Daniel Kressner; Román Moltó, JE. (2014). Memory-efficient Arnoldi algorithms for linearizations of matrix polynomials in Chebyshev basis. Numerical Linear Algebra with Applications. 21(4):569-588. doi:10.1002/nla.1913.

Memory-efficient Arnoldi algorithms for linearizations of matrix polynomials in Chebyshev basis

Daniel Kressner* Jose E. Roman†

November 4, 2013

Abstract

Novel memory-efficient Arnoldi algorithms for solving matrix polynomial eigenvalue problems are presented. More specifically, we consider the case of matrix polynomials expressed in the Chebyshev basis, which is often numerically more appropriate than the standard monomial basis for a larger degree d. The standard way of solving polynomial eigenvalue problems proceeds by linearization, which increases the problem size by a factor d. Consequently, the memory requirements of Krylov subspace methods applied to the linearization grow by this factor. In this paper, we develop two variants of the Arnoldi method that build the Krylov subspace basis implicitly, in a way that only vectors of length equal to the size of the original problem need to be stored. The proposed variants are generalizations of the so called Q-Arnoldi and TOAR methods, which have been developed for the monomial case. We also show how the typical ingredients of a full implementation of the Arnoldi method, including shift-and-invert and restarting, can be incorporated. Numerical experiments are presented for matrix polynomials up to degree 30 arising from the interpolation of nonlinear eigenvalue problems which stem from boundary element discretizations of PDE eigenvalue problems.

1 Introduction

This paper is concerned with the polynomial eigenvalue problem

$$P(\lambda)x = 0, \qquad x \neq 0, \tag{1}$$

where $P : \mathbb{C} \to \mathbb{C}^{n\times n}$ is a matrix polynomial of degree d. Often, P is expressed in the monomial basis, that is,

$$P(\lambda) = A_0 + \lambda A_1 + \lambda^2 A_2 + \cdots + \lambda^d A_d \tag{2}$$

with coefficient matrices $A_0, \ldots, A_d \in \mathbb{C}^{n\times n}$.

The numerical solution of the polynomial eigenvalue problem (1) with P as in (2) usually proceeds by *linearization*. This yields an equivalent $dn\times dn$ linear eigenvalue problem $\mathcal{L}_0 - \lambda\mathcal{L}_1$, to which standard linear eigenvalue solvers like the QZ algorithm or the Arnoldi method [11] can be applied. Many different linearizations are possible, and much progress has been made

*ANCHP, MATHICSE, EPF Lausanne, Switzerland, daniel.kressner@epfl.ch
†DSIC, Universitat Politècnica de València, Spain, jroman@dsic.upv.es

in the last few years in developing a unified framework [21], which allows, e.g., for structure preservation [20] and a unified sensitivity analysis [14, 1].

The major disadvantage of linearization is that it increases the problem size by a factor d. For example, this implies that the Arnoldi method applied to $\mathcal{L}_0 - \lambda\mathcal{L}_1$ operates with a Krylov subspace of $\mathbb{C}^{dn}$. In large-scale applications, it is not unlikely that the need to store k vectors of length dn to represent a k-dimensional Krylov subspace becomes a limiting factor, especially for larger d. To reduce these memory requirements, variants of Arnoldi methods tailored to polynomial eigenvalue problems have been developed which only require the storage of k vectors of length n. Examples of such methods include SOAR [3] (second-order Arnoldi method) and Q-Arnoldi [22] (quadratic Arnoldi method). Both methods have in common that they implicitly run the standard Arnoldi method and exploit the structure of the linearization to reduce memory requirements. They mainly differ in the way eigenvalues approximations are obtained: SOAR uses a projection of the original polynomial eigenvalue problem while Q-Arnoldi uses a projection of the linearization – just what the standard Arnoldi method would do. This makes it much simpler to implement restarting and locking procedures for Q-Arnoldi, at the price of losing the accelerated convergence sometimes observed for SOAR. Note that both SOAR and Q-Arnoldi have originally been defined for the quadratic case ($d = 2$), but they extend in a direct fashion to arbitrary degree, see, e.g., [18].

The standard Arnoldi method combined with appropriate reorthogonalization is numerically backward stable [25]. In contrast, both SOAR and Q-Arnoldi may suffer from numerical instabilities due the fact that the relations of the Arnoldi method are only maintained implicitly. Numerical evidence suggests that these instabilities become more pronounced as d increases. Recently, a new variant called TOAR [19] (two-level orthogonal Arnoldi procedure) has been proposed that provably avoids the instabilities of Q-Arnoldi. This is achieved by a more explicit representation of the orthogonality relations in the Arnoldi method, without increasing the storage requirements significantly. Although some details, such as robust restarting and locking procedures, are still under development, TOAR appears to be a very promising alternative to SOAR and Q-Arnoldi.

In this paper, we will specifically consider the case of matrix polynomials of high degree, say $d = 16$ or $d = 40$. Such high degrees typically result from the approximation of a nonlinear problem, for example, by means of Taylor expansion [15], polynomial interpolation [6, 7, 9] or Hermite interpolation [29]. It is well known that the monomial representation (2) leads to severe numerical difficulties for high degrees, unless the eigenvalues are (nearly) distributed along a circle [23]. In the (more typical) case when the eigenvalues on or close to an interval $I \subset \mathbb{R}$ are of interest, a non-monomial representation is numerically much more suitable. More specifically, if we consider w.l.o.g. $I = [-1, 1]$ (which can always be achieved by an affine linear transformation) then a numerically reliable representation [28] is given by

$$P(\lambda) = \tau_0(\lambda)A_0 + \tau_1(\lambda)A_1 + \cdots + \tau_d(\lambda)A_d, \tag{3}$$

where $\tau_0, \tau_1, \ldots, \tau_d$ denote the Chebyshev polynomials of the first kind.

Again, linearization techniques can be used to solve eigenvalue problems for matrix polynomials of the form (3) numerically. Linearizations for general polynomial bases satisfying a three-term recurrence have recently been proposed in [2]. The aim of this paper is to develop memory-efficient Arnoldi algorithms, similar to the ones discussed above, for such linearizations. We believe that this extension is particularly important when dealing with approximations to nonlinear eigenvalue problems. The reduced memory requirements may allow

one to work with much higher polynomial degrees for large-scale problems and consequently result in a significantly improved quality of the approximation.

The rest of this paper is organized as follows. In Section 2, we recall the linearization of polynomials in the Chebyshev basis and some of its fundamental properties. Section 3 shows that the Q-Arnoldi algorithm can be extended to this situation in an elegant way. For practical purposes, however, a variant of TOAR developed in Section 4 appears to be more important. In Section 5, we demonstrate the efficiency of this new variant of TOAR for addressing polynomial eigenvalue problems arising from the polynomial interpolation of discretized PDE eigenvalue problems.

2 Preliminaries

In this section, we briefly recall the linearization from [2] for a polynomial eigenvalue problem

$$\big(A_0\tau_0(\lambda) + \cdots + A_d\tau_d(\lambda)\big)x = 0, \tag{4}$$

where $\tau_0, \dots \tau_d$ are the Chebyshev polynomials of first kind, defined by the recurrence

$$\tau_0(\lambda) = 1, \qquad \tau_1(\lambda) = \lambda, \qquad \tau_{j+1}(\lambda) = 2\lambda\tau_j(\lambda) - \tau_{j-1}(\lambda). \tag{5}$$

Then the $dn \times dn$ matrices

$$\mathcal{L}_0 = \begin{bmatrix} 0 & I & & & \\ I & 0 & I & & \\ & \ddots & \ddots & \ddots & \\ & & I & 0 & I \\ -A_0 & \cdots & -A_{d-3} & A_d - A_{d-2} & -A_{d-1} \end{bmatrix}, \quad \mathcal{L}_1 = \begin{bmatrix} I & & & & \\ & 2I & & & \\ & & \ddots & & \\ & & & 2I & \\ & & & & 2A_d \end{bmatrix}, \tag{6}$$

yield a linear eigenvalue problem

$$\mathcal{L}_0 y = \lambda \mathcal{L}_1 y, \tag{7}$$

which turns out to be a strong linearization of (4). More specifically, if (λ, x) is an eigenpair of the polynomial eigenvalue problem (4) then (λ, v) is an eigenpair of (7), with

$$v = \begin{bmatrix} \tau_0(\lambda)x \\ \tau_1(\lambda)x \\ \vdots \\ \tau_{d-1}(\lambda)x \end{bmatrix} = \begin{bmatrix} x \\ \tau_1(\lambda)x \\ \vdots \\ \tau_{d-1}(\lambda)x \end{bmatrix}. \tag{8}$$

This can be directly seen from the structure of the matrices in (6), using the recurrence (5). Conversely, any eigenvector of $\mathcal{L}_0 - \lambda\mathcal{L}_1$ has the structure (8), see [21, Theorem 3.8], and thus easily allows for the extraction of the eigenvector x from its first n entries.

