Int. J. Appl. Math. Comput. Sci., 2015, Vol. 25, No. 4, 769–785
DOI: 10.1515/amcs-2015-0055

amcs

THE NON–SYMMETRIC *s*–STEP LANCZOS ALGORITHM: DERIVATION OF EFFICIENT RECURRENCES AND SYNCHRONIZATION–REDUCING VARIANTS OF BiCG AND QMR

STEFAN FEUERRIEGEL [a,*], H. MARTIN BÜCKER [b]

[a] Chair for Information Systems Research
University of Freiburg, Platz der Alten Synagoge, 79098 Freiburg, Germany
e-mail: stefan.feuerriegel@is.uni-freiburg.de

[b] Chair for Advanced Computing
Friedrich Schiller University Jena, Ernst-Abbe-Platz 2, 07743 Jena, Germany
e-mail: martin.buecker@uni-jena.de

The Lanczos algorithm is among the most frequently used iterative techniques for computing a few dominant eigenvalues of a large sparse non-symmetric matrix. At the same time, it serves as a building block within biconjugate gradient (BiCG) and quasi-minimal residual (QMR) methods for solving large sparse non-symmetric systems of linear equations. It is well known that, when implemented on distributed-memory computers with a huge number of processes, the synchronization time spent on computing dot products increasingly limits the parallel scalability. Therefore, we propose synchronization-reducing variants of the Lanczos, as well as BiCG and QMR methods, in an attempt to mitigate these negative performance effects. These so-called s-step algorithms are based on grouping dot products for joint execution and replacing time-consuming matrix operations by efficient vector recurrences. The purpose of this paper is to provide a rigorous derivation of the recurrences for the s-step Lanczos algorithm, introduce s-step BiCG and QMR variants, and compare the parallel performance of these new s-step versions with previous algorithms.

Keywords: synchronization-reducing, s-step Lanczos, s-step BiCG, s-step QMR, efficient recurrences.

1. Rethinking algorithm design

Current large-scale computer systems are sophisticated architectures based on multi- or manycore technology, with deep memory hierarchies and possible heterogeneity in the form of graphic or other coprocessors. For scientific and engineering applications, it is therefore currently challenging to achieve a high performance using these systems. Unfortunately, future extreme-scale computer systems are likely to become even more complex and so it will become increasingly hard to achieve a sustained performance that is somewhere near peak performance. It is widely recognized that there are various intricate challenges for future large-scale computing. Today, there is only a vague idea of how these future platforms will actually be built and how they will be programmed efficiently. Rather than summarizing this ongoing discussion (Cappello *et al.*, 2009; Davis *et al.*, 2012; Duff, 2012; Shalf *et al.*, 2011), we focus on novel algorithmic techniques that will be required to fully exploit current large-scale and future exascale systems.

Existing technology trends indicate that algorithm designers will have to pay crucial attention in order to reduce data movement at various memory levels and to reduce synchronization at various system levels. While the communication costs associated with data movement have become an important issue in today's parallel algorithm design, the cost associated with synchronization does not currently receive adequate consideration. However, synchronization costs will soon outweigh communication costs as the degree of parallelism increases further. In fact, synchronization dictates the overall performance of various algorithms on current large-scale and, in particular, future exascale systems. Therefore, we address the problem of designing

*Corresponding author

synchronization-reducing variants of popular iterative Krylov subspace methods that are based on the Lanczos algorithm (Lanczos, 1950).

The contribution of the present paper is to derive a new variant of the s-step Lanczos algorithm (Kim and Chronopoulos, 1992) with a normalization scheme that improves numerical stability. In contrast to previous publications, which all lack details on how to obtain the different underlying recurrences, this paper provides a thorough derivation. Based on this s-step Lanczos algorithm, we then introduce synchronization-reducing variants of two Krylov methods for the solution of large sparse non-symmetric systems of linear equations. More precisely, we propose new synchronization-reducing variants of the biconjugate gradient (BiCG) method (Fletcher, 1976) and the quasi-minimal residual (QMR) one (Freund and Nachtigal, 1994), and assess their numerical stability and parallel scalability. This publication is an extended version of our previous conference papers (Feuerriegel and Bücker, 2013a; 2013b) with additional details on the derivation of the recurrences underlying the s-step Lanczos algorithm. We also present some new computational experiments.

In this article, we use the following notation. Given two vectors, their dot product $\boldsymbol{v}^T\boldsymbol{w}$ is denoted by $\langle \boldsymbol{v}, \boldsymbol{w} \rangle$. The zero vector of dimension n is given by $\mathbf{0}_n$. The symbols $0_{n,m}$ and $\mathrm{I}_{n,m}$ are used for $n \times m$ zero and identity matrices, respectively. Concatenation of scalar entries that form a row vector is denoted by $[x_1, \ldots, x_n]$. Concatenation of vectors or matrices that form a (block) matrix is indicated by $[\boldsymbol{v}_1 \parallel \ldots \parallel \boldsymbol{v}_n]$.

The paper is organized as follows. After describing related work in Section 2, we review the classical Lanczos method in Section 3. In Section 4, we introduce a novel normalization scheme and the resulting normalized s-step Lanczos algorithm. The derivation of the new underlying recurrences is summarized in Section 5 and detailed in Appendix. Section 6 utilizes the s-step Lanczos algorithm to derive new synchronization-reducing variants of BiCG and QMR. In Section 7, the three s-step variants are compared to their classical versions in terms of both numerical stability and parallel performance.

2. Parallel Krylov methods

Non-symmetric eigenvalue problems arising from computational science and engineering are often large and sparse. When only a few dominant eigenvalues are required, Krylov subspace methods enter the picture. The Lanczos algorithm (Lanczos, 1950) is an archetype of this class of iterative methods. At the same time, it is an important building block of Krylov subspace methods for the solution of large sparse systems of linear equations.

When parallelizing the Lanczos or other Krylov subspace methods on message-passing architectures, naïve approaches proceed by parallelizing each underlying linear algebra operation individually. However, the resulting parallel performance of such approaches is known to be limited by communication and synchronization. To overcome this impediment to parallel scalability, significant research effort is spent in designing new Krylov algorithms specifically for parallel computers. The long history of these methods is described in several surveys (Saad, 1989; van der Vorst, 1990; Demmel *et al.*, 1993; Duff and van der Vorst, 1999; Bücker, 2002). A broad classification of these parallel iterative methods is as follows:

i. *Communication-overlapping* algorithms aim to reduce the impact of a communication event by overlapping it with computation and/or other communication (Ghysels *et al.*, 2013; Ghysels and Vanroose, 2014).

ii. *Communication-avoiding* algorithms rely on blocking to reduce the volume of communication (Mohiyuddin *et al.*, 2009; Hoemmen, 2010; Gustafsson *et al.*, 2012a; Carson *et al.*, 2014).

iii. *Synchronization-free* algorithms (Fischer and Freund, 1994) do not involve any global synchronization points (GSPs), defined as the locations of an algorithm at which all information local to a process has to be globally available for all processes in order to continue the computation.

iv. *Synchronization-organizing* algorithms orchestrate synchronization in an attempt to curtail the negative effects caused by global synchronization, for instance, by handling synchronization hierarchically (Curfman McInnes *et al.*, 2014) or using non-blocking all-reduce operations (Kandalla *et al.*, 2012).

v. *Synchronization-reducing* algorithms try to minimize the number of GSPs (Meurant, 1986; Van Rosendale, 1983; Bücker and Sauren, 1996; 1997; 1999; Zuo *et al.*, 2010; Zhu *et al.*, 2014).

While communication-avoiding algorithms successfully reduce the communication volume between processes, they do not directly focus on the synchronization between processes. However, synchronization will increasingly dominate the total execution time of future extreme-scale computer systems, in which the number of processes will be huge. Therefore, we focus on a novel synchronization-reducing Krylov algorithm. Here, a GSP is enforced by dot product-like operations involving a reduction operation on all participating processes. When only a single GSP is enforced for s iterations of the corresponding classical algorithm, this

synchronization-reducing algorithm is referred to as an s-step method (Chronopoulos, 1986; Chronopoulos and Gear, 1989; Chronopoulos and Swanson, 1996). The s-step Lanczos procedure was originally introduced for symmetric matrices (Kim and Chronopoulos, 1991) and later extended to non-symmetric ones (Kim and Chronopoulos, 1992).

3. Classical Lanczos method

The classical Lanczos algorithm (Lanczos, 1950) reduces a non-symmetric $N \times N$ matrix A to a tridiagonal form T_N. At the same time, it also produces two matrices,

$$V_N := [\boldsymbol{v}_1 \parallel \ldots \parallel \boldsymbol{v}_N] \in \mathbb{R}^{N\times N} \tag{1}$$

and

$$W_N := [\boldsymbol{w}_1 \parallel \ldots \parallel \boldsymbol{w}_N] \in \mathbb{R}^{N\times N}, \tag{2}$$

whose columns $\boldsymbol{v}_n$ and $\boldsymbol{w}_n$ are called Lanczos vectors. For the sake of notational simplicity, we assume here that the iteration proceeds up to step N whereas, in practice, it should stop after significantly fewer steps.

Definition 1. (*Classical Lanczos algorithm*) For a given non-symmetric matrix $A \in \mathbb{R}^{N\times N}$, the Lanczos algorithm generates a tridiagonal matrix $T_N \in \mathbb{R}^{N\times N}$ and matrices $V_N \in \mathbb{R}^{N\times N}$ and $W_N \in \mathbb{R}^{N\times N}$ such that

$$W_N^T V_N = \mathrm{I}_{N,N} \quad \text{(biorthonormality)}, \tag{3}$$

$$AV_N = V_N T_N, \tag{4}$$

$$A^T W_N = W_N T_N^T. \tag{5}$$

The classical Lanczos algorithm summarized in pseudocode in Algorithm 1 is based on three-term recurrences. That is, the execution of n iterations of this algorithm generates the tridiagonal matrix,

$$T_n := \begin{bmatrix} \alpha_1 & \beta_2 & & \\ \gamma_2 & \alpha_2 & \ddots & \\ & \ddots & \ddots & \beta_n \\ & & \gamma_n & \alpha_n \end{bmatrix}, \tag{6}$$

which is the $n \times n$ leading principle submatrix of T_N. Thus, (4) indicates that the next Lanczos vector, $\boldsymbol{v}_{n+1}$, involves a matrix-vector product with the matrix A and computations dependent solely on the two previous Lanczos vectors, $\boldsymbol{v}_n$ and $\boldsymbol{v}_{n-1}$. The resulting three-term recurrences are given in Step 4 of this algorithm.

