+1)k-j}(y^{(0)})$.

A formal expansion of each $y^{(j)}$ in terms of the eigenbasis of A can be expressed as

$$y^{(j)} = \frac{1}{h_j}\sum_{i=1}^n p_i^{(j)}(\lambda_i)v_i, \quad h_j = \left(\sum_{i=1}^n (p_i^{(j)}(\lambda_i))^2\right)^{1/2},$$

where $p_i^{(j)}(\lambda_i)$ is a polynomial of degree at most $kj-(j-1)$ in λ_i. Now let's consider the ratio of the components of $u^{(j+1)}$ and $y^{(j)}$ in the direction of each eigenvector v_i. First define

$$\eta_i^{(j+1)} := \frac{v_i \cdot \text{proj}_{v_i} y^{(j+1)}}{v_i \cdot \text{proj}_{v_i} y^{(j)}} = \frac{h_j}{h_{j+1}} \frac{p_i^{(j+1)}(\lambda_i)}{p_i^{(j)}(\lambda_i)} = \frac{p_i^{(j+1)}(\lambda_i)/h_{j+1}}{p_i^{(j)}(\lambda_i)/h_j}. \tag{4}$$

Then, noting

$$\text{proj}_{v_i} u^{(j+1)} = \frac{p_i^{(j)}(\lambda_i)}{h_j} v_i \Big((1-\gamma_j)\frac{h_j}{h_{j+1}}\frac{p_i^{(j+1)}(\lambda_i)}{p_i^{(j)}(\lambda_i)} + \gamma_j\Big),$$

we have

$$\widehat{\eta}_i^{(j+1)} := \frac{v_i \cdot \text{proj}_{v_i} u^{(j+1)}}{v_i \cdot \text{proj}_{v_i} y^{(j)}} = (1-\gamma_j)\eta_i^{(j+1)} + \gamma_j. \tag{5}$$

We are first interested in how to use (5) to select values of parameter γ_j that are useful for accelerating convergence. We can understand the correct interval from

Table 1. Number of iterations to residual convergence of 10^{-7} for constant and dynamically chosen extrapolation parameters used in Algorithm 3.1 with $k = 8$.

γ_j / Matrix	0	-0.25	-0.5	-0.75	$-\left\lvert\frac{\lambda_2^{(j)}}{2\lambda_1^{(j)}}\right\rvert^2$	$-\left\lvert\frac{\lambda_2^{(j)}}{\lambda_1^{(j)}}\right\rvert$	$-\left\lvert\frac{\lambda_2^{(j)}}{\lambda_1^{(j)}}\right\rvert^j$
A_1	192	94	73	76	80	97	98
A_2 =`Kuu`	86	47	53	54	43	56	46
A_3 =`ifiss_mat`	165	105	83	42	79	72	68
A_4 =`gearbox`	157	48	52	52	56	82	82
A_5 =`ss1`	85	93	91	95	212	390	75
A_6 =`Si87H76`	63	32	37	37	32	37	33

which to choose this parameter by considering the special cases in the following proposition.

Proposition 3.2. *Consider $\eta_i^{(j+1)}$ given by* (4) *and $\widehat{\eta}_i^{(j+1)}$ given by* (5). *Then*

$$\begin{aligned} &\textit{if}\ \ \gamma_j = 1 \ \ \textit{then}\ \ \widehat{\eta}_i^{(j+1)} = 1,\\ &\textit{if}\ \ \gamma_j = 0 \ \ \textit{then}\ \ \widehat{\eta}_i^{(j+1)} = \eta_i^{(j+1)},\\ &\textit{if}\ \ \gamma_j = -1 \ \ \textit{then}\ \ \widehat{\eta}_i^{(j+1)} = 2\eta_i^{(j+1)} - 1. \end{aligned}$$

Proof. This follows directly from evaluating (5) with $\gamma_j = -1, 0, 1$. □

Since $\widehat{\eta}_i^{(j+1)}$ is defined by an affine transformation of $\eta_i^{(j+1)}$, we can use these three data points to choose an appropriate interval for parameter γ_j. Setting $\gamma_j = 1$ yields stagnation: no improvement in any mode i from step j to step $j+1$. Setting $\gamma_j = 0$ returns $\eta_i^{(j+1)}$ for any mode i (no extrapolation). Values of γ_j in $(0, 1)$ then function as a relaxation parameter. On the other hand, setting $\gamma = -1$ amplifies the damping of modes i for which $1/3 < \eta_i^{(j+1)} < 1$, and maintains $\widehat{\eta}_i^{(j+1)} \in [-1, 1]$ for $\eta_i^{(j+1)} \in [0, 1]$.