The concept of invariant pairs [4, 5, 16] generalizes the relations above from eigenvectors to subspaces. A pair $(X, \Lambda) \in \mathbb{C}^{n\times k} \times \mathbb{C}^{k\times k}$ is called an invariant pair for (4) if it satisfies the matrix equation

$$A_0X\tau_0(\Lambda) + A_1X\tau_1(\Lambda) + \cdots + A_dX\tau_d(\Lambda) = 0, \tag{9}$$

where $\tau_j(\Lambda)$ is defined in the sense of the usual matrix polynomial. Then (V, Λ) is an invariant pair of (7), with

$$V = \begin{bmatrix} X\tau_0(\Lambda) \\ X\tau_1(\Lambda) \\ \vdots \\ X\tau_{d-1}(\Lambda) \end{bmatrix} = \begin{bmatrix} X \\ X\tau_1(\Lambda) \\ \vdots \\ X\tau_{d-1}(\Lambda) \end{bmatrix}, \tag{10}$$

see also [9]. Invariant pairs generalize the notion of eigenpairs in the following sense: If (X, Λ) is minimal, i.e., V has full column rank, then the eigenvalues of Λ are eigenvalues of (4) and $\operatorname{span}(X)$ contains the corresponding (generalized) eigenvectors [16]. The structure of (10) suggests a simple way of extracting such a minimal invariant pair (X, Λ) from the solution of the linearized eigenvalue problem; we refer to [4, 9] for more details. It is important to remark that the process of linearization may increase the sensitivity of the eigenvalue problem, in particular when the norm of the coefficients A_j varies strongly. In this case, it is usually helpful to perform iterative refinement after the extraction procedure. This can be performed by applying the Newton method to the equation (9) combined with a normalization condition [4, 16].

3 A P-Arnoldi method

In this section, we develop a new variant of the P-Arnoldi method for polynomials in the Chebyshev basis. Here, P-Arnoldi stands for the extension of Meerbergen's Q-Arnoldi method [22] to polynomials of higher degree (in the monomial basis). Our new method essentially consists of applying the standard Arnoldi method to

$$\mathcal{A} = \mathcal{L}_1^{-1}\mathcal{L}_0 = \frac{1}{2}\begin{bmatrix} 0 & 2I & & & & \\ I & 0 & I & & & \\ & \ddots & \ddots & \ddots & & \\ & & I & 0 & I & \\ -A_d^{-1}A_0 & \cdots & -A_d^{-1}A_{d-3} & I - A_d^{-1}A_{d-2} & -A_d^{-1}A_{d-1} \end{bmatrix} \tag{11}$$

implicitly, such that the generated basis can be represented in a memory-efficient way.

Algorithm 1 Arnoldi method

Input: Matrix $\mathcal{A}$, starting vector $\overline{u} \neq 0$, $k \in \mathbb{N}$.

Output: Matrix U and vector u such that $\big[U, u\big]$ is an orthonormal basis of the Krylov subspace $\mathcal{K}_{k+1}(\mathcal{A}, \overline{u}) = \operatorname{span}\{\overline{u}, \mathcal{A}\overline{u}, \ldots, \mathcal{A}^k\overline{u}\}$.

1: Normalize $u \leftarrow \overline{u}/\|\overline{u}\|_2$ and set $U \leftarrow [\,]$
2: **for** $j = 1, 2, \ldots, k$ **do**
3: Update $U \leftarrow \big[U, u\big]$
4: Compute $w \leftarrow \mathcal{A}u$
5: Compute $h_j \leftarrow U^*w$
6: Compute $\widetilde{u} \leftarrow w - Uh_j$
7: Set $h_{j+1,j} = \|\widetilde{u}\|_2$
8: Normalize $u \leftarrow \widetilde{u}/h_{j+1,j}$
9: **end for**

Algorithm 1 shows the standard Arnoldi method, which produces the so called *Arnoldi relation*

$$\mathcal{A}U = UH + \beta u e_k^*, \tag{12}$$

where H is a Hessenberg matrix collecting the vectors h_j and scalars $h_{j+1,j}$ generated during the algorithm. Moreover, $\beta = h_{k+1,k}$, and e_k is the kth unit vector of length k. It will sometimes be convenient to express the Arnoldi decomposition (12) as

$$\mathcal{A}U = \big[U, u\big]\underline{H}, \tag{13}$$

where $\underline{H} = \begin{bmatrix} H \\ \beta e_k^* \end{bmatrix}$.

For simplicity, the description of Algorithm 1 omits some important implementation details, such as reorthogonalization. We refer, e.g., to the ARPACK users' guide [17] or SLEPc [13] documentation for such details.

3.1 Structure of the Krylov subspace basis

As the polynomial degree d increases, the memory consumption of Algorithm 1 grows significantly due to the need to store the $dn \times k$ basis U. In the following, we explain how this effect can be avoided by exploiting the structure of U induced by the structure of the linearization. For this purpose, we partition

$$U = \begin{bmatrix} U_0 \\ \vdots \\ U_{d-1} \end{bmatrix}, \qquad u = \begin{bmatrix} u_0 \\ \vdots \\ u_{d-1} \end{bmatrix} \tag{14}$$

conformally with $\mathcal{A}$. The goal of our P-Arnoldi method will be to carry out all computations without explicit reference to the $U_1, \ldots, U_{d-1}$ blocks, so that only the $n \times k$ matrix U_0 needs to be stored.

A compact representation of U can be derived from the block rows of the Arnoldi decomposition (13). For example, by using the structure of $\mathcal{A}$ in (11) the first block row gives

$$U_1 = \big[U_0, u_0\big]\underline{H}. \tag{15}$$

The second block row gives

$$\frac{1}{2}U_0 + \frac{1}{2}U_2 = \big[U_1, u_1\big]\underline{H},$$

and hence $U_2 = 2\big[U_1, u_1\big]\underline{H} - U_0$. More generally, the jth row block yields

$$U_j = 2\big[U_{j-1}, u_{j-1}\big]\underline{H} - U_{j-2}, \qquad j = 2, \ldots, d-1. \tag{16}$$

This already shows that we can reconstruct all U_j blocks from U_0, u, H, and β. The following lemma yields a more explicit expression for (16).

Lemma 1. *Given an Arnoldi decomposition (13), with U and u partitioned as in (14), the relation*

$$U_j = \tilde{U}_j B_j \tag{17}$$

holds for all $j = 1, \ldots, d-1$, *where* $\tilde{U}_j = \big[U_0, u_0, \ldots, u_{j-1}\big]$, *and* B_j *is defined by the recurrence*

$$B_0 = I_k, \qquad B_1 = \underline{H}, \qquad B_j = 2 \begin{bmatrix} B_{j-1} & 0 \\ 0 & 1 \end{bmatrix} \underline{H} - \begin{bmatrix} B_{j-2} \\ 0 \\ 0 \end{bmatrix}. \tag{18}$$

Proof. The proof proceeds by induction with respect to j. The case $j = 0$ trivially holds and the case $j = 1$ follows directly from (15). For the induction step, we assume that relation (17) holds for $j - 1$ and j. Then the $(j + 1)$th block row of (13) gives

$$\begin{aligned} U_{j+1} &= 2\big[U_j, u_j\big]\underline{H} - U_{j-1} = 2\big[\tilde{U}_j B_j, u_j\big]\underline{H} - U_{j-1} \\ &= 2\left[\big[\tilde{U}_{j-1}, u_{j-1}\big] B_j, u_j\right] \underline{H} - U_{j-1} \\ &= 2\big[\tilde{U}_{j-1}, u_{j-1}, u_j\big] \begin{bmatrix} B_j & 0 \\ 0 & 1 \end{bmatrix} \underline{H} - \tilde{U}_{j-1} B_{j-1} \\ &= \big[\tilde{U}_{j-1}, u_{j-1}, u_j\big] \left(2 \begin{bmatrix} B_j & 0 \\ 0 & 1 \end{bmatrix} \underline{H} - \begin{bmatrix} B_{j-1} \\ 0 \\ 0 \end{bmatrix} \right) \\ &= \tilde{U}_{j+1} B_{j+1}. \end{aligned}$$

This shows relation (17) for $j + 1$, which concludes the proof. □

Considering Algorithm 1, the operations that need to be performed with U_j are matrix-vector products with U_j itself or with U_j^*. Lemma 1 implies that these matrix-vector products can be performed without storing or constructing $U_1, \ldots, U_d$, once we have precomputed the $(k + j) \times k$ matrices B_j. However, this precomputation yields an additional cost of order $O(dk^3)$, which can actually be avoided. This will be a consequence of the following result.