In this algorithm, there are two dot products in Steps 3 and 5, both of which enforce a GSP. In other words, each iteration requires two separate synchronizations of all processes that execute this algorithm. In particular, the result of the first GSP, α_n, needs to be available before the computation of the second GSP begins.

Algorithm 1. Classical Lanczos algorithm.

Input: Non-symmetric matrix $A \in \mathbb{R}^{N\times N}$, as well as starting vectors $\boldsymbol{v}_1, \boldsymbol{w}_1 \in \mathbb{R}^N$, with $\boldsymbol{w}_1^T \boldsymbol{v}_1 = 1$.

Output: After n iterations, the algorithm returns a tridiagonal matrix $T_n = \mathrm{tridiag}(\gamma, \alpha, \beta) \in \mathbb{R}^{n\times n}$ with diagonals $\gamma = (\gamma_2, \ldots, \gamma_n)$, $\alpha = (\alpha_1, \ldots, \alpha_n)$, and $\beta = (\beta_2, \ldots, \beta_n)$, as well as the Lanczos basis $V_n = [\boldsymbol{v}_1 \parallel \ldots \parallel \boldsymbol{v}_n] \in \mathbb{R}^{N\times n}$.

1: Initialize vectors $\boldsymbol{v}_0 \leftarrow \mathbf{0}_N$ and $\boldsymbol{w}_0 \leftarrow \mathbf{0}_N$ and set scalars $\beta_1 \leftarrow 0$ and $\gamma_1 \leftarrow 0$.

2: **for** $n = 1$ **until** Convergence **do**

3: Compute $\alpha_n = \boldsymbol{w}_n^T A \boldsymbol{v}_n$ with **global synchronization**.

4: Compute

$$\tilde{\boldsymbol{v}}_{n+1} = A\boldsymbol{v}_n - \alpha_n \boldsymbol{v}_n - \beta_n \boldsymbol{v}_{n-1},$$

$$\tilde{\boldsymbol{w}}_{n+1} = A^T \boldsymbol{w}_n - \alpha_n \boldsymbol{w}_n - \gamma_n \boldsymbol{w}_{n-1}.$$

5: Choose γ_{n+1} and β_{n+1} such that

$$\gamma_{n+1}\beta_{n+1} = \tilde{\boldsymbol{w}}_{n+1}^T \tilde{\boldsymbol{v}}_{n+1}$$

with **global synchronization**.

6: Scale the Lanczos basis via

$$\boldsymbol{v}_{n+1} = \frac{1}{\gamma_{n+1}} \tilde{\boldsymbol{v}}_{n+1} \quad \text{and} \quad \boldsymbol{w}_{n+1} = \frac{1}{\beta_{n+1}} \tilde{\boldsymbol{w}}_{n+1}.$$

7: **end for**

In practice, different versions of the classical Lanczos algorithm are typically preferred over Algorithm 1. One of the reasons is that Algorithm 1 allows the scaling of only one of the sequences of Lanczos vectors, either $\boldsymbol{v}_n$ or $\boldsymbol{w}_n$. However, to control the numerical stability, one would like to scale both to, say,

$$\|\boldsymbol{v}_n\|_2 = 1 \quad \text{and} \quad \|\boldsymbol{w}_n\|_2 = 1. \tag{7}$$

This is accomplished by replacing the identity in (3) by a diagonal matrix whose nonzero elements are used to scale the second sequence of Lanczos vectors. In addition, there is another version of this algorithm that involves an LU decomposition of the tridiagonal matrix (6) leading to coupled two-term recurrences (Gutknecht, 1997).

4. s-Step Lanczos method

A single block iteration of the non-symmetric s-step Lanczos algorithm introduced by Kim and Chronopoulos (1992) generates s iterations of the classical Lanczos algorithm using only a single GSP. Rather than computing a pair of individual Lanczos vectors $\boldsymbol{v}_k$ and $\boldsymbol{w}_k$, the k-th block iteration of the s-step Lanczos algorithm computes

a pair of blocks of s Lanczos vectors denoted by

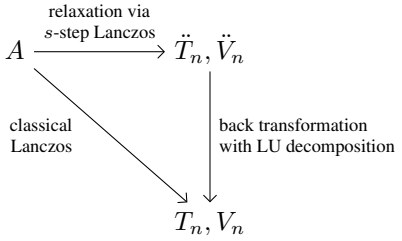
$$\overline{V}_k := \left[\boldsymbol{v}_k^1 \parallel \ldots \parallel \boldsymbol{v}_k^s\right] \in \mathbb{R}^{N\times s} \tag{8}$$

and

$$\overline{W}_k := \left[\boldsymbol{w}_k^1 \parallel \ldots \parallel \boldsymbol{w}_k^s\right] \in \mathbb{R}^{N\times s}. \tag{9}$$

After presenting an overview of the algorithm, we introduce a normalization scheme and present the results of the underlying orthogonalization process.

4.1. High-level overview. The s-step Lanczos method proceeds in two steps. First, *relaxed* Lanczos vectors are computed in a block-wise fashion. In each block iteration, a new block containing s of these vectors as columns is computed. Second, a back transformation is applied to these vectors. The overall structure of the algorithm is schematically depicted as follows:



The following definition summarizes important facts of the s-step Lanczos algorithm using the block formulation of the Lanczos vectors (8) and (9), as well as a corresponding block notation for matrices.

Definition 2. (*s-Step Lanczos algorithm*) Let $n = sk$ with $1 \le n \le N$. For a given non-symmetric matrix $A \in \mathbb{R}^{N\times N}$, the s-step Lanczos algorithm generates an upper Hessenberg matrix $\ddot{T}_n \in \mathbb{R}^{n\times n}$, as well as two additional matrices $\ddot{V}_n \in \mathbb{R}^{N\times n}$ and $\ddot{W}_n \in \mathbb{R}^{N\times n}$, such that

$$\ddot{W}_n^T \ddot{V}_n = \text{block biorthogonal}, \tag{10}$$

$$A\ddot{V}_n = \ddot{V}_n \ddot{T}_n + f_{k+1}\boldsymbol{v}_{k+1}^1 \left[0, \ldots, 0, 1\right], \tag{11}$$

$$A^T\ddot{W}_n = \ddot{W}_n \ddot{T}_n + f_{k+1}\boldsymbol{w}_{k+1}^1 \left[0, \ldots, 0, 1\right]. \tag{12}$$

The upper Hessenberg matrix $\ddot{T}_n$ is block tridiagonal,

$$\ddot{T}_n := \begin{bmatrix} \overline{G}_1 & \overline{E}_2 & & \\ \overline{F}_2 & \overline{G}_2 & \ddots & \\ & \ddots & \ddots & \overline{E}_k \\ & & \overline{F}_k & \overline{G}_k \end{bmatrix} \in \mathbb{R}^{n\times n}, \tag{13}$$

with a nonzero in the upper right corner of the block,

$$\overline{F}_i = \begin{bmatrix} & f_i \\ & \end{bmatrix} \in \mathbb{R}^{s\times s}, \tag{14}$$

while each $\overline{E}_i$ is a dense $s\times s$ matrix, and each $\overline{G}_i \in \mathbb{R}^{s\times s}$ is in upper Hessenberg form. The block-wise grouping of the relaxed Lanczos vectors is given by

$$\ddot{V}_n := \left[\overline{V}_1 \parallel \ldots \parallel \overline{V}_k\right] \in \mathbb{R}^{N\times n}, \tag{15}$$

$$\ddot{W}_n := \left[\overline{W}_1 \parallel \ldots \parallel \overline{W}_k\right] \in \mathbb{R}^{N\times n}. \tag{16}$$

When implemented, the s-step Lanczos algorithm iterates $k = n/s$ block iterations yielding both $\ddot{T}_n$ and $\ddot{V}_n$. It then turns $\ddot{T}_n$ and $\ddot{V}_n$ into the matrices T_n and V_n from the classical Lanczos procedure. This back transformation is sketched in the following theorem.

Theorem 1. *Let $\ddot{W}_n^T\ddot{V}_n$ be a non-singular matrix and let*

$$\ddot{W}_n^T\ddot{V}_n = \ddot{L}_n\ddot{U}_n \tag{17}$$

denote its LU decomposition. Then, $\ddot{T}_n$, $\ddot{V}_n$ and $\ddot{W}_n$ can be transformed into T_n, W_n and V_n, originating from the classical Lanczos method in the absence of breakdowns:

$$T_n = \ddot{U}_n\ddot{T}_n\ddot{U}_n^{-1}, \tag{18}$$

$$V_n = \ddot{V}_n\ddot{U}_n^{-1}, \tag{19}$$

$$W_n^T = \ddot{L}_n^{-1}\ddot{W}_n^T. \tag{20}$$

Proof. See the work of Kim and Chronopoulos (1992). ■

4.2. Normalization scheme. The original s-step Lanczos algorithm (Kim and Chronopoulos, 1992) implemented in double precision floating-point arithmetic can involve a numerical overflow. More precisely, we observed that the floating-point values in $\ddot{W}_n^T\ddot{V}_n$ can grow rapidly. To reduce the possibility of numerical overflow, we introduce the normalization scheme

$$\tilde{\boldsymbol{v}}_{k+1}^1 := f_{k+1}\boldsymbol{v}_{k+1}^1, \tag{21}$$

$$\tilde{\boldsymbol{w}}_{k+1}^1 := f_{k+1}\boldsymbol{w}_{k+1}^1, \tag{22}$$

where

$$f_{k+1} := \sqrt{\left|\langle\tilde{\boldsymbol{w}}_{k+1}^1, \tilde{\boldsymbol{v}}_{k+1}^1\rangle\right|}. \tag{23}$$

Thus, we have

$$\langle\boldsymbol{w}_{k+1}^1, \boldsymbol{v}_{k+1}^1\rangle = \pm 1. \tag{24}$$

This differs from the version of Kim and Chronopoulos (1992), in which any normalization is avoided, corresponding to $f_{k+1} := 1$ in the new scheme. In addition to this normalization scheme, it turns out that it is also convenient to normalize $\tilde{\boldsymbol{v}}_{k+1}^j$ and $\tilde{\boldsymbol{w}}_{k+1}^j$, such that

$$\langle\boldsymbol{w}_{k+1}^j, \boldsymbol{v}_{k+1}^j\rangle = \pm 1 \quad \text{for } j = 2, \ldots, s \tag{25}$$

holds. Therefore, we scale the Lanczos vectors by

$$\boldsymbol{v}_{k+1}^j = \sigma_{k+1}^j\tilde{\boldsymbol{v}}_{k+1}^j \quad \text{for } j = 2, \ldots, s \tag{26}$$

$$\boldsymbol{w}_{k+1}^j = \sigma_{k+1}^j \tilde{\boldsymbol{w}}_{k+1}^j \quad \text{for } j = 2, \ldots, s, \tag{27}$$

where the scaling factors are given by

$$\sigma_{k+1}^j := |\langle \tilde{w}_{k+1}^j, \tilde{\boldsymbol{v}}_{k+1}^j \rangle|^{-\frac{1}{2}} \text{ for } j = 2, \ldots, s. \tag{28}$$