We are next interested in describing how the modes defined by (5) grow or decay compared to those defined by (4) for $\gamma_j \in (-1, 0)$. These results are summarized in the following proposition.

Proposition 3.3. *Let $\gamma_j \in (-1, 0)$, and consider $\eta_i^{(j+1)}$ given by* (4) *and $\widehat{\eta}_i^{(j+1)}$ given by* (5). *Then*

$$\begin{aligned} |\widehat{\eta}_i^{(j+1)}| &< \eta_i^{(j+1)}, \quad \textit{for}\ \ \gamma_j/(\gamma_j - 2) < \eta_i^{(j+1)} < 1,\\ |\widehat{\eta}_i^{(j+1)}| &< 1, \quad \textit{for}\ \ (\gamma_j + 1)/(\gamma_j - 1) < \eta_i^{(j+1)} < 1,\\ \widehat{\eta}_i^{(j+1)} &= 0, \quad \textit{for}\ \ \eta_i^{(j+1)} = -\gamma_j/(1 - \gamma_j). \end{aligned}$$

Proof. This follows directly from (5) which states $\widehat{\eta}_i^{(j+1)}$ in terms of a shifting and scaling of $\eta_i^{(j+1)}$ in terms of the parameter γ_j. □

Proposition 3.3 shows that choosing $\gamma_j \in (-1, 0)$ is beneficial for modes that slowly decrease (with no sign change) from Arnoldi(k). Modes which Arnoldi(k) damps very quickly ($\eta_i^{(j+1)}$ close to zero), are still decreased in magnitude, but not as much. The modes which Arnoldi(k) damps slowly but with a change of sign however increase in magnitude. This explains some of the nonmonotonic behavior in the convergence plots of Subsection 3.2.

Remark 3.4. *For example, as shown in Subsection 3.2, the constant extrapolation parameter $\gamma_j = -0.75$ performs well on most of the tests. From Proposition 3.3, setting $\gamma_j = -0.75$ yields*

$$\begin{aligned} |\widehat{\eta}_i^{(j+1)}| &< \eta_i^{(j+1)}, \quad \text{for } 3/11 < \eta_i^{(j+1)} < 1, \\ |\widehat{\eta}_i^{(j+1)}| &< 1, \quad \text{for } -1/7 < \eta_i^{(j+1)} < 1, \\ \widehat{\eta}_i^{(j+1)} &= 0, \quad \text{for } \eta_i^{(j+1)} = 3/7. \end{aligned}$$

The parameters $\gamma_j = -0.5$ and $\gamma_j = -0.25$ are also considered. From Proposition 3.3, the parameters closer to zero satisfy $|\widehat{\eta}_i^{(j+1)}| < \eta_i^{(j+1)}$, on a slightly wider positive interval, and $|\widehat{\eta}_i^{(j+1)}| < 1$, on a wider interval. However, the modes damped the most for $\gamma_j = -0.75$ are those that decay more slowly than those damped the most by $\gamma_j = -0.25$ and $\gamma_j = -0.5$, and this plays a significant role the the performance of the method.

Finally, the justification for the dynamically chosen parameter γ_s used in Subsection 3.2 is contained in the next result. Here we consider the choice $\gamma_j = -|\lambda_2/\lambda_1|^j$. The effectiveness of an approximation to this choice using $\lambda_2^{(j)}$ and $\lambda_1^{(j)}$, the latest approximation of the two dominant eigenvalues as returned from Arnoldi(u^j, A, k) at iteration j of Algorithm 3.1, is demonstrated in Subsection 3.2.

Lemma 3.5. *Consider $\eta_i^{(j+1)}$ given by* (4) *and $\widehat{\eta}_i^{(j+1)}$ given by* (5). *Set $\gamma_j = -|\lambda_2/\lambda_1|^j$, and suppose $0 < \eta_i^{(j+1)} \le |\lambda_i/\lambda_1|^j$. Then*

$$-\left|\frac{\lambda_2}{\lambda_1}\right|^j < \widehat{\eta}_i^{(j+1)} \le \left|\frac{\lambda_2}{\lambda_1}\right|^j \left|\frac{\lambda_i}{\lambda_1}\right|^j .$$

The assumption that $|\eta_i^{(j+1)}| \le |\lambda_i/\lambda_1|^j$ is heuristically justified by the supposition that Arnoldi(k) damps subdominant modes faster than the standard power iteration; see [15] and the numerical tests of Subsection 3.2. The assumption that $\eta_i^{(j+1)} > 0$ will not necessarily hold at each iteration of Algorithm 3.1, but as seen in Lemma 3.3, this is the assumption that guarantees accelerated damping of subdominant modes.