Lemma 2. *Given an Arnoldi decomposition* (12), *with* U *and* u *partitioned as in* (14), *the relation*

$$U_j = U_0 \tau_j(H) + \beta u_0 e_k^* \tilde{\tau}_j(H) + 2\beta \sum_{i=1}^{j-1} u_i e_k^* \tilde{\tau}_{j-i}(H) \tag{19}$$

holds for all $j = 1, \ldots, d-1$, *where* $\tilde{\tau}_0, \ldots, \tilde{\tau}_{d-1}$ *denote the Chebyshev polynomials of second kind, defined by the recursion*

$$\tilde{\tau}_0(\lambda) = 0, \qquad \tilde{\tau}_1(\lambda) = 1, \qquad \tilde{\tau}_{j+1}(\lambda) = 2\lambda \tilde{\tau}_j(\lambda) - \tilde{\tau}_{j-1}(\lambda). \tag{20}$$

Proof. Again, the proof proceeds by induction with respect to j. The case $j = 0$ trivially holds. For the case $j = 1$, consider the first block row of (12):

$$U_1 = U_0 H + \beta u_0 e_k^* = U_0 \tau_1(H) + \beta u_0 e_k^* \tilde{\tau}_1(H).$$

This coincides with (19) for $j = 1$. For the induction step, we assume that relation (19) holds

for $j-1$ and j. Then the $(j+1)$th block row of (13) gives

$$\begin{aligned} U_{j+1} &= 2U_jH - U_{j-1} + 2\beta u_j e_k^* \\ &= U_0\left(2\tau_j(H)H - \tau_{j-1}(H)\right) + \beta u_0 e_k^*\left(\tilde{\tau}_j(H)H - \tilde{\tau}_{j-1}(H)\right) \\ &\quad +4\beta\sum_{i=1}^{j-1} u_i e_k^*\tilde{\tau}_{j-i}(H)H - 2\beta\sum_{i=1}^{j-2} u_i e_k^*\tilde{\tau}_{j-i-1}(H) + 2\beta u_j e_k^* \\ &= U_0\tau_{j+1}(H) + \beta u_0 e_k^*\tilde{\tau}_{j+1}(H) + 2\beta\sum_{i=1}^{j-2} u_i e_k^*\tilde{\tau}_{j-i+1}(H) \\ &\quad +4\beta u_{j-1}e_k^*H + 2\beta u_j e_k^* \\ &= U_0\tau_{j+1}(H) + \beta u_0 e_k^*\tilde{\tau}_{j+1}(H) + 2\beta\sum_{i=1}^{j} u_i e_k^*\tilde{\tau}_{j-i+1}(H), \end{aligned}$$

which coincides with (19) for $j+1$ and thus concludes the proof. □

3.2 Algorithmic details

We now proceed with discussing the efficient implementation of the three main computational tasks required by the Arnoldi algorithm, i.e., steps 4–6 in Algorithm 1. The first task, the matrix-vector product with $\mathcal{A}$, is straightforward and is shown in Algorithm 2.

Algorithm 2 P-Arnoldi: Matrix-vector product with $\mathcal{A}$

Input: Matrix $\mathcal{A}$, vector u partitioned as in (14)
Output: Vector $w = \mathcal{A}u$ partitioned as u

1: Set $w_0 = u_1$
2: **for** $j = 1, \ldots, d-2$ **do**
3: Set $w_j = \frac{1}{2}(u_{j-1} + u_{j+1})$
4: **end for**
5: Compute $w_{d-1} = \frac{1}{2}u_{d-2} - \frac{1}{2}A_d^{-1}\sum_{i=0}^{d-1} A_i u_i$

For steps 5 and 6, which perform the orthogonalization against the Arnoldi basis U, we will make use of Lemma 2. In step 5 of Algorithm 1, the orthogonalization coefficient h_j is computed, which amounts to evaluating an expression of the form

$$h_j = \sum_{j=0}^{d-1} U_j^* w_j = \sum_{j=0}^{d-1}\tau_j(H^*)U_0^* w_j + \overline{\beta}\sum_{j=0}^{d-1} u_0^* w_j\tilde{\tau}_j(H^*)e_k + 2\beta\sum_{j=0}^{d-1}\sum_{i=1}^{j-1} u_i^* w_j\tilde{\tau}_{j-i}(H^*)e_k \quad (21)$$

for given vectors $w_0, \ldots, w_{d-1}$. The first sum in (21) can be evaluated by Clenshaw's algorithm [8] for evaluating polynomials in Chebyshev basis. This requires the computation of $d-1$ matrix-vector products $U_0^* w_j$, which can be reorganized as a single matrix-matrix product: $U_0^*[w_0, \ldots, w_{d-1}]$. The last two sums in (21) amount to a matrix-matrix multiplication with the $k \times (d-1)$ matrix

$$T = \overline{\beta}\left[\tilde{\tau}_{d-1}(H^*)e_k,\ \ldots,\ \tilde{\tau}_2(H^*)e_k,\ \tilde{\tau}_1(H^*)e_k\right] \quad (22)$$

which can be formed recursively from right-to-left using the recurrence relation (20). These ideas lead to Algorithm 3.

Algorithm 3 P-Arnoldi: Orthogonalization coefficients h_j

Input: Matrices U_0, H, vectors u, w partitioned as in (14), scalar β
Output: Vector $h_j = U^* w$

1: Compute $Y = U_0^* \big[w_0, w_1, \ldots, w_{d-1}\big]$
2: Set $b_{d-1} = Y e_d$ and $b_{d-2} = Y e_{d-1} + 2H^* b_{d-1}$
3: **for** $j = d-3, d-4, \ldots, 1$ **do**
4: Set $b_j = Y e_{j+1} + 2H^* b_{j+1} - b_{j+2}$
5: **end for**
6: Set $s_1 = Y e_1 + H^* b_1 - b_2$
7: Build matrix T in (22)
8: Compute $Z = \big[u_0, \ldots, u_{d-2}\big]^* \big[w_1, \ldots, w_{d-1}\big]$.
9: Set $v = Z(1,:)^T$
10: **for** $j = 2, \ldots, d-1$ **do**
11: Compute $v(1:d-j) = v(1:d-j) + Z(j, 1:d-j)^T$
12: **end for**
13: Return $h_j = s_1 + Tv$

Finally, step 6 of Algorithm 1 requires the computation of

$$U_0 b, \quad U_1 b, \quad \ldots, \quad U_{d-1} b$$

for a given vector b. Using Lemma 2, this means that we have to evaluate

$$U_j b = U_0 \tau_j(H) b + \beta u_0 e_k^* \tilde{\tau}_j(H) b + 2\beta \sum_{i=1}^{j-1} u_i e_k^* \tilde{\tau}_{j-i}(H) b$$

for $j = 0, \ldots, d-1$. After precomputing

$$T = \beta \big[\tau_0(H) b, \ldots, \tau_{d-1}(H) b\big], \qquad \tilde{T} = \beta \big[\tilde{\tau}_1(H) b, \ldots, \tilde{\tau}_{d-1}(H) b\big], \tag{23}$$

using the recurrence relations, this formula can be easily evaluated, see Algorithm 4.

To summarize, the P-Arnoldi method for polynomials in Chebyshev basis is Algorithm 1 with steps 4–6 realized by Algorithms 2–4. This allows to work with U_0 only and thus reduces the memory requirements from $O(dnk)$ to $O(kn + dn)$. Similarly to the monomial case [22], the P-Arnoldi method easily allows for the use of implicit restarting and locking techniques, e.g., by means of the Krylov-Schur algorithm [26].

4 A shift-and-invert TOAR method

The P-Arnoldi method applied to $\mathcal{A}$ can be expected to first converge to the eigenvalues in the periphery of the spectrum [25]. However, often the eigenvalues of interest are those closest to a given target value σ, which is typically in the interior of the spectrum. The shift-and-invert technique addresses this problem by applying the Arnoldi method to $(\mathcal{A} - \sigma I)^{-1}$, which maps

Algorithm 4 P-Arnoldi: Matrix-vector product Ub

Input: Matrices $U_0 = [u_1, \ldots, u_k]$, H, vector u partitioned as in (14), vector b, scalar β

Output: Ub

1: Build matrices T and $\tilde{T}$ in (23)
2: Compute $[v_1, \ldots, v_{d-1}] := e_k^T \tilde{T}$
3: Set

$$C = \begin{bmatrix} v_1 & v_2 & v_3 & \cdots & v_{d-1} \\ & 2v_1 & 2v_2 & \cdots & 2v_{d-2} \\ & & 2v_1 & \ddots & \vdots \\ & & & \ddots & 2v_2 \\ & & & & 2v_1 \end{bmatrix}$$

4: Return $\mathrm{vec}(U_0 T + [u_0, \ldots, u_{d-2}]C)$

the eigenvalues closest to σ to largest magnitude eigenvalues. In this section, we focus on the particular case $\sigma = 0$ and comment on the case $\sigma \neq 0$ in Section 4.3.

After k steps of the Arnoldi method applied to $\mathcal{A}^{-1}$ we obtain a relation of the form

$$\mathcal{A}^{-1} U = UH + \beta u e_k^*. \tag{24}$$

Multiplying with $\mathcal{A}$ and H^{-1} from both sides to (24) yields

$$\mathcal{A}U = UH^{-1} + \beta \tilde{u} \tilde{e}^* \tag{25}$$

with $\tilde{u} = -\mathcal{A}u$ and $\tilde{e} = H^{-*} e_k$. This has (nearly) the same form as (12) and an obvious extension of Lemma 2 can be applied. Hence, the relation (25) makes it possible to extend the P-Arnoldi to this situation. Note that the extraction of Ritz pairs is performed in the standard way, i.e., based on (24). When performing experiments with the resulting P-Arnoldi method for $\mathcal{A}^{-1}$, we encountered numerical difficulties for larger d. These difficulties are due to the fact that the norms of $\tau_j(H^{-1})$ grow quickly if H^{-1} has eigenvalues too far away from the interval $[-1, 1]$. Although in applications involving larger d one is normally only interested in eigenvalues on or close to that interval, this does not prevent some Ritz values from being far away from $[-1, 1]$. To some extent, these problems can be addressed by using a restarting strategy that purges all such Ritz values. However, it turns out that there is a more elegant solution, which will be discussed in this section.