This normalization scheme requires a new derivation of the underlying recurrences in Section 5 since these differ from the recurrences of Kim and Chronopoulos (1992). The resulting s-step Lanczos algorithm is depicted in Algorithm 2. Here, each block iteration k generates two blocks of s Lanczos vectors. The first Lanczos vector in each block is computed by a recurrence involving Lanczos vectors from the two previous blocks shown in (30) and (31). The remaining $s-1$ Lanczos vectors in each block are computed in (32) and (33). The computation of the Lanczos vectors involves the coefficient vectors $\boldsymbol{g}_k^s$, $\boldsymbol{e}_k^s$ and $\boldsymbol{t}_k^2, \boldsymbol{t}_k^3, \ldots, \boldsymbol{t}_k^s$ that need to be chosen to satisfy the block biorthogonality condition (10). The LU decomposition in Step 16, as well as the back transformation in Step 17, workS on the data from the current block iteration. For instance, since by (10) the matrix $\ddot{W}_{sk}^T \ddot{V}_{sk}$ is block diagonal, only an LU decomposition of the $s \times s$ matrix

$$\overline{M}_k := \overline{W}_k^T \overline{V}_k \tag{29}$$

has to be carried out in Step 16.

Finally, consider a block iteration of Algorithm 2 that computes a pair of s Lanczos vectors. We emphasize that each block iteration requires only a single GSP in Step 7.

4.3. Orthogonalizing the s-step Lanczos basis. The coefficient vectors $\boldsymbol{e}_k^s$, $\boldsymbol{g}_k^i$, and $\boldsymbol{t}_k^j$ are chosen to satisfy the biorthogonality (10). This is achieved by solving systems of linear equations with the coefficient matrix (29) that is assumed to be non-singular. The following theorem states the properties of the coefficient vectors.

Theorem 2. *The vectors $\boldsymbol{e}_k^i$, $\boldsymbol{g}_k^i$ and $\boldsymbol{t}_k^j$ with $i = 1, \ldots, s$ and $j = 2, \ldots, s$ are given by the solutions of the following $s \times s$ systems of linear equations:*

$$\overline{M}_{k-1} \boldsymbol{e}_k^i = \boldsymbol{c}_k^i,$$

where

$$\boldsymbol{c}_k^i := \left[\langle \boldsymbol{w}_{k-1}^1, A\boldsymbol{v}_k^i \rangle, \ldots, \langle \boldsymbol{w}_{k-1}^s, A\boldsymbol{v}_k^i \rangle\right]^T, \tag{34}$$

$$\overline{M}_k \boldsymbol{g}_k^i = \boldsymbol{d}_k^i,$$

where

$$\boldsymbol{d}_k^i := \left[\langle \boldsymbol{w}_k^1, A\boldsymbol{v}_k^i \rangle, \ldots, \langle \boldsymbol{w}_k^s, A\boldsymbol{v}_k^i \rangle\right]^T, \tag{35}$$

$$\overline{M}_k \boldsymbol{t}_k^j = \boldsymbol{b}_k^j,$$

where

$$\boldsymbol{b}_k^j := \left[\langle \boldsymbol{w}_k^1, A^{j-1}\boldsymbol{v}_{k+1}^1 \rangle, \ldots, \langle \boldsymbol{w}_k^s, A^{j-1}\boldsymbol{v}_{k+1}^1 \rangle\right]^T. \tag{36}$$

Algorithm 2. Synchronization-reducing s-step Lanczos.

Input: Non-symmetric matrix $A \in \mathbb{R}^{N \times N}$, starting vectors $\boldsymbol{v}_1^1, \boldsymbol{w}_1^1 \in \mathbb{R}^N$ with $\boldsymbol{v}_1^1 = \boldsymbol{w}_1^1$ and parameter s.

Output: After $k = n/s$ block iterations, return a tridiagonal matrix $T_n \in \mathbb{R}^{n \times n}$ and Lanczos basis $V_n \in \mathbb{R}^{N \times n}$.

1: Initialize $\overline{V}_0 \leftarrow 0_{N,s}$ and $\overline{W}_0 \leftarrow 0_{N,s}$ and compute

$$\overline{V}_1 \leftarrow \left[\boldsymbol{v}_1^1 \,\|\, A\boldsymbol{v}_1^1 \,\|\, \ldots \,\|\, A^{s-1}\boldsymbol{v}_1^1\right],$$
$$\overline{W}_1 \leftarrow \left[\boldsymbol{w}_1^1 \,\|\, A^T\boldsymbol{w}_1^1 \,\|\, \ldots \,\|\, (A^T)^{s-1}\boldsymbol{w}_1^1\right].$$

2: Compute dot products $\langle \boldsymbol{w}_1^1, A^j \boldsymbol{v}_1^1 \rangle$ for $j = 0, \ldots, 2s-1$.

3: **for** $k = 1$ **until** Convergence **do**

4: Compute $\overline{M}_k^{i,j}$, $\boldsymbol{c}_k^s$ and $\boldsymbol{d}_k^{i,j}$ for $i = 1, \ldots, s$ and $j = 1, \ldots, s$ using efficient recurrences from Theorem 3.

5: Solve $\overline{M}_{k-1}\boldsymbol{e}_k^s = \boldsymbol{c}_k^s$ and $\overline{M}_k \boldsymbol{g}_k^s = \boldsymbol{d}_k^s$.

6: Orthogonalize $\tilde{\boldsymbol{v}}_{k+1}^1$ against $\overline{W}_k$ and $\tilde{\boldsymbol{w}}_{k+1}^1$ against $\overline{V}_k$ by

$$\tilde{\boldsymbol{v}}_{k+1}^1 \leftarrow A\boldsymbol{v}_k^s - \overline{V}_{k-1}\boldsymbol{e}_k^s - \overline{V}_k \boldsymbol{g}_k^s, \tag{30}$$
$$\tilde{\boldsymbol{w}}_{k+1}^1 \leftarrow A^T\boldsymbol{w}_k^s - \overline{W}_{k-1}\boldsymbol{e}_k^s - \overline{W}_k \boldsymbol{g}_k^s. \tag{31}$$

7: Compute $2s$ dot products $\langle \tilde{\boldsymbol{w}}_{k+1}^1, A^j \tilde{\boldsymbol{v}}_{k+1}^1 \rangle$ for $j = 0, \ldots, 2s-1$ with **global synchronization**.

8: Compute normalization coefficient

$$f_{k+1} \leftarrow \sqrt{|\langle \tilde{\boldsymbol{w}}_{k+1}^1, \tilde{\boldsymbol{v}}_{k+1}^1 \rangle|}.$$

9: Carry out normalization

$$\boldsymbol{v}_{k+1}^1 \leftarrow \tilde{\boldsymbol{v}}_{k+1}^1 / f_{k+1} \quad \text{and} \quad \boldsymbol{w}_{k+1}^1 \leftarrow \tilde{\boldsymbol{w}}_{k+1}^1 / f_{k+1}.$$

10: Normalize, for $j = 0, \ldots, 2s-1$, via

$$\langle \boldsymbol{w}_{k+1}^1, A^j \boldsymbol{v}_{k+1}^1 \rangle \leftarrow \langle \tilde{\boldsymbol{w}}_{k+1}^1, A^j \tilde{\boldsymbol{v}}_{k+1}^1 \rangle \Big/ f_{k+1}^2.$$

11: Compute $\boldsymbol{b}_k^j$ for $j = 2, \ldots, s+1$ using efficient recurrences from Theorem 3.

12: Solve $\overline{M}_k \boldsymbol{t}_k^j = \boldsymbol{b}_k^j$ for $j = 2, \ldots, s$.

13: Orthogonalize $\left[A\boldsymbol{v}_{k+1}^1 \,\|\, \ldots \,\|\, A^{s-1}\boldsymbol{v}_{k+1}^1\right]$ against $\overline{W}_k$ and the block $\left[A^T\boldsymbol{w}_{k+1}^1 \,\|\, \ldots \,\|\, (A^T)^{s-1}\boldsymbol{w}_{k+1}^1\right]$ against $\overline{V}_k$ by

$$\tilde{\boldsymbol{v}}_{k+1}^j \leftarrow A^{j-1}\boldsymbol{v}_{k+1}^1 - \overline{V}_k \boldsymbol{t}_k^j, \qquad j = 2, \ldots, s \tag{32}$$
$$\tilde{\boldsymbol{w}}_{k+1}^j \leftarrow (A^T)^{j-1}\boldsymbol{w}_{k+1}^1 - \overline{W}_k \boldsymbol{t}_k^j, j = 2, \ldots, s. \tag{33}$$

14: Compute normalization coefficients σ_{k+1}^j for $j = 2, \ldots, s$ using efficient recurrences from Theorem 3.

15: Normalize, for $j = 2, \ldots, s$, via

$$\boldsymbol{v}_{k+1}^j \leftarrow \tilde{\boldsymbol{v}}_{k+1}^j \sigma_{k+1}^j \quad \text{and} \quad \boldsymbol{w}_{k+1}^j \leftarrow \tilde{\boldsymbol{w}}_{k+1}^j \sigma_{k+1}^j.$$

16: Update LU decomposition of $\ddot{L}_{sk}\ddot{U}_{sk} \leftarrow \ddot{W}_{sk}^T \ddot{V}_{sk}$.

17: Update back transformation

$$T_{sk} \leftarrow \ddot{U}_{sk}\ddot{T}_{sk}\ddot{U}_{sk}^{-1}, \quad V_{sk} \leftarrow \ddot{V}_{sk}\ddot{U}_{sk}^{-1},$$
$$W_{sk}^T \leftarrow \ddot{L}_{sk}^{-1}\ddot{W}_{sk}^T.$$

18: **end for**

Proof. Given by Feuerriegel and Bücker (2013a), as well as Kim and Chronopoulos (1992). ∎

The solution of these small and dense linear systems is computationally affordable. The next section summarizes the implementation details for setting up the coefficient matrices and the right-hand sides.