Proof. For the first inequality

$$\widehat{\eta}_i^{(j+1)} = \eta_i^{(j+1)} + |\lambda_2/\lambda_1|^j \eta_i^{(j+1)} - |\lambda_2/\lambda_1|^j > -|\lambda_2/\lambda_1|^j.$$

For the second inequality

$$\widehat{\eta}_i^{(j+1)} \le |\lambda_i/\lambda_1|^j + |\lambda_2/\lambda_1|^j|\lambda_i/\lambda_1|^j - |\lambda_2/\lambda_1|^j \le |\lambda_2/\lambda_1|^j|\lambda_i/\lambda_1|^j.$$

□

3.2. Convergence tests. All of our numerical tests were performed in Matlab, and the $k \times k$ eigenproblems required by Arnoldi(k) were solved using the command `eig`. In Table 1, the number of iterations to convergence with six choices of extrapolation parameter γ_j are compared on the following test set: $A_1 = \text{diag}(1000, -999, 998, \ldots, 2, -1)$; $A_2 =$`Kuu`, symmetric positive definite with $n = 7102$; $A_3 =$ `ifiss_mat`, neither positive definite nor symmetric, with $n = 96307$; $A_4 =$ `gearbox`, symmetric and not positive definite, with $n = 153746$; $A_5 =$ `ss1`, neither positive definite nor symmetric, with $n = 205282$; $A_6 =$ `Si87H76`, symmetric

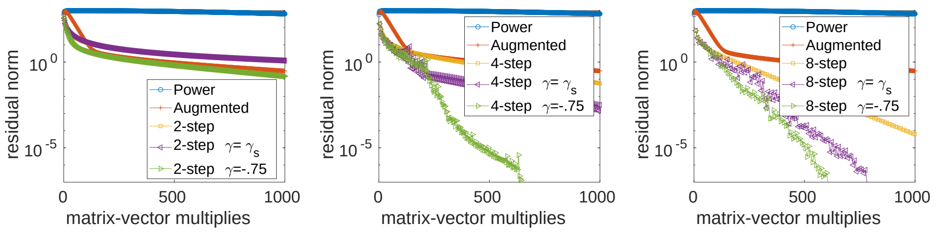

FIGURE 2. Residual history for Algorithm 3.1, with $k = 2$ (left), $k = 4$ (center) and $k = 8$ (right), with $\gamma_j = 0$ (k-step), $\gamma_j = \gamma_s = -\big(\lambda_2^{(j)}/\lambda_1^{(j)}\big)^j$ and $\gamma_j = -0.75$, compared with the power iteration and Augmented method of [11] with $\eta = 40$, applied to $A_1 = \text{diag}(1000, -999, \ldots, 2, -1)$.

and not positive definite, with $n = 240369$. Matrices A_2—A_6 are available from the SuiteSparse matrix collection [4]. Each iteration is started with $y^{(0)} = (1, \ldots, 1)^t$.