4.1 Two-level orthogonalization Arnoldi (TOAR)

The instability of the P-Arnoldi method is due to the presence of $\tau_j(H^{-1})$ in the representation of the orthonormal Arnoldi basis U. In the following, we will extend the TOAR method [19], which avoids the use of such polynomials and thus preserves stability, to polynomials in Chebyshev basis. More specifically, TOAR uses the representation

$$[U, u] = \begin{bmatrix} QV_0 & Qv_0 \\ \vdots & \vdots \\ QV_{d-1} & Qv_{d-1} \end{bmatrix} = \begin{bmatrix} Q & & \\ & \ddots & \\ & & Q \end{bmatrix} \begin{bmatrix} V & v \end{bmatrix}, \tag{26}$$

with $Q \in \mathbb{C}^{n\times(d+k)}$, $Q^*Q = I$, and $V \in \mathbb{C}^{d(d+k)\times k}$, $[V, v]^*[V, v] = I$. Note that it may happen that the number of columns of Q becomes less than $d + k$ in the course of the method , due to the removal of linearly dependent vectors. For simplicity, we neglect this possibility in the following description and refer to [19] for details.

Similar to the previous section, we will now discuss the efficient implementation of steps 4–6 of the Arnoldi method (Algorithm 1) applied to $\mathcal{A}^{-1}$, and the expansion of the basis representation (26).

Matrix-vector product. To compute $w = \mathcal{A}^{-1}u$ with u represented as in (26), we have to solve the linear system

$$\begin{bmatrix} w_0 \\ w_1 \\ \vdots \\ w_{d-2} \\ w_{d-1} \end{bmatrix} = \begin{bmatrix} 0 & I & & & \\ I & 0 & I & & \\ & \ddots & \ddots & \ddots & \\ & & I & 0 & I \\ -A_0 & \cdots & -A_{d-3} & A_d - A_{d-2} & -A_{d-1} \end{bmatrix}^{-1} \begin{bmatrix} Qv_0 \\ 2Qv_1 \\ \vdots \\ 2Qv_{d-2} \\ 2A_dQv_{d-1} \end{bmatrix}.$$

This can be solved with the recurrences described in [9, §2.3]. For odd indices, the recurrence takes the form

$$w_1 = Qv_0, \quad w_{2j+1} = 2Qv_{2j} - w_{2j-1}, \quad j = 1, 2, \ldots, \tag{27a}$$

while for even indices

$$w_{2j} = y_{2j-1} + (-1)^j w_0, \quad j = 1, 2, \ldots, \tag{27b}$$

where

$$y_1 = 2Qv_1, \quad y_{2j+1} = 2Qv_{2j+1} - y_{2j-1}, \quad j = 1, 2, \ldots, \tag{27c}$$

and w_0 is obtained from solving the linear system

$$\begin{aligned}(-A_0 + A_2 - A_4 + A_6 - \cdots)w_0 \;\; = \;\; & (A_1w_1 + A_3w_3 + A_5w_5 + \cdots) \\ & +(A_2y_1 + A_4y_3 + A_6y_5 + \cdots).\end{aligned}$$

For this purpose the LU decomposition of $M = (-A_0 + A_2 - A_4 + A_6 - \cdots)$ is precomputed once and for all. As we will see below, only the first block w_0 is needed in the subsequent steps of the Arnoldi method. This results in a (minor) computational saving, see Algorithm 5. On the other hand, there is some overhead involved with reconstructing the vector u from Q and v. Fortunately, this computation can be arranged in a way such that a single matrix-matrix multiplication is sufficient, see step 5 of Algorithm 5.

Computation of U^*w. The first step of the orthogonalization procedure consists of computing $U^*w = V^*(I \otimes Q)^*w$, where $\otimes$ denotes the usual Kronecker product. Taking into account the presence of Q in the recurrences (27), the following lemma derives analogous recurrences for the components of $\hat{w} = (I \otimes Q)^*w$.

Lemma 3. *For given $v_0, \ldots, v_{d-1}$ and $\hat{w}_0 = Q^*w_0 \in \mathbb{C}^k$, let the vectors $\hat{w}_i$, $i = 1, \ldots, d-1$ be defined by the recurrences*

$$\hat{w}_1 = v_0, \quad \hat{w}_{2j+1} = 2v_{2j} - \hat{w}_{2j-1}, \quad j = 1, 2, \ldots, \tag{28a}$$

$$\hat{w}_{2j} = \hat{y}_{2j-1} + (-1)^j \hat{w}_0, \quad j = 1, 2, \ldots, \tag{28b}$$

$$\hat{y}_1 = 2v_1, \quad \hat{y}_{2j+1} = 2v_{2j+1} - \hat{y}_{2j-1}, \quad j = 1, 2, \ldots. \tag{28c}$$

Algorithm 5 TOAR: Matrix-vector product with $\mathcal{A}^{-1}$

Input: Matrices $\mathcal{A}$ and Q, input vector v.
Output: First component w_0 of $w = \mathcal{A}^{-1}u$, where $u = (I \otimes Q)v$.

Set $Y \leftarrow \begin{bmatrix} v_0, 2v_1 \end{bmatrix}$
for $i = 2, 3, \ldots, d-2$ **do**
  Update $Y \leftarrow \begin{bmatrix} Y, \, 2v_i - y_{i-2} \end{bmatrix}$
end for
Compute $Z \leftarrow QY$
Compute $\overline{w} \leftarrow -A_d z_{d-2} + \sum_{i=1}^{d-1} A_i z_{i-1}$
if d is even **then**
  Update $\overline{w} \leftarrow \overline{w} + A_d(2A_d Q v_{d-1} - z_{d-3})$
end if
Solve linear system $w_0 \leftarrow M^{-1}\overline{w}$ with $M = \sum_{j=0}^{\lfloor (d-2)/2 \rfloor} (-1)^{(j+1)} A_{2i}$

Then the relations

1. $w_i = Q\hat{w}_i$ *and* $y_i = Q\hat{y}_i$ *hold for all* odd i*;*
2. $w_i = Q\hat{w}_i + (-1)^{i/2}(I - QQ^*)w_0$ *holds for all* even i*;*

where w_i *and* y_i *are defined by the recurrences* (27)*. In particular,* $\hat{w}_i = Q^* w_i$ *for* $i = 1, \ldots, d-1$*.*

Proof. 1. We prove the first part by induction over i. The case $i = 1$ immediately follows from the definition of the recurrences. Now suppose that $w_i = Q\hat{w}_i$ and $y_i = Q\hat{y}_i$ hold for $i = 2j - 1$. Then (27a) implies

$$w_{i+2} = w_{2j+1} = 2Qv_{2j} - w_{2j-1} = Q(2v_{2j} - \hat{w}_{2j-1}) = Q\hat{w}_{i+2},$$

where we have used (28a). Similarly, (27c) implies

$$y_{i+2} = y_{2j+1} = 2Qv_{2j+1} - y_{2j-1} = Q(2v_{2j+1} - \hat{y}_{2j-1}) = Q\hat{y}_{i+2}.$$

2. The case $i = 0$ follows from rearranging the relation $Q\hat{w}_0 = QQ^*w_0 = w_0 - (I - QQ^*)w_0$. For $i = 2j$ with $j \geq 1$, we obtain from (27b), (28b), and the relation $y_{2j-1} = Q\hat{y}_{2j-1}$ from the first part that

$$\begin{aligned} Q\hat{w}_{2j} &= Q\hat{y}_{2j-1} + (-1)^j Q\hat{w}_0 = y_{2j-1} + (-1)^j \left(w_0 - (I - QQ^*)w_0\right) \\ &= w_{2j} - (-1)^j (I - QQ^*)w_0, \end{aligned}$$

which concludes the proof of the second part. □

After w_0 has been computed by Algorithm 5, the recurrences (28) of Lemma 3 allow us to compute the orthogonalization coefficients $h_j = \begin{bmatrix} U, u \end{bmatrix}^* w$ cheaply:

$$h_j = \begin{bmatrix} V & v \end{bmatrix}^* \begin{bmatrix} Q & & \\ & \ddots & \\ & & Q \end{bmatrix}^* \begin{bmatrix} w_0 \\ \vdots \\ w_{d-1} \end{bmatrix} = \begin{bmatrix} V & v \end{bmatrix}^* \begin{bmatrix} \hat{w}_0 \\ \vdots \\ \hat{w}_{d-1} \end{bmatrix}. \tag{29}$$

Expansion of basis. From the discussion above, it is already clear that the vector w_0 carries all the information needed for expanding the basis. The following theorem formalizes this statement.