5. Deriving efficient recurrence equations

Computing the right-hand side vectors $\boldsymbol{b}_k^j$, $\boldsymbol{c}_k^j$, $\boldsymbol{d}_k^j$ and the coefficient matrix $\overline{M}_k$ explicitly for all relevant j's is not only a computationally expensive task, but also involves a considerable number of dot products. In fact, the evaluation of these dot products would destroy the benefits of the s-step approach instantly. Fortunately, there is a remedy to this problem which consists in retrieving these products recursively from the $2\,s$ dot products $\langle \boldsymbol{w}_k^1, \boldsymbol{v}_k^1\rangle, \langle \boldsymbol{w}_k^1, A\boldsymbol{v}_k^1\rangle, \ldots, \langle \boldsymbol{w}_k^1, A^{2s-1}\boldsymbol{v}_k^1\rangle$, as stated in the following theorem. These dot products introduce the *only* GSP per block iteration.

Throughout this article, the symbols $\boldsymbol{b}_k^{i,j}$, $\boldsymbol{c}_k^{i,j}$, $\boldsymbol{d}_k^{i,j}$, $\boldsymbol{g}_k^{i,j}$ and $\boldsymbol{t}_k^{i,j}$ denote the i-th element of the vectors $\boldsymbol{b}_k^j$, $\boldsymbol{c}_k^j$, $\boldsymbol{d}_k^j$, $\boldsymbol{g}_k^j$ and $\boldsymbol{t}_k^j$, respectively. The symbol $\overline{M}_k^{i,j}$ denotes the matrix element of $\overline{M}_k$ at row i and column j.

Theorem 3. (Efficient recurrences) *The scaling factors* (28)*, as well as the coefficient matrices and the right-hand sides involved in Theorem 2, are given as follows, where we define* $\sigma_k^1 := 1$ *and use the indices* $i = 1, \ldots, s$*:*

$$\sigma_k^j = |\langle \boldsymbol{w}_k^1, A^{2j-2}\boldsymbol{v}_k^1\rangle - (\boldsymbol{t}_{k-1}^j)^T \overline{M}_{k-1}\boldsymbol{t}_{k-1}^j|^{-\frac{1}{2}} \quad \text{for } j = 2, \ldots, s,$$

$$\overline{M}_k^{i,j} = \sigma_k^i\sigma_k^j\left[\langle \boldsymbol{w}_k^1, A^{i+j-2}\boldsymbol{v}_k^1\rangle - (\boldsymbol{t}_{k-1}^i)^T \overline{M}_{k-1}\boldsymbol{t}_{k-1}^j\right] \quad \text{for } j = 1, \ldots, s,$$

$$\boldsymbol{b}_k^{i,j} = \frac{\sigma_k^i}{\sigma_k^s}\left[\langle f_{k+1}\boldsymbol{w}_{k+1}^1, A^{i+j-s-2}\boldsymbol{v}_{k+1}^1\rangle + \sum_{\substack{\iota=2s\\ +3-i-j}}^{s} \frac{\sigma_k^\iota \boldsymbol{g}_k^{\iota,s}\boldsymbol{b}_k^{i-s+\iota-1,j}}{\sigma_k^{i-s+\iota-1}}\right] \quad \text{for } j = 2, \ldots, s+1,$$

$$\boldsymbol{c}_k^{j,s} = 0 \quad \text{for } j = 1, \ldots, s-1,$$

$$\boldsymbol{c}_k^{s,s} = \sigma_k^s\left[\boldsymbol{b}_{k-1}^{s,s+1} - \left[\boldsymbol{d}_{k-1}^{s,1}, \ldots, \boldsymbol{d}_{k-1}^{s,s}\right]\boldsymbol{t}_{k-1}^s\right],$$

$$\boldsymbol{d}_k^{i,j} = \sigma_k^i\sigma_k^j\Big[\langle \boldsymbol{w}_k^1, A^{i+j-1}\boldsymbol{v}_k^1\rangle - (\boldsymbol{t}_{k-1}^{i+1})^T\overline{M}_{k-1}\boldsymbol{t}_{k-1}^j - (\boldsymbol{t}_{k-1}^i)^T\overline{M}_{k-1}\boldsymbol{t}_{k-1}^{j+1} + (\boldsymbol{t}_{k-1}^i)^T\left[\boldsymbol{d}_{k-1}^1 \parallel \ldots \parallel \boldsymbol{d}_{k-1}^s\right]\boldsymbol{t}_{k-1}^j\Big] \quad \text{for } j = 1, \ldots, s.$$

Proof. See Appendix. ∎

Table 1 compares the main computational cost for $n = sk$ iterations of the classical Lanczos algorithm, Algorithm 1, and k block iterations of two s-step variants. The first s-step variant is the one without normalization introduced by Kim and Chronopoulos (1992), while the second is the one proposed in Algorithm 2. Recall that the latter requires only a single GSP per block iteration. This table reports the number of operations, as well as vector storages of size N, neglecting all corresponding costs of vectors of dimension s. Though the s-step variants slightly raise the computational cost, they reduce the number of GSPs by a factor of $\mathcal{O}(s)$.

6. s-Step BiCG and QMR methods

The Lanczos algorithm is now used to design Krylov subspace methods for the solution of linear systems. The aim of this section is to derive new synchronization-reducing s-step variants of BiCG and QMR.

6.1. Solving linear systems using the Lanczos basis.

The Lanczos basis $V_n \in \mathbb{R}^{N\times n}$ generated by any Lanczos algorithm can be used to iteratively solve non-symmetric systems of linear equations,

$$A\boldsymbol{x} = \boldsymbol{b},$$

where $\boldsymbol{x}, \boldsymbol{b} \in \mathbb{R}^N$. Given an initial guess $\boldsymbol{x}_0$ to the exact solution $\boldsymbol{x}$, the current approximation is given by

$$\boldsymbol{x}_n = \boldsymbol{x}_0 + V_n\boldsymbol{z}_n, \tag{37}$$

where the coefficient vector $\boldsymbol{z}_n \in \mathbb{R}^n$ needs to be determined. For reasons of numerical stability, we will also use another basis,

$$P_n = [\boldsymbol{p}_1 \parallel \ldots \parallel \boldsymbol{p}_n] \in \mathbb{R}^{N\times n},$$

defined by

$$V_n = P_nU_n.$$

Here, U_n is the $n \times n$ unit upper triangular factor from the LU decomposition of the $(n+1) \times n$ tridiagonal matrix,

$$\underline{T}_n := \left[\begin{array}{ccccc} & & T_n & & \\ \hline 0 & \ldots & 0 & \gamma_{n+1} \end{array}\right],$$

which is obtained from appending another row at the bottom of the tridiagonal matrix T_n defined in (6). This LU decomposition,

$$L_nU_n = \left[\begin{array}{ccccc} \tau_1 & & & & \\ \omega_2 & \tau_2 & & & \\ & \ddots & \ddots & & \\ & & \omega_n & \tau_n & \\ \hline & & & & \omega_{n+1} \end{array}\right] \left[\begin{array}{ccccc} 1 & \mu_2 & & & \\ & 1 & \ddots & & \\ & & \ddots & \mu_n & \\ & & & 1 & \end{array}\right], \tag{38}$$

Table 1. Comparison of cost for $n = sk$ iterations of the classical Lanczos algorithms and k block iterations of the synchronization-reducing s-step Lanczos variants.

Operation/Storage	Classical (Algorithm 1)	Unnormalized s-step (Kim and Chronopoulos, 1992)	Normalized s-step (Algorithm 2)
Dot products (of length N)	$2\,sk$	$2\,sk$	$2\,sk$
Vector updates (of length N)	$6\,sk$	$2\,s(s-1)k + 4\,sk$	$2\,s(s-1)k + 8\,sk - k$
Matrix-vector products	$2\,sk$	$2\,sk + k$	$2\,sk + k$
Synchronization points	$\mathcal{O}(sk)$	k	k
Back transformations	—	$2k$	$2k$
Vector storage (of length N)	4	$4\,s$	$4\,s$

is computed for $n \geq 2$ via

$$\omega_n = \gamma_n, \tag{39}$$

$$\mu_n = \beta_n/\tau_{n-1}, \tag{40}$$

$$\tau_n = \alpha_n - \omega_n\mu_n \qquad \text{with } \tau_1 := \alpha_1. \tag{41}$$

Recall from Section 3 that this LU decomposition is the basic building block of the Lanczos variant based on coupled two-term recurrences. We will use this variant in the remainder of the paper because it has a better reputation in terms of numerical stability.

From the structure of U_n according to (38), we find

$$\boldsymbol{p}_n = \boldsymbol{v}_n - \mu_n\boldsymbol{p}_{n-1}. \tag{42}$$

Furthermore, the relation of this formulation based on coupled-two term recurrences to the corresponding one based on three-term recurrences is given by

$$AP_n = V_{n+1}L_n. \tag{43}$$

Now, we replace the vector $\boldsymbol{z}_n$ in (37) by the n-dimensional coefficient vector

$$\boldsymbol{y}_n := U_n\boldsymbol{z}_n$$

and arrive at

$$\boldsymbol{x}_n = \boldsymbol{x}_0 + P_n\boldsymbol{y}_n. \tag{44}$$

The residual vector defined by

$$\boldsymbol{r}_n := \boldsymbol{b} - A\boldsymbol{x}_n$$

is then, using (43), transformed into

$$\begin{aligned}\boldsymbol{r}_n &= \boldsymbol{r}_0 - AP_n\boldsymbol{y}_n = \omega_1\boldsymbol{v}_1 - V_{n+1}L_n\boldsymbol{y}_n \\ &= V_{n+1}\left([\omega_1, 0, \ldots, 0]^T - L_n\boldsymbol{y}_n\right),\end{aligned} \tag{45}$$

where $\boldsymbol{v}_1 := \boldsymbol{r}_0/\omega_1$ with a scaling factor $\omega_1 := \|\boldsymbol{r}_0\|_2$.