Three constant extrapolation parameters, $\gamma_j = -0.25, -0.5, -0.75$ are compared with three dynamically chosen parameters, $\gamma_j = -|\lambda_2^{(j)}/\lambda_1^{(j)}|^2/4$, $\gamma_j = -|\lambda_2^{(j)}/\lambda_1^{(j)}|$ and $\gamma_j = -|\lambda_2^{(j)}/\lambda_1^{(j)}|^j$, where $\lambda_1^{(j)}$ and $\lambda_2^{(j)}$ are the first and second eigenvalues returned on iteration j from Arnoldi$(u^{(j)}, A, k)$. The parameter $\gamma_j = -|\lambda_2^{(j)}/\lambda_1^{(j)}|^2/4$ is related to that used in [5]. The parameters $\gamma_j = -|\lambda_2^{(j)}/\lambda_1^{(j)}|$ and $\gamma_j = -|\lambda_2^{(j)}/\lambda_1^{(j)}|^j$, are based on the extrapolation parameter used in [11]. The last of these, $\gamma_j = \gamma_s = -|\lambda_2^{(j)}/\lambda_1^{(j)}|^j$ is denoted with subscript "s" due to its similarity to the parameter that defines the "simple" method in [11], which has established convergence properties for power iterations. All three dynamically chosen parameters are ensured to satisfy $-1 \le \gamma_j \le 0$, which is the interval of interest, as per the discussion above. On the whole, the constant extrapolation parameters significantly reduce computation in most cases, with a minor increase in the number of iterations in one tested case. The dynamically chosen parameter $\gamma_j = -|\lambda_2^{(j)}/\lambda_1^{(j)}|^j$ reduced the number of iterations in all cases, although not as effectively as the constant extrapolation in the case of A_4 =`gearbox`. With the exception of A_5 =`ss1`, the extrapolation methods reduced the number of iterations (without any additional matrix-vector multiplies) generally by 50% or more. The two extrapolation parameters from table 1 which provide the greatest benefit in most cases, while causing minimal damage in the worst, are the constant parameter $\gamma_j = -0.75$, and the dynamic paramter $\gamma_j = -|\lambda_2^{(j)}/\lambda_1^{(j)}|^j$. These two successful parameters are next considered for different choices k in the k-step method.

3.3. Convergence details. Figures 2—4 compare Algorithm 3.1 with parameters $\gamma_j = -0.75$ and $\gamma_j = \gamma_s = -|\lambda_2^{(j)}/\lambda_1^{(j)}|^j$ with the k-step method ($\gamma_j = 0$) using different values of k, together with the power iteration and an extrapolated power iteration referred to as the Augmented method of [11]. Each plot shows the l_2 norm of the residual for the dominant eigenpair on the y-axis, and the number of matrix-vector multiplies on the x-axis. In figure 2 it is notable for matrix A_1 that the constant extrapolation $\gamma_j = -0.75$ accelerates convergence for $k = 4, 8$, whereas the dynamically assigned parameter γ_s accelerates convergence nearly as well, but only for $k = 8$. This illustrates that the constant parameter may be preferable in cases where the second eigenvalue may not be approximated well. Figure 3 (left) with $k = 4$ illustrates with A_3 = `ifiss_mat` that the extrapolation does not

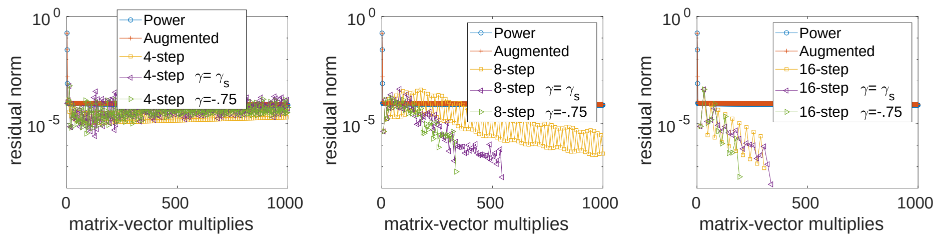

FIGURE 3. Residual history for Algorithm 3.1, with $k = 4$ (left), $k = 8$ (center) and $k = 16$ (right), with $\gamma_j = 0$ (k-step), $\gamma_j = \gamma_s = -\big(\lambda_2^{(j)}/\lambda_1^{(j)}\big)^j$ and $\gamma_j = -0.75$, compared with the power iteration and Augmented method of [11] with $\eta = 40$, applied to $A_3 =$ `ifiss_mat`.

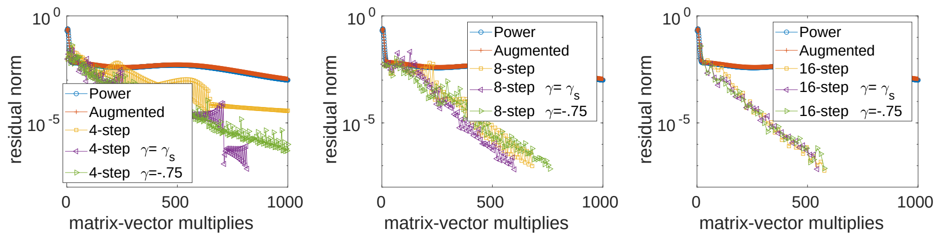

FIGURE 4. Residual history for Algorithm 3.1, with $k = 2$ (left), $k = 4$ (center) and $k = 8$ (right), with $\gamma_j = 0$ (k-step), $\gamma_j = \gamma_s = -\big(\lambda_2^{(j)}/\lambda_1^{(j)}\big)^j$ and $\gamma_j = -0.75$, compared with the power iteration and Augmented method of [11] with $\eta = 40$, applied to $A_5 =$ `ss1`.