Theorem 4. *Suppose that* $\big[U, u\big]$ *satisfies the Arnoldi decomposition* (24) *and is partitioned as in* (14)*. Moreover, let* Q *be an orthonormal basis of* $span\big\{U_0, u_0, U_1, u_1, \dots, U_{d-1}, u_{d-1}\big\}$*. Consider the vector* $\tilde{u}$ *that is obtained from orthogonalizing* $w = \mathcal{A}^{-1}u$ *against* $span\big\{U, u\big\}$*. Then*

$$span\big\{U_0, u_0, \tilde{u}_0, U_1, u_1, \tilde{u}_1, \dots, U_{d-1}, u_{d-1}, \tilde{u}_{d-1}\big\} = span\{Q, w_0\},$$

where w_0 *is the first block of* w*.*

Proof. Inspecting the recursions (27), the vectors $w_1, \dots, w_{d-1}$ are obtained as linear combinations of w_0 and vectors from $\mathrm{span}\{Q\}$. We therefore have

$$\mathrm{span}\big\{U_0, u_0, w_0, U_1, u_1, w_1, \dots, U_{d-1}, u_{d-1}, w_{d-1}\big\} = \mathrm{span}\{Q, w_0\},$$

which completes the proof because replacing w by $\tilde{u} = w - \big[U, u\big]\big[U, u\big]^* w$ does not change the space on the left. □

According to Theorem 4, we expand Q with a new vector q computed as $\alpha q = w_0 - Q\hat{w}_0$, where $\hat{w}_0 = Q^* w_0$ and $\alpha = \|w_0 - Q\hat{w}_0\|_2$. The next step in the Arnoldi process consists of completing the orthogonalization of w:

$$\tilde{u} = w - [U, u]h_j = \begin{bmatrix} w_0 \\ \vdots \\ w_{d-1} \end{bmatrix} - \begin{bmatrix} Q & & \\ & \ddots & \\ & & Q \end{bmatrix} \begin{bmatrix} V & v \end{bmatrix} h_j. \tag{30}$$

For this purpose, we first compute the vector s resulting from Gram-Schmidt orthogonalization of $\hat{w} = (I \otimes Q)^* w$ against $[V, v]$:

$$\begin{bmatrix} s_0 \\ \vdots \\ s_{d-1} \end{bmatrix} = \begin{bmatrix} \hat{w}_0 \\ \vdots \\ \hat{w}_{d-1} \end{bmatrix} - \begin{bmatrix} V & v \end{bmatrix} h_j. \tag{31}$$

Using the relations of Lemma 3, the blocks of (30) with odd index satisfy

$$\tilde{u}_{2j+1} = w_{2j+1} - Q(\hat{w}_{2j+1} - s_{2j+1}) = Qs_{2j+1},$$

while the blocks with even index satisfy

$$\tilde{u}_{2j} = w_{2j} - Q(\hat{w}_{2j} - s_{2j}) = Qs_{2j} + (-1)^j(I - QQ^*)w_0 = \begin{bmatrix} Q & q \end{bmatrix} \begin{bmatrix} s_{2j} \\ (-1)^j\alpha \end{bmatrix}.$$

Therefore, the orthogonal matrix $[V, v]$ is expanded with the vector

$$\tilde{v} = \big[s_0^*, \alpha, s_1^*, 0, s_2^*, -\alpha, s_3^*, 0, \dots\big]^*$$

after normalization. Note that $\|\tilde{v}\|_2 = \|\tilde{u}\|_2 =: h_{j+1,j}$.

Algorithm 6 Two-level orthogonalization Arnoldi (TOAR) method for $\mathcal{A}^{-1}$

Input: Matrix $\mathcal{A}$, starting vector $\overline{u} \neq 0$, $k \in \mathbb{N}$

Output: Matrices Q, $\begin{bmatrix} V, v \end{bmatrix}$ such that the matrix $\begin{bmatrix} U, u \end{bmatrix}$ defined in (26) is an orthonormal basis of $\mathcal{K}_{k+1}(\mathcal{A}^{-1}, \overline{u})$

1: Normalize $u \leftarrow \overline{u}/\|\overline{u}\|_2$
2: Compute QR factorization $\begin{bmatrix} u_0, u_1, \ldots, u_{d-1} \end{bmatrix} = QR$ with $R = [r_0, \ldots, r_{d-1}] \in \mathbb{R}^{d \times d}$
3: Set $V \leftarrow [\,]$ and $v \leftarrow \begin{bmatrix} r_0^*, r_1^*, \ldots, r_{d-1}^* \end{bmatrix}^*$
4: **for** $j = 1, 2, \ldots, k$ **do**
5: Compute w_0 using Algorithm 5
6: Compute $\hat{w}_0 \leftarrow Q^* w_0$ and $\widetilde{q} \leftarrow w_0 - Q\hat{x}_0$
7: Normalize $\alpha \leftarrow \|\widetilde{q}\|_2, \quad q \leftarrow \widetilde{q}/\alpha$
8: Update $Q \leftarrow \begin{bmatrix} Q, q \end{bmatrix}$
9: Compute $\hat{w}_i$ for $i = 1, \ldots, d-1$ using the recurrences (28) and set $\hat{w} \leftarrow \begin{bmatrix} \hat{w}_0^*, \ldots, \hat{w}_{d-1}^* \end{bmatrix}^*$
10: Compute $h_j \leftarrow \begin{bmatrix} V, v \end{bmatrix}^* \hat{w}$ and $s \leftarrow \hat{w} - \begin{bmatrix} V, v \end{bmatrix} h_j$
11: Update components of V: $V_0 \leftarrow \left[\begin{smallmatrix} V_0 & v_0 \\ 0 & 0 \end{smallmatrix}\right]$, $V_1 \leftarrow \left[\begin{smallmatrix} V_1 & v_1 \\ 0 & 0 \end{smallmatrix}\right]$, …, $V_{d-1} \leftarrow \left[\begin{smallmatrix} V_{d-1} & v_{d-1} \\ 0 & 0 \end{smallmatrix}\right]$
12: Set $\tilde{v} = \begin{bmatrix} s_0^*, \alpha, s_1^*, 0, s_2^*, -\alpha, s_3^*, 0, \ldots \end{bmatrix}^*$
13: Normalize $v \leftarrow \tilde{v}/h_{j+1,j}$ with $h_{j+1,j} = \|\tilde{v}\|_2$
14: **end for**

Summary. The considerations above lead to the TOAR method for $\mathcal{A}^{-1}$ summarized in Algorithm 6. The first and second level orthogonalizations are carried out in line 6 and 10, respectively. In both cases, reorthogonalization can (and should) be done if required. Apart from the memory savings, Algorithm 6 also entails a significant increase of computational efficiency compared to standard Arnoldi or P-Arnoldi for larger d. This is due to fact that orthogonalization is much cheaper, with the main cost being concentrated in the first level, which involves only the matrix Q. Although the second level orthogonalization causes some overhead, its cost remains negligible for $n \gg k$.

4.2 Krylov-Schur Restart

In practical implementations of Arnoldi methods, it is fundamental to incorporate an effective restarting strategy. In the following, we will propose such a strategy for TOAR based on the Krylov-Schur method, by adapting and extending the discussion on implicit restarting TOAR for the monomial representation in [27].

The Krylov-Schur method [26] is based on generalizing the Arnoldi decomposition (24) to a Krylov decomposition

$$\mathcal{A}^{-1} U = UB + ub^*, \tag{32}$$

where B is a general $k \times k$ matrix (not necessarily in Hessenberg form) and b is a general vector of length k.

To restart the Krylov decomposition (32), we first compute the Schur form $B = YSY^*$ with an upper triangular matrix S and a unitary matrix Y. Applying this transformation to (32) yields

$$\mathcal{A}^{-1} \tilde{U} = \tilde{U} S + u\tilde{b}^*, \tag{33}$$

where $\tilde{U} = UY$ and $\tilde{b} = Y^*b$. The ordering of the eigenvalues in the Schur form S is such that the $p < k$ Ritz values of interest are contained in its leading principal $p \times p$ submatrix $\check{S}$. We then truncate (33) to

$$\mathcal{A}^{-1}\check{U} = \check{U}\check{S} + u\check{b}^*, \tag{34}$$

with $\check{U} = U\check{Y}$, where $\check{Y}$ contains the first p columns of Y and $\check{b}$ contains the first p entries of $\tilde{b}$. The Arnoldi process is continued from (34) by expanding the decomposition again to order k, truncating it back to order p, and so on.

Let us now consider the case that the decomposition (32) has been generated with Algorithm 6, yielding the representation (26) of $\big[U, u\big]$. Then the components of U satisfy $U_j = QV_j$ with orthonormal $Q \in \mathbb{C}^{n\times(d+k)}$ and $V_j \in \mathbb{C}^{(d+k)\times k}$. In principle, the restarting procedure described above can be implemented in a straightforward fashion, by simply keeping only the first p columns of V_jY. However, this would not effect the desired reduction as the (usually much larger) factor Q remains of the same size.

In order to also truncate Q, let us recall that Theorem 4 implies that $\big[U_0, u_0, U_1, u_1, \ldots, U_{d-1}, u_{d-1}\big]$ and consequently also $\big[V_0, v_0, V_1, v_1, \ldots, V_{d-1}, v_{d-1}\big]$ have rank $d+k$ only. Similarly, the matrix

$$\big[V_0\check{Y}, v_0, V_1\check{Y}, v_1, \ldots, V_{d-1}\check{Y}, v_{d-1}\big] \tag{35}$$

has rank $d+p$. This is due to its relation to the truncated Krylov-Schur decomposition (34), which can be retransformed into a standard Arnoldi decomposition [26] of order p (with different starting vector) and hence Theorem 4 applies. By, e.g., a truncated SVD, we can find an orthonormal matrix $W \in \mathbb{C}^{(d+k)\times(d+p)}$ such that its columns span the row space of the matrix in (35). If we now define

$$\check{Q} = QW, \quad \check{V}_j = W^*V_j\check{Y}, \quad \check{v}_j = W^*v_j, \quad j = 0, \ldots, d-1,$$

then we obtain the following representation of the truncated Arnoldi basis $\check{U}$:

$$[\check{U}, u] = \begin{bmatrix} \check{Q}\check{V}_0 & \check{Q}\check{v}_0 \\ \vdots & \vdots \\ \check{Q}\check{V}_{d-1} & \check{Q}\check{v}_{d-1} \end{bmatrix}.$$

This is exactly the form of the usual representation (26) in the TOAR method, allowing us to expand the truncated Arnoldi decomposition (34) using the techniques of Algorithm 6.