6.2. Synchronization-reducing s-step biconjugate gradient algorithm. The idea of BiCG (Fletcher, 1976) is to determine the coefficient vector $\boldsymbol{y}_n$ by zeroing out the first n entries of the vector in parentheses in (45). More precisely, if the symbol $\bullet$ denotes an arbitrary value in the $(n+1)$-th component of that vector, we require that this vector satisfy

$$[\omega_1, 0, \ldots, 0]^T - L_n\boldsymbol{y}_n = [0, \ldots, 0, \bullet]^T. \tag{46}$$

Let $\boldsymbol{y}_n = [\kappa_1, \ldots, \kappa_n]^T$ denote the entries of the coefficient vector. Inserting this ansatz into (45) gives

$$\begin{aligned}\boldsymbol{r}_n &= V_{n+1}[0, \ldots, 0, -\omega_{n+1}\kappa_n]^T \\ &= -\omega_{n+1}\kappa_n\boldsymbol{v}_{n+1}.\end{aligned}$$

Assuming that (46) is fulfilled in the first $n-1$ components, we find for the $(n-1)$-th component of (46) that

$$0 = -\omega_n\kappa_{n-1} - \tau_n\kappa_n.$$

Hence

$$\kappa_n = -\frac{\omega_n\kappa_{n-1}}{\tau_n} \quad \text{with} \quad \kappa_0 := -1. \tag{47}$$

The process of fixing $\boldsymbol{y}_n$ is easily updated in each iteration step because $\boldsymbol{y}_{n-1}$ coincides with the first $n-1$ components of $\boldsymbol{y}_n$; more precisely,

$$\boldsymbol{y}_n = \begin{bmatrix} \boldsymbol{y}_{n-1} \\ \kappa_n \end{bmatrix}.$$

Thus, from (44), the n-th approximation is obtained via

$$\boldsymbol{x}_n = \boldsymbol{x}_0 + P_{n-1}\boldsymbol{y}_{n-1} + \kappa_n\boldsymbol{p}_n = \boldsymbol{x}_{n-1} + \kappa_n\boldsymbol{p}_n. \tag{48}$$

We now let Algorithm 2 generate the Lanczos basis V_n and put together (39)–(41), (42), (47), and (48). The pseudocode of the resulting synchronization-reducing s-step BiCG is presented in Algorithm 3. The k-th block iteration of the s-step BiCG algorithm computes s Lanczos vectors and updates the LU decomposition of $\underline{T}_n$ from (38) with $n = sk$. Throughout the s-step BiCG algorithm, the call to the s-step Lanczos method in Step 3 introduces the only GSP per block iteration.

6.3. Synchronization-reducing s-step quasi-minimal residual algorithm. The idea of QMR (Freund and Nachtigal, 1991; 1994) is to find the coefficient vector

Algorithm 3. Synchronization-reducing s-step BiCG.

Input: Non-symmetric matrix $A \in \mathbb{R}^{N\times N}$, right-hand side $\boldsymbol{b} \in \mathbb{R}^N$, initial guess $\boldsymbol{x}_0 \in \mathbb{R}^N$ and parameter s.

Output: After $k = n/s$ block iterations, the algorithm returns an approximation $\boldsymbol{x}_n$ to the solution $A^{-1}\boldsymbol{b}$.

1: Initialize vectors $\boldsymbol{r}_0 \leftarrow \boldsymbol{b} - A\boldsymbol{x}_0$, $\boldsymbol{v}_1 \leftarrow \boldsymbol{r}_0/\left\|\boldsymbol{r}_0\right\|_2$, $\boldsymbol{p}_1 \leftarrow \boldsymbol{v}_1$ and set scalars $\kappa_0 \leftarrow -1$, $\omega_1 \leftarrow \left\|\boldsymbol{r}_0\right\|_2$, $\mu_1 \leftarrow 0$, and $\tau_1 \leftarrow \alpha_1$ where α_1 is taken from the Lanczos process initialized with $\boldsymbol{w}_1^1 \leftarrow \boldsymbol{v}_1^1$.
2: **for** $k = 1$ **until** Convergence **do**
3: Compute next s Lanczos vectors $\boldsymbol{v}_{s(k-1)+1}$, …, $\boldsymbol{v}_{sk}$ as well as next entries in $\underline{T}_{sk}$, i.e., $\alpha_{s(k-1)+1}, \ldots, \alpha_{sk}$, $\beta_{s(k-1)+1}, \ldots, \beta_{sk}$, $\gamma_{s(k-1)+1}, \ldots, \gamma_{sk}$.
4: **for** $n = s(k-1)+1$ **to** sk **do**
5: Update LU decomposition of $\underline{T}_{sk}$ via $\omega_n \leftarrow \gamma_n$, $\mu_n \leftarrow \beta_n/\tau_{n-1}$, $\tau_n \leftarrow \alpha_n - \omega_n\mu_n$.
6: Compute vector $\boldsymbol{p}_n \leftarrow \boldsymbol{v}_n - \mu_n\boldsymbol{p}_{n-1}$.
7: Set $\kappa_n \leftarrow -\omega_n\kappa_{n-1}/\tau_n$ and compute approximation $\boldsymbol{x}_n \leftarrow \boldsymbol{x}_{n-1} + \kappa_n\boldsymbol{p}_n$.
8: **end for**
9: **end for**

$\boldsymbol{y}_n$ by minimizing the Euclidean norm of the vector in parentheses in (45). To solve these least-squares problems, we follow Sauren and Bücker (1998) as stated in the subsequent Theorem 4.

Theorem 4. *The unique solution of*

$$\boldsymbol{y}_n := \arg\min_{\boldsymbol{y}\in\mathbb{R}^n} \left\| [\omega_1, 0, \ldots, 0]^T - L_n\boldsymbol{y} \right\|_2 \tag{49}$$

is given by the recurrences

$$\boldsymbol{y}_n = \begin{bmatrix} \boldsymbol{y}_{n-1} \\ 0 \end{bmatrix} + \boldsymbol{g}_n, \tag{50}$$

$$\boldsymbol{g}_n = \theta_n \begin{bmatrix} \boldsymbol{g}_{n-1} \\ 0 \end{bmatrix} + [0, \ldots, 0, \xi_n]^T, \qquad n \geq 2, \tag{51}$$

with $\boldsymbol{y}_1 = \boldsymbol{g}_1 = [\xi_1]$, *where*

$$\theta_n = \frac{|\tau_n|^2 (1-\sigma_n)}{\sigma_n |\tau_n|^2 + |\omega_{n+1}|^2}, \tag{52}$$

$$\xi_n = -\frac{\omega_n \overline{\tau}_n \xi_{n-1}}{\sigma_n |\tau_n|^2 + |\omega_{n+1}|^2}, \quad \xi_0 = -1, \tag{53}$$

$$\sigma_{n+1} = \frac{\sigma_n |\tau_n|^2}{\sigma_n |\tau_n|^2 + |\omega_{n+1}|^2}, \qquad \sigma_1 = 1. \tag{54}$$

The current approximation is given by

$$\boldsymbol{x}_n = \boldsymbol{x}_{n-1} + \boldsymbol{d}_n, \tag{55}$$

where $\boldsymbol{d}_n = \theta_n\boldsymbol{d}_{n-1} + \xi_n\boldsymbol{p}_n$.

Again, we let Algorithm 2 generate the Lanczos basis V_n and put together the equations from Theorem 4. The resulting s-step QMR algorithm is given by the pseudo-code in Algorithm 4. Once more, the invocation

Algorithm 4. Synchronization-reducing s-step QMR.

Input: Non-symmetric matrix $A \in \mathbb{R}^{N\times N}$, right-hand side $\boldsymbol{b} \in \mathbb{R}^N$, initial guess $\boldsymbol{x}_0 \in \mathbb{R}^N$ and parameter s.

Output: After $k = n/s$ block iterations, the algorithm returns an approximation $\boldsymbol{x}_n$ to the solution $A^{-1}\boldsymbol{b}$.

1: Initialize vectors $\boldsymbol{r}_0 \leftarrow \boldsymbol{b} - A\boldsymbol{x}_0$, $\boldsymbol{v}_1 \leftarrow \boldsymbol{r}_0/\left\|\boldsymbol{r}_0\right\|_2$, $\boldsymbol{p}_1 \leftarrow \boldsymbol{v}_1$, $\boldsymbol{d}_0 \leftarrow \boldsymbol{0}_N$ and set scalars $\xi_0 \leftarrow -1$, $\sigma_1 \leftarrow 1$, $\omega_1 \leftarrow \left\|\boldsymbol{r}_0\right\|_2$, $\mu_1 \leftarrow 0$, and $\tau_1 \leftarrow \alpha_1$, where α_1 is taken from the Lanczos process initialized with $\boldsymbol{w}_1^1 \leftarrow \boldsymbol{v}_1^1$.
2: **for** $k = 1$ **until** Convergence **do**
3: Compute next s Lanczos vectors $\boldsymbol{v}_{s(k-1)+1}$, …, $\boldsymbol{v}_{sk}$, as well as next entries in $\underline{T}_{sk}$, i.e., $\alpha_{s(k-1)+1}, \ldots, \alpha_{sk}$, $\beta_{s(k-1)+1}, \ldots, \beta_{sk}$, $\gamma_{s(k-1)+1}, \ldots, \gamma_{sk}$.
4: **for** $n = s(k-1)+1$ **to** sk **do**
5: Update LU decomposition of $\underline{T}_{sk}$ via $\omega_n \leftarrow \gamma_n$, $\mu_n \leftarrow \beta_n/\tau_{n-1}$, $\tau_n \leftarrow \alpha_n - \omega_n\mu_n$.
6: Compute vector $\boldsymbol{p}_n \leftarrow \boldsymbol{v}_n - \mu_n\boldsymbol{p}_{n-1}$.
7: $\theta_n \leftarrow \frac{|\tau_n|^2(1-\sigma_n)}{\sigma_n|\tau_n|^2+|\omega_{n+1}|^2}$, $\xi_n \leftarrow -\frac{\omega_n\overline{\tau}_n\xi_{n-1}}{\sigma_n|\tau_n|^2+|\omega_{n+1}|^2}$, $\sigma_{n+1} \leftarrow \frac{\sigma_n|\tau_n|^2}{\sigma_n|\tau_n|^2+|\omega_{n+1}|^2}$.
8: Compute $\boldsymbol{d}_n \leftarrow \theta_n\boldsymbol{d}_{n-1} + \xi_n\boldsymbol{p}_n$ and approximation $\boldsymbol{x}_n \leftarrow \boldsymbol{x}_{n-1} + \boldsymbol{d}_n$.
9: **end for**
10: **end for**

of the s-step Lanczos subroutine in Step 3 represents the only GSP per block iteration.

7. Numerical experiments

This section compares classical and synchronization-reducing s-step variants of Lanczos, BiCG and QMR in terms of numerical accuracy and parallel performance, using an example by Freund and Nachtigal (1991). We consider the differential equation

$$-\Delta u + 40\,(xu_x + yu_y + zu_z) - 250\,u = f \tag{56}$$

on $\Omega = (0,1)\times(0,1)\times(0,1)$ with $u = 0$ on the boundary. Using first-order centered differences and $\sqrt[3]{N}$ discretization points in each direction, we arrive at a linear system of order N. The right-hand side $f(x,y,z)$ is determined such that the vector of all ones is the exact solution. The initial guess is set to $\boldsymbol{x}_0 := \boldsymbol{0}_N$.