necessarily induce convergence where the k-step method itself does not converge. The center and right plots with $k = 8$ and $k = 16$ illustrate that the extrapolation can still significantly improve convergence where the residual for the k-step method demonstrates oscillatory behavior. Figure 4 (left) with $k = 4$ illustrates with $A_5 =$ `ss1` an instance where the dynamically chosen γ_s reduces the number of iterations more effectively than the constant parameter for a smaller value of k, however the center and right plots with $k = 8$ and $k = 16$ show little difference with or without the extrapolation.

The examples shown in table 1 and figures 2—4 together illustrate that for large enough values of k, the k-step method converges efficiently with respect to the number of matrix-vector multiplies; however, for the range of k values for which the k-step method converges but not efficiently, the extrapolation improves convergence and restores efficiency. These results are consistent with the observations in [13] and [15] that the restarted Arnoldi or k-step method with $k = j \times k'$ is more effective than j iterations of the k'-step method. However, the extrapolation reduces the sensitivity of the method to the choice of k, so that smaller values of k can produce nearly the same efficiency, and with less demand on system memory.

4. LOBPCG and k-step methods

Locally Optimal Block Preconditioned Conjugate Gradient (LOBPCG) methods [1, 8, 9] have proved to be very effective for solving eigenproblems. Here we compare them with the k-step methods. For simplicity, we restrict to the case that there is no preconditioning and the eigenproblem is standard, not generalized. Referring to the Wikipedia page[1] for LOBPCG, we can write the basic algorithm as follows. First define the Rayleigh quotient $\rho(\mathbf{x})$ by

$$\rho(\mathbf{x}) = \frac{\mathbf{x}^t A\mathbf{x}}{\mathbf{x}^t\mathbf{x}},$$

and the residual

$$\mathbf{r}(\mathbf{x}) = A\mathbf{x} - \rho(\mathbf{x})\mathbf{x}.$$

Given $\mathbf{x}_i$, LOBPCG defines

$$\mathbf{x}_{i+1} = \arg \max_{\mathbf{y}\in \operatorname{span}\{\mathbf{x}_i,\mathbf{r}(\mathbf{x}_i)\}} \rho(\mathbf{y}).$$

But assuming that $A\mathbf{x}_i \neq \rho(\mathbf{x}_i)\mathbf{x}_i$, that is, the algorithm has not converged,

$$\operatorname{span}\{\mathbf{x}_i, \mathbf{r}(\mathbf{x}_i)\} = \operatorname{span}\{\mathbf{x}_i, A\mathbf{x}_i\}.$$

Therefore this form of LOBPCG is the same as the 2-step algorithm. In the Wikipedia page for LOBPCG, the 3-step algorithm is also described.

5. Enhancements of k-step methods

There are several possible enhancements of k-step methods.

5.1. Block k-step methods. Block k-step methods can also be useful. For simplicity, suppose that k is even. A block 2-step method, with block size $k/2$, would work as follows. First pick $k/2$ initial vectors $\mathbf{w}_i$. (This could be done by generating $k/2$ Krylov vectors.) Next perform one Krylov step for each i:

$$\mathbf{w}_{i+k/2} = A\mathbf{w}_i$$

and solve the reduced $k\times k$ eigenproblem arising by projecting onto this k-dimensional subspace. Then pick the $k/2$ most extreme eigenpairs, $(\lambda_i, \mathbf{w}_i)$, $i = 1, \ldots, k/2$. With these k vectors, we form the reduced eigenproblem and repeat as desired. We could of course also consider k to be a multiple of μ for $\mu > 2$. We could then retain only k/μ vectors and perform $\mu - 1$ Krylov steps for each retained vector.

5.2. Preconditioning and generalized eigenproblems. The approach taken to k-step methods described via LOBPCG methods can clearly be used to implement them. Thus preconditioning and generalized eigenproblems can also be used. We postpone to a later publication the study of extrapolation in these contexts.