4.3 Extension to $\sigma \neq 0$

In this section, we briefly discuss the extension of the TOAR method described above to $\sigma \neq 0$, that is, the shift-and-invert Arnoldi method operating with $(\mathcal{L}_0 - \sigma\mathcal{L}_1)^{-1}\mathcal{L}_1$ for a nonzero shift σ. For this purpose, it is useful to note that the recurrences (27), which provided the foundation of TOAR, can be derived from a block LU factorization of the matrix $\mathcal{L}_0$.

In the following, we derive a generalization of the recurrences (27) from a block LU factorization of the matrix

$$\mathcal{L}_0 - \sigma\mathcal{L}_1 = \begin{bmatrix} -\sigma I & I & & & & \\ I & -2\sigma I & I & & & \\ & I & -2\sigma I & I & & \\ & & \ddots & \ddots & \ddots & \\ & & & I & -2\sigma I & I \\ -A_0 & -A_1 & \cdots & -A_{d-3} & A_d - A_{d-2} & -A_{d-1} - 2\sigma A_d \end{bmatrix}. \tag{36}$$

For this purpose, we proceed in a way analogous to the proof of [2, Theorem 3.1]. We first permute the block columns as follows:

$$(\mathcal{L}_0 - \sigma\mathcal{L}_1)\Pi = \begin{bmatrix} I & & & & & -\sigma I \\ -2\sigma I & I & & & & I \\ I & -2\sigma I & I & & & \\ & \ddots & \ddots & \ddots & & \\ & & I & -2\sigma I & I & \\ -A_1 & \cdots & -A_{d-3} & A_d - A_{d-2} & -A_{d-1} - 2\sigma A_d & -A_0 \end{bmatrix},$$

with the block cyclic shift $\Pi = \left[\begin{smallmatrix} 0 & I_n \\ I_{(d-1)n} & 0 \end{smallmatrix}\right]$. This matrix has the block LU factorization $L_\sigma U_\sigma$ with

$$L_\sigma = \begin{bmatrix} I & & & & & \\ -2\sigma I & I & & & & \\ I & -2\sigma I & I & & & \\ & \ddots & \ddots & \ddots & & \\ & & I & -2\sigma I & I & \\ -A_1 & \cdots & -A_{d-3} & A_d - A_{d-2} & -A_{d-1} - 2\sigma A_d & P(\sigma) \end{bmatrix}$$

and

$$U_\sigma = \begin{bmatrix} I & & & & -\tau_1(\sigma) I \\ & I & & & -\tau_2(\sigma) I \\ & & I & & -\tau_3(\sigma) I \\ & & & \ddots & \vdots \\ & & & I & -\tau_{d-1}(\sigma) I \\ & & & & I \end{bmatrix}.$$

Hence,

$$(\mathcal{L}_0 - \sigma\mathcal{L}_1)^{-1} = \Pi U_\sigma^{-1} L_\sigma^{-1}. \tag{37}$$

Note that U_σ^{-1} is identical with U_σ, except for a sign change in the first $d-1$ blocks of the last block column.

Using (37) and the block structure of L_σ, U_σ, we will now discuss the computation of $w = (\mathcal{L}_0 - \sigma\mathcal{L}_1)^{-1}\mathcal{L}_1 u$ for a vector $u = \left[u_0^*, u_1^*, \cdots, u_{d-1}^*\right]^*$. First, the components $y_0, y_1, \ldots, y_{d-1} \in \mathbb{C}^n$ of

$$y := L_\sigma^{-1}\mathcal{L}_1 u$$

are given by

$$\begin{aligned} y_0 &= u_0 \\ y_1 &= 2u_1 + 2\sigma y_0 \\ y_2 &= 2u_2 + 2\sigma y_1 - y_0 \\ &\vdots \\ y_{d-2} &= 2u_{d-2} + 2\sigma y_{d-3} - y_{d-4} \\ y_{d-1} &= w_0, \end{aligned}$$

with

$$w_0 := P(\sigma)^{-1}\left[2A_d u_{d-1} + A_1 y_0 + \cdots + A_{d-3} y_{d-4} + (A_{d-2} - A_d) y_{d-3} + (A_{d-1} + 2\sigma A_d) y_{d-2}\right].$$

Using that

$$w = \Pi U_\sigma^{-1} y,$$

this yields the following expressions for its components $w_0, w_1, \ldots, w_{d-1} \in \mathbb{R}^n$:

$$\begin{aligned} w_0 &= w_0 \\ w_1 &= u_0 + \tau_1(\sigma) w_0 \\ w_2 &= 2u_1 + 2\sigma y_0 + \tau_2(\sigma) w_0 \\ w_3 &= 2u_2 + 2\sigma y_1 - y_0 + \tau_3(\sigma) w_0 \\ &\vdots \\ w_{d-1} &= 2u_{d-2} + 2\sigma y_{d-3} - y_{d-4} + \tau_{d-1}(\sigma) w_0. \end{aligned}$$

These formulas can be used in place of the recurrences (27) to develop a TOAR method for nonzero σ. In particular, when applying $(\mathcal{L}_0 - \sigma\mathcal{L}_1)^{-1}\mathcal{L}_1$ to a vector u represented as in (26), it is possible to pull out the matrix Q and derive recurrences for the reduced vectors analogous to (28), Apart from these changes, there is only one additional modification required to adapt Algorithm 6: the vector $\tilde{v}$ generated in step 12 of Algorithm 6 to expand V now takes the form

$$\tilde{v} = \begin{bmatrix} s_0 \\ \tau_0(\sigma)\alpha \\ s_1 \\ \tau_1(\sigma)\alpha \\ \vdots \\ s_{d-1} \\ \tau_{d-1}(\sigma)\alpha \end{bmatrix}. \tag{38}$$

Remark 5. *The linearizations discussed in [2] for general polynomial bases satisfying a three-term recurrence relation all admit a highly structured block LU factorization similar to the one discussed above. It is therefore likely that the TOAR method can be extended to this more general setting. However, we feel that a detailed discussion would go beyond the scope of this paper.*

5 Numerical experiments

We have evaluated the performance of the shift-and-invert TOAR method from Section 4 on three application problems. Our examples arise from certain discretizations of elliptic PDE eigenvalue problems, leading to a nonlinear eigenvalue problem

$$T(\lambda)x = 0, \qquad x \neq 0, \tag{39}$$

where $T : \mathbb{C} \to \mathbb{C}^{n\times n}$ is an analytic matrix-valued function. The solution of this problem can be approximated by replacing $T(\lambda)$ with a polynomial $P(\lambda)$ interpolating T at $d+1$ interpolation points. In particular, when the eigenvalues on or very close to a real interval are of interest, interpolation in the Chebyshev points can be used. As discussed in [9], the resulting polynomial P of degree d can be easily represented in the Chebyshev basis. After evaluating $T(\lambda)$ for $d+1$ values of λ, the rest of the computation proceeds with $P(\lambda)$. This

gives an advantage, compared to solvers directly addressing (39), when the evaluation of $T(\lambda)$ is very costly.

The TOAR method applied to the linearization of P results in an approximate invariant pair (V, Λ), where V has the form (10). As discussed in Section 2, we can then extract an approximate invariant pair (X, Λ) for P. Subsequently, we apply 1 or 2 iterations of Newton's method to refine (X, Λ). In all our experiments below, this is sufficient to attain an accuracy on the level of machine precision.

All computations have been carried out under MATLAB R2012a on a computer with an Intel Core i7 processor at 2.6 GHz with 8 GB of memory.

5.1 Laplace eigenvalue problem on the Fichera corner

We first address the 3D Laplace eigenvalue problem with Dirichlet boundary conditions [24]

$$\begin{aligned} -\Delta u &= \lambda^2 u \quad \text{in } \mathcal{D} \subset \mathbb{R}^3, \\ u &= 0 \qquad \text{on } \mathcal{B} := \partial\mathcal{D}, \end{aligned}$$

which can be reformulated by means of the representation formula for the Helmholtz operator as the boundary integral equation

$$\frac{1}{4\pi} \int_{\mathcal{B}} \frac{e^{i\lambda\|\xi-\eta\|}}{\|\xi-\eta\|} u_n(\eta) dS(\eta) = 0 \qquad \text{for all } \xi \in \mathcal{B},$$

where u_n is the exterior normal derivative of u. A Galerkin-based discretization eventually leads to a nonlinear eigenvalue problem (39). We report results for the Fichera corner $\mathcal{D} = [0,1]^3 \backslash [\frac{1}{2}, 1]^3$ using a boundary element space of piecewise constant functions on a surface mesh of 2400 triangles ($h = 0.1$).