To this end, we carried out a parallel implementation of the Lanczos algorithm using PETSc (Balay *et al.*, 1997) and included it as an additional eigenvalue solver

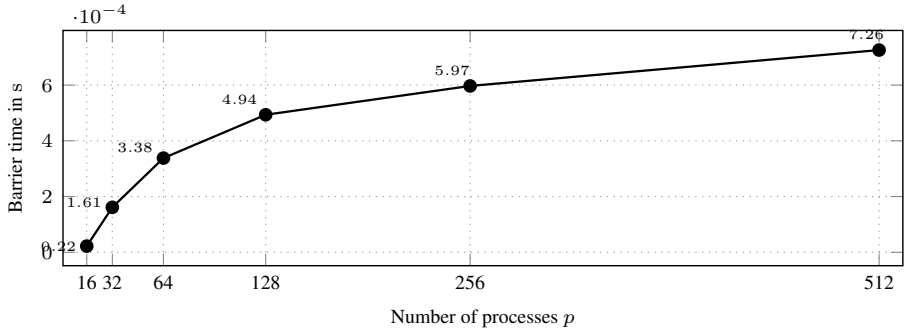


Fig. 1. Barrier time versus number of processes on the *Nehalem*-based cluster.

inside SLEPc (Hernandez *et al.*, 2005). Both s-step BiCG and QMR are implemented in parallel using PETSc 3.0 and compared to their classical variants implemented in PETSc. We use a block Jacobi preconditioner and choose $N = 2^{24}$ since PETSc recommends at least $10\,000$ to $20\,000$ unknowns per process. All computations are performed on a *Nehalem*-based cluster[1] whose barrier times measured via PETSc are given in Figure 1. A synchronization with a reduction operation accounts for an average of 5.97×10^{-4} s with 256 processes and 7.26×10^{-4} s with 512 processes.

7.1. Synchronization-reducing s-step Lanczos algorithm. Figure 2 compares the Lanczos implementations. All algorithms are started with

$$\boldsymbol{v}_1^1 = \boldsymbol{w}_1^1 = [1, \ldots, 1]^T . \tag{57}$$

The diagram shows the convergence history of the error of the biorthogonality property of A. The deviation of the unnormalized 2-step variant exceeds 10^0 from iteration 6 onwards and the algorithm breaks down at iteration 45 with a numerical overflow. The normalized 2-step variant competes well until iteration 35 with a deviation below 10^0. However, when increasing s to $s = 5$, the divergence of the normalized variant already starts around iteration 33 with a deviation above 10^0. The tendency for growing numerical instabilities when s is increased is well known (Carson and Demmel, 2014; Kim, 2010).

Previous research (Gustafsson *et al.*, 2012b; Kim and Chronopoulos, 1992; Kim and Kim, 2005) has already demonstrated that, compared with the classical algorithm, execution time is reduced by the unnormalized s-step algorithm. As a result, we excluded the unnormalized s-step method from the analysis and only remark that the parallel performance characteristics of the normalized and unnormalized variants are almost identical. The focus is instead upon the parallel performance of the normalized s-step Lanczos algorithm. We compared the new normalized s-step variant to the classical one implemented in SLEPc. When spurious eigenvalues are present, the Lanczos process can converge to wrong values. This behavior appears at the same time as when the Lanczos vectors start to lose biorthogonality. As a possible remedy, one can apply so-called *reorthogonalization*, which is supported by our SLEPc implementation; see the work of Feuerriegel and Bücker (2013a) for more details.

Figure 3 compares the relative (left) and absolute (right) time savings per iteration when using the normalized s-step variant instead of the classical algorithm. Here, reorthogonalization is not used. The s-step algorithm performs slower than the classical algorithm for $p = 8$ and $p = 16$ processes. However, in this case it performs faster for both $p = 32$ and $p = 64$ processes. With 64 processes, for instance, the corresponding time saving per iteration accounts for 0.0042 s or $16.39\,\%$, respectively.

7.2. Numerical accuracy of synchronization-reducing linear solvers. In exact arithmetic, the s-step variants and their classical counterparts are mathematically equivalent. However, their numerical behavior can differ in finite-precision arithmetic. The relative residual norm for BiCG is plotted against the iteration number in Figure 4. The corresponding convergence behavior for QMR is shown in Figure 5. The findings are as follows: (i) the relative residual norms for classical and s-step algorithms are similar, (ii) an increasing iteration index n augments the numerical discrepancy between s-step and classical solvers, (iii) numerical instabilities grow with an increase in step size s—as the s-step Lanczos basis becomes more unstable, it can even result in the possibility of a breakdown.

[1]Each node of this cluster at RWTH Aachen University, Germany, consists of 2 sockets, each equipped with Intel Xeon X5570 quadcore processors running at 2.93 GHz. Each core has a separate L1 and L2 cache, while 4 cores share an L3 cache of size 8 MB. So each node of this cluster is made up of 8 cores called processes hereafter. The nodes are connected by a quad data rate InfiniBand network.

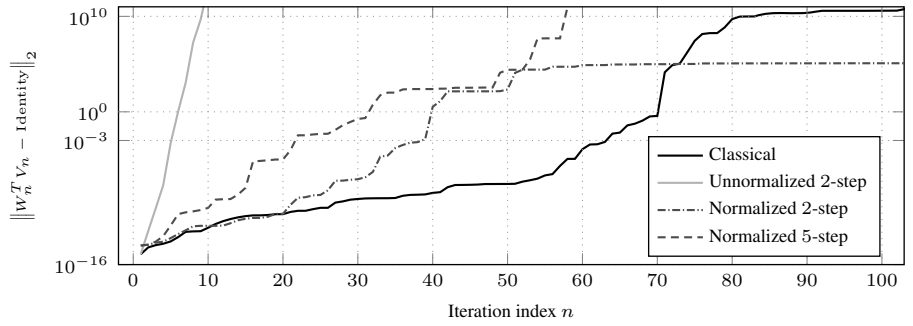


Fig. 2. Convergence history of the biorthogonality property for the classical algorithm, as well as for the unnormalized and the normalized s-step Lanczos variant.

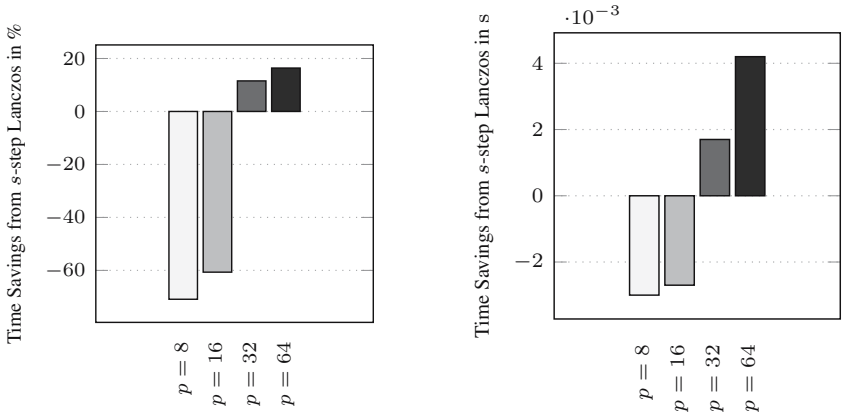


Fig. 3. Relative (left) and absolute (right) time savings per iteration, when using the normalized s-step variant instead of the classical algorithm. We choose $N = 4096$, $s = 2$ and vary the number of processes p.

7.3. Parallel performance of synchronization-reducing linear solvers. Figure 6 compares the ratio of the total time that is spent on various linear algebra operations. Due to barrier operations arising from GSPs, one would expect a gradual increase in the proportional influence of inner products. For the classical BiCG, the percentage of the total time resulting from inner products grows from around 12% for 64 processes to almost 47% for 512 processes. For the 2-step BiCG, however, there is only a moderate rise from approximately 8% to 13% when varying the processes from 64 to 512. As almost half of the total computation time using 512 processes is spent on the evaluation of inner products, the scalability to a large number of processes is limited for the classical BiCG. The 2-step BiCG variant scales significantly better.

Figure 7 compares the same linear algebra operations in terms of their absolute time consumption. The overall runtime of the classical BiCG is affected by inner products to a large extent. In contrast, the 2-step BiCG computes all inner products in less time. With 512 processes, the classical BiCG needs $0.76\,\mathrm{s}$ for computing all inner products, whereas the 2-step BiCG needs only $0.17\,\mathrm{s}$.

In Fig. 8, the average speedup for a single iteration is depicted. With an increasing number of processes, the speedup of both s-step solvers, BiCG and QMR, ascends more linearly when compared to the classical variants that begin to flatten out. Measurements for step sizes $s > 2$ are unavailable for more than 256 processes due to breakdowns. As a trend, however, we conclude that, given a network with a relatively time-consuming barrier operation, the scalability of the new s-step variants is significantly improved in comparison with the classical algorithms.

8. Concluding remarks

The purpose of s-step methods is to reduce the number of global synchronization points on distributed-memory computers by a factor of $\mathcal{O}(s)$. Rather than carrying out s separate iterations of a traditional method, these methods rely on using a single block iteration that is equivalent in increasing the dimension of the Krylov subspace, i.e., by restructuring the original algorithms in such a way that multiple inner products are grouped for joint execution. We derive a new s-step Lanczos algorithm with normalization of the underlying Krylov

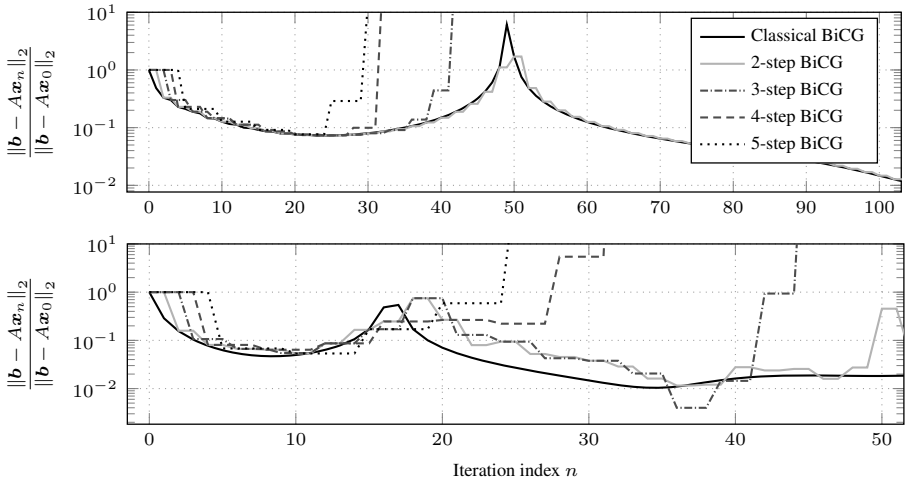


Fig. 4. Comparison of the classical BiCG method and s-step variants across different step sizes without (top) and with block Jacobi (bottom) preconditioning using two processes.