6. Conclusions and perspectives

We have considered using small eigenproblems as a technique to enhance the performance of matrix-free methods, such as the power method. We introduce the concept of the k-step method, which we identify as an Arnoldi method and also the LOBPCG method. We have examined the performance as a function of k for various test matrices. But most importantly, we have shown that extrapolation can be used as a simple post-processing procedure to enhance the k-step method at essentially zero cost. Extrapolation can be added easily to packages such as SLEPc [7].

[1]`https://en.wikipedia.org_wiki_LOBPCG`

Acknowledgments

SP is supported in part by the National Science Foundation NSF DMS-1852876 and NSF DMS-2011519. Both authors would like to thank Nilima Nigam for valuable feedback on a preliminary version of the manuscript, and an anonymous referee for suggestions that added significant clarity to this paper.

References

[1] P. Benner and T. Mach, Locally optimal block preconditioned conjugate gradient method for hierarchical matrices, PAMM, 11 (2011), pp. 741–742.

[2] Å. Björck, Numerics of Gram–Schmidt orthogonalization, Linear Algebra and Its Applications, 197 (1994), pp. 297–316.

[3] E. Cancès and L. R. Scott, van der Waals interactions between two hydrogen atoms: The Slater-Kirkwood method revisited, SIAM Journal on Mathematical Analysis, 50 (2018), pp. 381–410.

[4] T. A. Davis and Y. Hu, The University of Florida sparse matrix collection, ACM Transactions on Mathematical Software (TOMS), 38 (2011), pp. 1–25.

[5] C. De Sa, B. He, I. Mitliagkas, C. Ré, and P. Xu, Accelerated stochastic power iteration, Proceedings of Machine Learning Research, 84 (2019), pp. 58–67.

[6] P. Farrell, L. Mitchell, L. R. Scott, and F. Wechsung, Robust discretization and multigrid solution for incompressible and nearly incompressible continua, TBD, (2021).

[7] V. Hernandez, J. E. Roman, and V. Vidal, SLEPc: A scalable and flexible toolkit for the solution of eigenvalue problems, ACM Transactions on Mathematical Software (TOMS), 31 (2005), pp. 351–362.

[8] A. Knyazev, Recent implementations, applications, and extensions of the Locally Optimal Block Preconditioned Conjugate Gradient method (LOBPCG), arXiv preprint arXiv:1708.08354, (2017).

[9] A. V. Knyazev, Toward the optimal preconditioned eigensolver: Locally optimal block preconditioned conjugate gradient method, SIAM Journal on Scientific Computing, 23 (2001), pp. 517–541.

[10] R. B. Lehoucq and D. C. Sorensen, Deflation techniques for an implicitly restarted arnoldi iteration, SIAM Journal on Matrix Analysis and Applications, 17 (1996), pp. 789–821.

[11] N. Nigam and S. Pollock, A simple extrapolation method for clustered eigenvalues, Numerical Algorithms, (2021) `https://doi.org/10.1007/s11075-021-01108-7`.

[12] C. C. Paige, M. Rozložník, and Z. Strakos, Modified Gram-Schmidt (MGS), least squares, and backward stability of MGS-GMRES, SIAM Journal on Matrix Analysis and Applications, 28 (2006), pp. 264–284.

[13] S. Pollock and L. R. Scott, Using small eigenproblems to accelerate power method iterations, Research Report UC/CS TR-2021-10, Dept. Comp. Sci., Univ. Chicago, 2021.

[14] Y. Saad, On the rates of convergence of the Lanczos and the block-Lanczos methods, SIAM Journal on Numerical Analysis, 17 (1980), pp. 687–706.

[15] Y. Saad, Variations on Arnoldi's method for computing eigenelements of large unsymmetric matrices, Linear algebra and its applications, 34 (1980), pp. 269–295.

[16] L. R. Scott, Kinetic energy flow instability with application to Couette flow, Research Report UC/CS TR-2020-07, Dept. Comp. Sci., Univ. Chicago, 2020.

[17] D. C. Sorensen, Numerical methods for large eigenvalue problems, Acta Numerica, 11 (2002), pp. 519.

Department of Mathematics, University of Florida, Gainesville, FL 32605, USA
E-mail: `s.pollock@ufl.edu`
URL: `https://people.clas.ufl.edu/spollock/`

Department of Computer Science, University of Chicago, Chicago, IL 60637, USA
E-mail: `ridg@uchicago.edu`
URL: `https://people.cs.uchicago.edu/∼ridg/`