We have run the method of section 4 for various degrees d of the interpolating Chebyshev polynomial, from $d = 3$ up to $d = 30$, and compared it with applying the standard Krylov-Schur algorithm. The solvers were configured to compute 6 eigenvalues with a maximum subspace dimension of $k = 24$ and a convergence tolerance 10^{-12} for the norm of the residual. The reported results are averaged over 20 runs with random initial vectors.

Figure 1 confirms the spectral convergence of the smallest computed eigenvalue as the interpolation degree increases. As expected, both TOAR and Krylov-Schur behave similarly, and full machine precision is achieved already for a polynomial of degree 17.

TOAR and Krylov-Schur behave very similarly in terms of the number of required restarts and execution time, see Figure 2. The latter is due to the fact that the computational cost is dominated by operations involving the coefficients A_i of the polynomial. Our current implementation does not make use of fast techniques for boundary element discretizations, like hierarchical matrices, and treats A_i as dense matrices. We expect that incorporating such a fast technique significantly decreases the time spent on operations with A_i. This likely results in a more pronounced difference between the execution times for TOAR and Krylov-Schur. This can be seen from Table 1, where we have accounted for the times spent on the initial LU factorization of $P(0)$, on the matrix-vector products $w = \mathcal{A}^{-1}u$ and on all other operations separately. It turns out that Krylov-Schur takes 0.20 seconds per restart, whereas TOAR takes only 0.07 seconds, when neglecting the times for the LU factorization and the matrix-vector products. However, we would like to emphasize that the major computational saving is in the additional memory required for storing the basis vectors; this is reduced by a factor of approximately d when using TOAR.

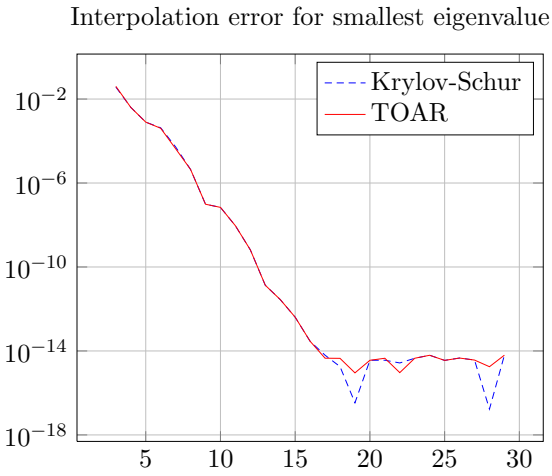

Figure 1: Absolute error between the smallest eigenvalues of the nonlinear and the interpolating polynomial eigenvalue problem, computed by Krylov-Schur and TOAR, for polynomial degrees $d = 3, 4, \ldots$.

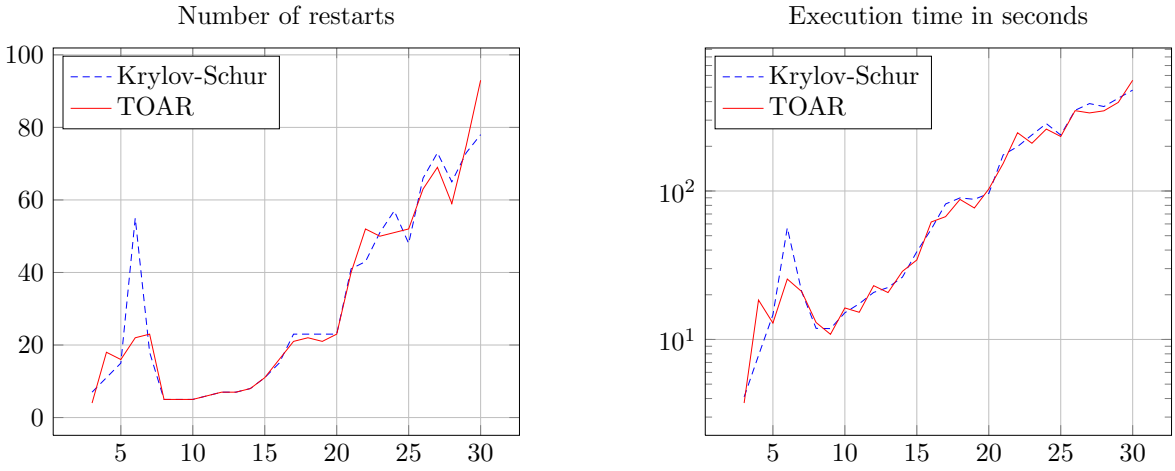

Figure 2: Comparison of Krylov-Schur and TOAR in terms of number of restarts (left) and execution time (right) for polynomial degrees $d = 3, 4 \ldots$.

Table 1: Execution times (in seconds) and number of restarts for the BEM discretization from Section 5.1 with polynomial degree 20.

	LU factorization	Products $\mathcal{A}^{-1}u$	Other operations	Restarts
Krylov-Schur	0.57	97.25	5.11	25
TOAR	0.68	88.80	1.41	21

Table 2: Execution times (in seconds) and number of restarts for the fluid-structure interaction problem with polynomial degree 11, when computing 12 eigenvalues with a basis of 48 vectors.

	LU factorization	Products $\mathcal{A}^{-1}u$	Other operations	Restarts
Krylov-Schur	69.10	134.66	7.54	2
TOAR	63.27	130.64	0.7	2

5.2 Fluid-structure interaction

As a second example for a nonlinear eigenvalue problem (39), we consider the mixed finite element/boundary element discretization of a 3D fluid-structure interaction problem discussed in [10]. This results in block structured coefficients, where the diagonal block corresponding to the finite element discretization is sparse. Table 2 reports the execution times on a discretization with $n = 6223$ and polynomial degree 11.

5.3 Loaded string

Finally, we apply our algorithm to the rational eigenvalue problem `loaded_string` from the NLEVP benchmark collection [**?**]. In this case,

$$T(\lambda) = A - \lambda B + \frac{\lambda}{\lambda - 1} C$$

for $n \times n$ symmetric positive definite tridiagonal matrices A, B, and $C = e_n e_n^T$. We are interested in the six smallest eigenvalues to the right of the pole 1 and perform Chebyshev interpolation on the interval $[4, 400]$. Again, Krylov-Schur and TOAR are applied to the resulting polynomial eigenvalue problem $P(\lambda)$ in Chebyshev basis, with a maximum subspace dimension of $k = 32$ and a convergence tolerance 10^{-12}. Compared to the examples from Sections 5.1 and 5.2, the cost for performing matrix-vector products $\mathcal{A}^{-1}u$ is much less dominant, due to the tridiagonal structure of $P(\lambda)$. This effect can be clearly seen in the (total) execution times reported in Figure 3 for $n = 1\,000$ and $n = 10\,000$. TOAR now visibly outperforms Krylov-Schur; for $n = 10\,000$ it is up to an order of magnitude faster. The times obtained for $n = 1\,000$ are somewhat erratic, especially for larger degree, due to the variance in the number of required restarts. To mitigate this effect, Figure 4 additionally shows the execution time per restart.

For this problem, TOAR also requires significantly less memory. For example, for $n = 10\,000$ and $d = 20$, TOAR consumes 14 Mbyte while Krylov-Schur consumes 73 Mbyte in total.

6 Conclusions

We have presented two Arnoldi methods, P-Arnoldi and TOAR, for the solution of polynomial eigenvalue problems expressed in the Chebyshev basis. The methods represent a significant reduction of storage requirements, compared to standard Arnoldi applied to the linearization, since the generated basis consists of vectors of length n rather than vectors of full length dn, where d is the degree of the polynomial. At the moment, the P-Arnoldi method remains a curiosity, at least for numerical computations. Although it can be formulated in an elegant way, its instability for Ritz values not close to $[-1, 1]$ makes it unsuitable for handling polynomials

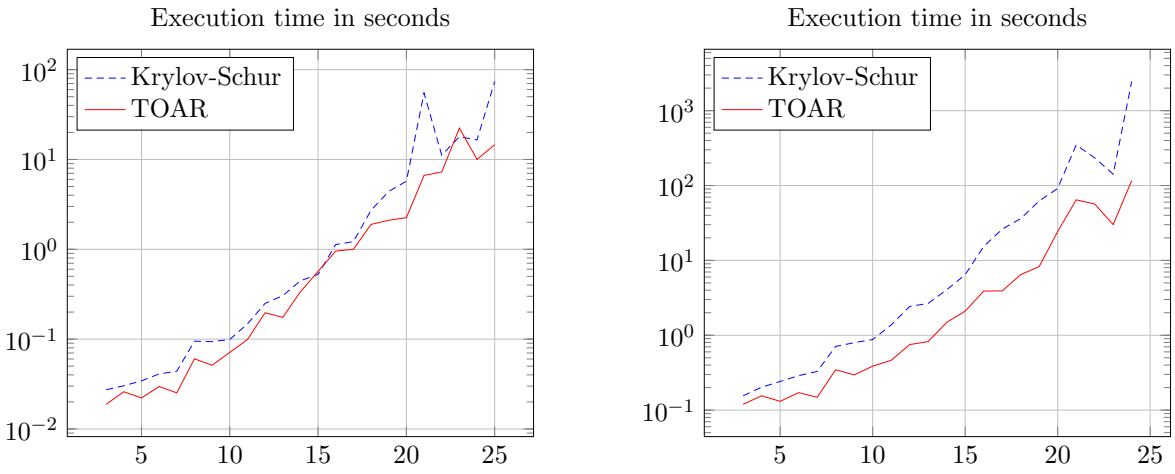

Figure 3: Comparison of execution time of Krylov-Schur and TOAR when solving the loaded string problem for $n = 1\,000$ (left) and $n = 10\,000$ (right) for polynomial degrees $d = 3, 4 \ldots$.