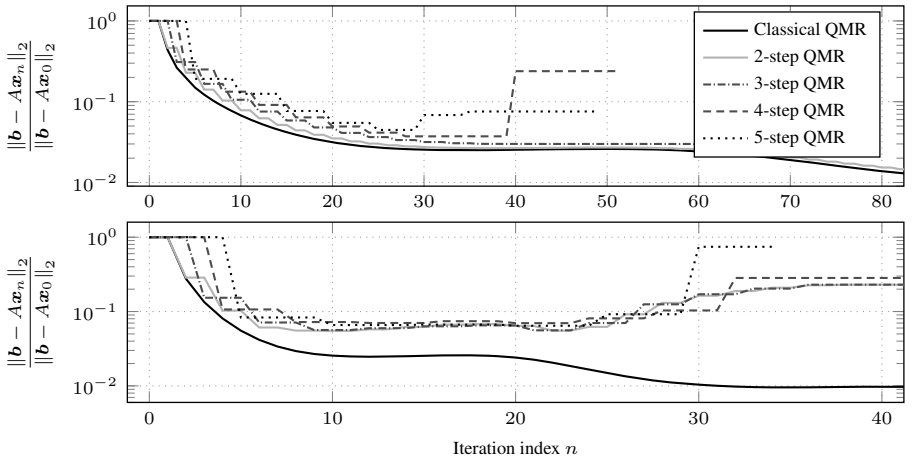


Fig. 5. Comparison of the classical QMR method and s-step variants across different step sizes without (top) and with block Jacobi (bottom) preconditioning using two processes.

basis using efficient recurrences. Numerical experiments indicate that this new normalized s-step variant—like the previous unnormalized s-step one—is more scalable than the traditional Lanczos algorithm. In addition, this new normalized variant exhibits improved numerical accuracy compared to a previous unnormalized variant. Consequently, this s-step Lanczos algorithm shows a possible path to advance parallel scalability on current large-scale and future extreme-scale supercomputers.

However, there is still room for further improvement. Most notably, the numerical stability tends to decrease with an increase in s. Future work is necessary to investigate promising remedies, such as the use of a basis that is different from the monomial basis (Carson *et al.*, 2014). Another viable alternative is to introduce residual replacement strategies (Carson and Demmel, 2014; van der Vorst and Ye, 2000). The techniques addressed by Gustafsson *et al.* (2012a) might also further improve the numerical stability. In addition, look-ahead techniques (Freund and Nachtigal, 1991; 1994; Freund and Hochbruck, 1991; 1992) are advantageous in preventing breakdowns.

Acknowledgment

Parts of this research were conducted while the authors were in residence at the Institute for Scientific Computing, the Center for Computational Engineering Science, and the Aachen Institute for Advanced Study in Computational Engineering Science at RWTH Aachen University, Germany. Financial support from Deutsche Forschungsgemeinschaft (German Research Foundation) through the grant GSC 111 is gratefully acknowledged.

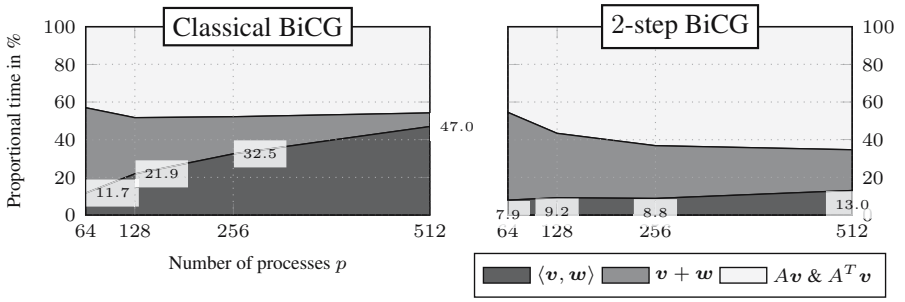


Fig. 6. Proportional time spent in linear algebra operations measured across 200 iterations including initialization for the classical BiCG (left) and the s-step BiCG method (right).

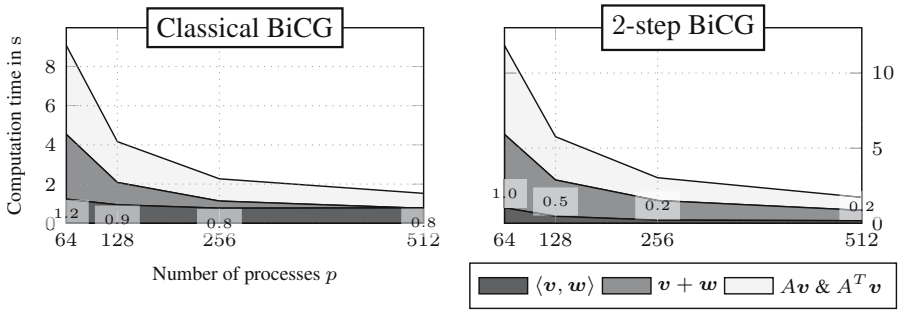


Fig. 7. Total time spent in linear algebra operations measured across 200 iterations including initialization for the classical BiCG (left) and the s-step BiCG method (right).

References

Balay, S., Gropp, W.D., McInnes, L.C. and Smith, B.F. (1997). Efficient management of parallelism in object oriented numerical software libraries, *in* E. Arge *et al.* (Eds.), *Modern Software Tools in Scientific Computing*, Birkhäuser Press, Boston, MA, pp. 163–202.

Bücker, H.M. (2002). Iteratively solving large sparse linear systems on parallel computers, *in* J. Grotendorst, D. Marx and A. Muramatsu (Eds.), *Quantum Simulations of Complex Many-Body Systems: From Theory to Algorithms*, NIC Series, Vol. 10, John Von Neumann Institute for Computing, Jülich, pp. 521–548.

Bücker, H.M. and Sauren, M. (1996). A parallel version of the quasi-minimal residual method based on coupled two-term recurrences, *in* J. Waśniewski *et al.* (Eds.), *Applied Parallel Computing*, Lecture Notes in Computer Science, Vol. 1184, Springer, Berlin, pp. 157–165.

Bücker, H.M. and Sauren, M. (1997). A variant of the biconjugate gradient method suitable for massively parallel computing, *in* G. Bilardi *et al.* (Eds.), *Solving Irregularly Structured Problems in Parallel*, Lecture Notes in Computer Science, Vol. 1253, Springer, Berlin, pp. 72–79.

Bücker, H.M. and Sauren, M. (1999). Reducing global synchronization in the biconjugate gradient method, *in* T. Yang (Ed.), *Parallel Numerical Computations with Applications*, Kluwer Academic Publishers, Norwell, MA, pp. 63–76.

Cappello, F., Geist, A., Gropp, B., Kale, L., Kramer, B. and Snir, M. (2009). Toward exascale resilience, *International Journal of High Performance Computing Applications* **23**(4): 374–388.

Carson, E. and Demmel, J. (2014). A residual replacement strategy for improving the maximum attainable accuracy of s-step Krylov subspace methods, *SIAM Journal on Matrix Analysis and Applications* **35**(1): 22–43.

Carson, E., Knight, N. and Demmel, J. (2014). Avoiding communication in nonsymmetric Lanczos-based Krylov subspace methods, *SIAM Journal on Scientific Computing* **35**(5): S42–S61.

Chronopoulos, A.T. (1986). A class of parallel iterative methods implemented on multiprocessors, *Technical report UIUCDCS-R-86-1267*, Department of Computer Science, University of Illinois, Urbana, IL.

Chronopoulos, A.T. and Gear, C.W. (1989). s-Step iterative methods for symmetric linear systems, *Journal of Computational and Applied Mathematics* **25**(2): 153–168.

Chronopoulos, A.T. and Swanson, C.D. (1996). Parallel iterative s-step methods for unsymmetric linear systems, *Parallel Computing* **22**(5): 623–641.

Curfman McInnes, L., Smith, B., Zhang, H. and Mills, R.T. (2014). Hierarchical Krylov and nested Krylov methods for extreme-scale computing, *Parallel Computing* **40**(1): 17–31.

Davis, N.E., Robey, R.W., Ferenbaugh, C.R., Nicholaeff, D. and Trujillo, D.P. (2012). Paradigmatic shifts for exascale supercomputing, *Journal of Supercomputing* **62**(2): 1023–1044.

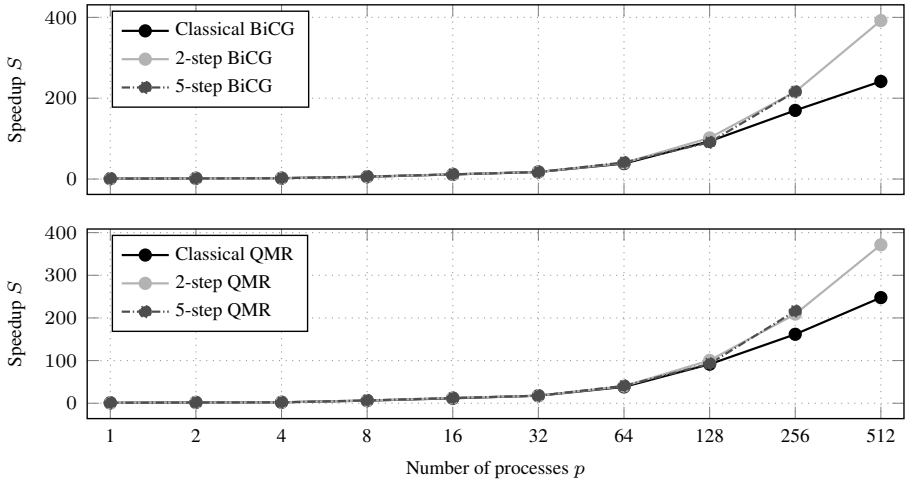


Fig. 8. Average speedup $S = T_{\mathrm{par}}(1)/T_{\mathrm{par}}(p)$ for a single iteration of BiCG (top) and QMR (bottom) averaged across 200 iterations excluding initialization time.

Demmel, J., Heath, M. and van der Vorst, H. (1993). Parallel numerical linear algebra, *Acta Numerica* **2**(1): 111–197.

Duff, I.S. (2012). European exascale software initiative: Numerical libraries, solvers and algorithms, *in* D. Hutchison *et al.* (Eds.), *Euro-Par 2011: Parallel Processing Workshops*, Lecture Notes in Computer Science, Vol. 7155, Springer, Berlin, pp. 295–304.