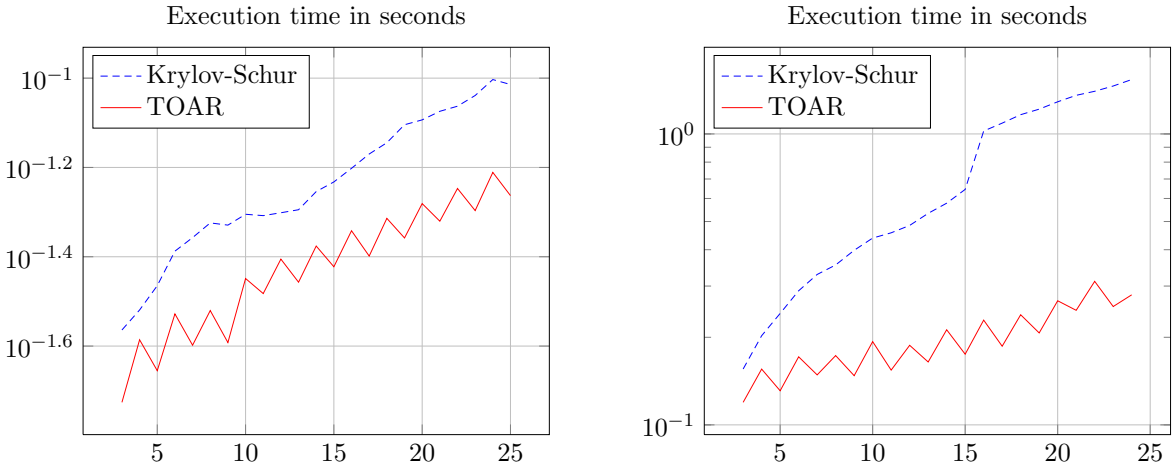

Figure 4: Comparison of execution time per restart of Krylov-Schur and TOAR when solving the loaded string problem for $N = 1\,000$ (left) and $N = 10\,000$ (right) for polynomial degrees $d = 3, 4 \ldots$.

of larger degree. In contrast, the TOAR method appears to be numerically stable even for comparably large degrees. Moreover, we have shown that it is possible to turn this into a fully fledged Arnoldi algorithm, by incorporating shift-and-invert and restarting techniques. All these ingredients are crucial for addressing polynomial eigenvalue problems that arise from the interpolation of discretized nonlinear PDE eigenvalue problems. Our numerical experiments demonstrate the advantages of our algorithm compared to the standard approach in terms of memory requirements and computational time.

As indicated in Remark 5, there is nothing particular about the Chebyshev basis; all algorithms can certainly be extended to more general polynomial bases satisfying a three-term recurrence. Another possible extension would consist of using the Krylov subspace methods developed in this paper for solving linear systems with $P(\lambda)$ for many different values of λ [12].

Acknowledgments

The authors thank Cedric Effenberger for many insightful discussions and for providing the matrices used in the numerical experiments. They also thank Yangfeng Su for sending a preprint of [19]. This work was carried out while the second author was visiting EPFL during the Spring 2013.

References

[1] B. Adhikari, R. Alam, and D. Kressner. Structured eigenvalue condition numbers and linearizations for matrix polynomials. *Linear Algebra Appl.*, 435(9):2193–2221, 2011.

[2] A. Amiraslani, R. M. Corless, and P. Lancaster. Linearization of matrix polynomials expressed in polynomial bases. *IMA J. Numer. Anal.*, 29(1):141–157, 2009.

[3] Z. Bai and Y. Su. SOAR: a second-order Arnoldi method for the solution of the quadratic eigenvalue problem. *SIAM J. Matrix Anal. Appl.*, 26(3):640–659, 2005.

[4] T. Betcke and D. Kressner. Perturbation, extraction and refinement of invariant pairs for matrix polynomials. *Linear Algebra Appl.*, 435(3):574–536, 2011.

[5] W.-J. Beyn and V. Thümmler. Continuation of invariant subspaces for parameterized quadratic eigenvalue problems. *SIAM J. Matrix Anal. Appl.*, 31(3):1361–1381, 2009.

[6] D. Bindel and A. Hood. Localization theorems for nonlinear eigenvalues. arXiv:1303.4668, 2013.

[7] M. A. Botchev, G. L. G. Sleijpen, and A. Sopaheluwakan. An SVD-approach to Jacobi-Davidson solution of nonlinear Helmholtz eigenvalue problems. *Linear Algebra Appl.*, 431(3-4):427–440, 2009.

[8] C. W. Clenshaw. A note on the summation of Chebyshev series. *Mathematics of Computation*, 9(51):118–120, 1955.

[9] C. Effenberger and D. Kressner. Chebyshev interpolation for nonlinear eigenvalue problems. *BIT*, 52(4):933–951, 2012.

[10] C. Effenberger, D. Kressner, O. Steinbach, and G. Unger. Interpolation-based solution of a nonlinear eigenvalue problem in fluid-structure interaction. *PAMM*, 12(1):633–634, 2012.

[11] G. H. Golub and C. F. Van Loan. *Matrix Computations*. Johns Hopkins University Press, Baltimore, MD, third edition, 1996.

[12] L. Grammont, N. J. Higham, and F. Tisseur. A framework for analyzing nonlinear eigenproblems and parametrized linear systems. *Linear Algebra Appl.*, 435(3):623–640, 2011.

[13] V. Hernandez, J. E. Roman, and V. Vidal. SLEPc: a scalable and flexible toolkit for the solution of eigenvalue problems. *ACM Trans. Math. Software*, 31(3):351–362, 2005.

[14] N. J. Higham, D. S. Mackey, and F. Tisseur. The conditioning of linearizations of matrix polynomials. *SIAM J. Matrix Anal. Appl.*, 28(4):1005–1028, 2006.

[15] N. Kamiya, E. Andoh, and K. Nogae. Eigenvalue analysis by the boundary element method: new developments. *Engng. Anal. Bound. Elms.*, 12:151–162, 1993.

[16] D. Kressner. A block Newton method for nonlinear eigenvalue problems. *Numer. Math.*, 114(2):355–372, 2009.

[17] R. B. Lehoucq, D. C. Sorensen, and C. Yang. *ARPACK users' guide*. SIAM, Philadelphia, PA, 1998. Solution of large-scale eigenvalue problems with implicitly restarted Arnoldi methods.

[18] Y. Lin, L. Bao, and Y. Wei. Model-order reduction of large-scale kth-order linear dynamical systems via a kth-order Arnoldi method. *Int. J. Comput. Math.*, 87(1-3):435–453, 2010.

[19] Y. Lu and Y. Su. Two-level orthogonal Arnoldi process for the solution of quadratic eigenvalue problems. Submitted.

[20] D. S. Mackey, N. Mackey, C. Mehl, and V. Mehrmann. Structured polynomial eigenvalue problems: good vibrations from good linearizations. *SIAM J. Matrix Anal. Appl.*, 28(4):1029–1051, 2006.

[21] D. S. Mackey, N. Mackey, C. Mehl, and V. Mehrmann. Vector spaces of linearizations for matrix polynomials. *SIAM J. Matrix Anal. Appl.*, 28(4):971–1004, 2006.

[22] K. Meerbergen. The quadratic Arnoldi method for the solution of the quadratic eigenvalue problem. *SIAM J. Matrix Anal. Appl.*, 30(4):1463–1482, 2008/09.

[23] G. A. Sitton, C. S. Burrus, J. W. Fox, and S. Treitel. Factoring very-high-degree polynomials. *IEEE Signal Processing Magazine*, 20(6):27–42, 2003.

[24] O. Steinbach and G. Unger. A boundary element method for the Dirichlet eigenvalue problem of the Laplace operator. *Numer. Math.*, 113(2):281–298, 2009.

[25] G. W. Stewart. *Matrix Algorithms. Vol. II*. SIAM, Philadelphia, PA, 2001. Eigensystems.

[26] G. W. Stewart. A Krylov-Schur algorithm for large eigenproblems. *SIAM J. Matrix Anal. Appl.*, 23(3):601–614, 2001/02.

[27] Y. Su. A compact Arnoldi algorithm for polynomial eigenvalue problems, 2008. Talk presented at RANMEP2008.

[28] L. N. Trefethen. *Approximation Theory and Approximation Practice*. SIAM, 2012.

[29] R. Van Beeumen, K. Meerbergen, and W. Michiels. A Rational Krylov Method Based on Hermite Interpolation for Nonlinear Eigenvalue Problems. *SIAM J. Sci. Comput.*, 35(1):A327–A350, 2013.