Duff, I.S. and van der Vorst, H.A. (1999). Developments and trends in the parallel solution of linear systems, *Parallel Computing* **25**(13–14): 1931–1970.

Feuerriegel, S. and Bücker, H.M. (2013a). A normalization scheme for the non-symmetric s-step Lanczos algorithm, *in* J. Kolodziej *et al.* (Eds.), *ICA3PP, Part II*, Lecture Notes in Computer Science, Vol. 8286, Springer, Berlin, pp. 30–39.

Feuerriegel, S. and Bücker, H.M. (2013b). Synchronization-reducing variants of the biconjugate gradient and the quasi-minimal residual methods, *in* J. Kolodziej *et al.* (Eds.), *ICA3PP, Part I*, Lecture Notes in Computer Science, Vol. 8285, Springer, Berlin, pp. 226–235.

Fischer, B. and Freund, R. (1994). An inner product-free conjugate gradient-like algorithm for Hermitian positive definite systems, *in* J. Brown *et al.* (Eds.), *Cornelius Lanczos International Centenary Conference*, SIAM, Philadelphia, PA, pp. 288–290.

Fletcher, R. (1976). Conjugate gradient methods for indefinite systems, *in* G. Watson (Ed.), *Numerical Analysis*, Lecture Notes in Computer Science, Vol. 506, Springer, Berlin, pp. 73–89.

Freund, R. and Nachtigal, N. (1991). QMR: A quasi-minimal residual method for non-Hermitian linear systems, *Numerische Mathematik* **60**(1): 315–339.

Freund, R.W. and Hochbruck, M. (1991). A biconjugate gradient type algorithm on massively parallel architectures, *in* R. Vichnevetsky and J.J.H. Miller (Eds.), *IMACS'91*, Criterion Press, Dublin, pp. 720–721.

Freund, R.W. and Hochbruck, M. (1992). A biconjugate gradient-type algorithm for the iterative solution of non-Hermitian linear systems on massively parallel architectures, *in* C. Brezinski and U. Kulisch (Eds.), *IMACS'91*, North Holland, Amsterdam, pp. 169–178.

Freund, R.W. and Nachtigal, N.M. (1994). An implementation of the QMR method based on coupled two-term recurrences, *SIAM Journal on Scientific Computing* **15**(2): 313.

Ghysels, P., Ashby, T.J., Meerbergen, K. and Vanroose, W. (2013). Hiding global communication latency in the GMRES algorithm on massively parallel machines, *SIAM Journal on Scientific Computing* **35**(1): C48–C71.

Ghysels, P. and Vanroose, W. (2014). Hiding global synchronization latency in the preconditioned conjugate gradient algorithm, *Parallel Computing* **40**(7): 224–238.

Gustafsson, M., Demmel, J. and Holmgren, S. (2012a). Numerical evaluation of the communication-avoiding Lanczos algorithm, *Technical Report 2012-001*, Department of Information Technology, Uppsala University, Uppsala.

Gustafsson, M., Kormann, K. and Holmgren, S. (2012b). Communication-efficient algorithms for numerical quantum dynamics, *in* K. Jónasson (Ed.), *Applied Parallel and Scientific Computing,* Lecture Notes in Computer Science, Vol. 7134, Springer, Berlin, pp. 368–378.

Gutknecht, M.H. (1997). Lanczos-type solvers for nonsymmetric linear systems of equations, *Acta Numerica* **6**(1): 271–397.

Hernandez, V., Roman, J.E. and Vidal, V. (2005). SLEPc: A scalable and flexible toolkit for the solution of eigenvalue problems, *ACM Transactions on Mathematical Software* **31**(3): 351–362.

Hoemmen, M.F. (2010). *Communication-avoiding Krylov Subspace Methods*, Ph.D. thesis, EECS Department, University of California, Berkeley, CA.

Kandalla, K., Yang, U., Keasler, J., Kolev, T., Moody, A., Subramoni, H., Tomko, K., Vienne, J., De Supinski, B. and Panda, D. (2012). Designing non-blocking allreduce with collective offload on InfiniBand clusters: A case study with conjugate gradient solvers, *Proceedings of the 2012 IEEE 26th International Parallel Distributed Processing Symposium (IPDPS), Shanghai, China*, pp. 1156–1167.

Kim, S.K. (2010). Efficient biorthogonal Lanczos algorithm on message passing parallel computer, *in* C.H. Hsu and V. Malyshkin (Eds.), *MTPP 2010*, Lecture Notes in Computer Science, Vol. 6083, Springer, Berlin, pp. 293–299.

Kim, S.K. and Chronopoulos, A. (1991). A class of Lanczos-like algorithms implemented on parallel computers, *Parallel Computing* **17**(6–7): 763–778.

Kim, S.K. and Chronopoulos, A.T. (1992). An efficient nonsymmetric Lanczos method on parallel vector computers, *Journal of Computational and Applied Mathematics* **42**(3): 357–374.

Kim, S.K. and Kim, T.H. (2005). A study on the efficient parallel block Lanczos method, *in* J. Zhang, J.-H. He and Y. Fu (Eds.), *Computational and Information Science*, Lecture Notes in Computer Science, Vol. 3314, Springer, Berlin/Heidelberg, pp. 231–237.

Lanczos, C. (1950). An iteration method for the solution of the eigenvalue problem of linear differential and integral operators, *Journal of Research of the National Bureau of Standards* **45**(4): 255–282.

Meurant, G. (1986). The conjugate gradient method on supercomputers, *Supercomputer* **13**: 9–17.

Mohiyuddin, M., Hoemmen, M., Demmel, J. and Yelick, K. (2009). Minimizing communication in sparse matrix solvers, *Conference on High Performance Computing Networking, Storage and Analysis, Portland, OR, USA*, pp. 36:1–36:12.

Saad, Y. (1989). Krylov subspace methods on supercomputers, *SIAM Journal on Scientific and Statistical Computing* **10**(6): 1200–1232.

Sauren, M. and Bücker, H.M. (1998). On deriving the quasi-minimal residual method, *SIAM Review* **40**(4): 922–926.

Shalf, J., Dosanjh, S. and Morrison, J. (2011). Exascale computing technology challenges, *in* D. Hutchison *et al.* (Eds.), *High Performance Computing for Computational Science, VECPAR 2010*, Lecture Notes in Computer Science, Vol. 6449, Springer, Berlin, pp. 1–25.

van der Vorst, H. (1990). Iterative methods for the solution of large systems of equations on supercomputers, *Advances in Water Resources* **13**(3): 137–146.

van der Vorst, H.A. and Ye, Q. (2000). Residual replacement strategies for Krylov subspace iterative methods for the convergence of true residuals, *SIAM Journal on Scientific Computing* **22**(3): 835–852.

Van Rosendale, J. (1983). Minimizing inner product data dependencies in conjugate gradient iteration, *NASA Contractor Report NASA-CR-172178*, NASA Langley Research Center, Hampton, VA.

Zhu, S.-X., Gu, T.-X. and Liu, X.-P. (2014). Minimizing synchronizations in sparse iterative solvers for distributed supercomputers, *Computers & Mathematics with Applications* **67**(1): 199–209.

Zuo, X., Gu, T.-X. and Mo, Z. (2010). An improved GPBi-CG algorithm suitable for distributed parallel computing, *Applied Mathematics and Computation* **215**(12): 4101–4109.

Stefan Feuerriegel graduated from the Aachen Institute for Advanced Study in Computational Engineering Science (AICES) at RWTH Aachen University in 2011. Since then, he has been a Ph. D. student at the Chair of Information Systems Research of the University of Freiburg with a focus on big data analytics and text mining.

H. Martin Bücker is currently a full professor in the Department of Mathematics and Computer Science, Friedrich Schiller University Jena, Germany. Prior to this, he was a researcher at Forschungszentrum Jülich, a visiting scientist at Argonne National Laboratory, and a senior researcher at RWTH Aachen University. His research focuses on parallel algorithms, high-performance computing, combinatorial scientific computing, and automatic differentiation.

Appendix
Deriving efficient recurrence equations

Recurrence for the normalization factor. Recall from Section 4.2 that the normalization scheme guarantees $\langle \boldsymbol{w}_k^j, \boldsymbol{v}_k^j \rangle = \pm 1$ by simply scaling the vectors $\tilde{\boldsymbol{v}}_k^j$ and $\tilde{\boldsymbol{w}}_k^j$ by scalars σ_k^j. We formally define $\sigma_k^1 := 1$. For all other $j = 2, \ldots, s$, this normalization factor can be rewritten as

$$\sigma_k^j = |\langle \tilde{\boldsymbol{w}}_k^j, \tilde{\boldsymbol{v}}_k^j \rangle|^{-\frac{1}{2}} \overset{(29)}{=} \left| \frac{\sigma_k^j \sigma_k^j}{\overline{M}_k^{j,j}} \right|^{\frac{1}{2}}$$

$$\overset{(A5)}{=} \left| \langle \boldsymbol{w}_k^1, A^{2j-2} \boldsymbol{v}_k^1 \rangle - \left(\boldsymbol{t}_{k-1}^j \right)^T \overline{M}_{k-1} \boldsymbol{t}_{k-1}^j \right|^{-\frac{1}{2}}.$$

Defining auxiliary variables $\Omega_k^{i,j}$, $\Xi_k^{i,j}$ and $\Psi_k^{i,j}$. To simplify the subsequent derivation of the recurrence relations, we introduce three auxiliary variables. The first variable is given by

$$\begin{aligned}
\Omega_k^{i,j} &= \langle \boldsymbol{w}_k^1, A^{i-1} \overline{V}_{k-1} \boldsymbol{t}_{k-1}^j \rangle \\
&= \langle \left(A^T\right)^{i-1} \boldsymbol{w}_k^1, \overline{V}_{k-1} \boldsymbol{t}_{k-1}^j \rangle \\
&\overset{(32)}{=} \langle \tilde{\boldsymbol{w}}_k^i + \overline{W}_{k-1} \boldsymbol{t}_{k-1}^i, \overline{V}_{k-1} \boldsymbol{t}_{k-1}^j \rangle \\
&\overset{(10)}{=} \langle \overline{W}_{k-1} \boldsymbol{t}_{k-1}^i, \overline{V}_{k-1} \boldsymbol{t}_{k-1}^j \rangle \\
&\overset{(29)}{=} \left(\boldsymbol{t}_{k-1}^i \right)^T \overline{M}_{k-1} \boldsymbol{t}_{k-1}^j.
\end{aligned}$$

In addition, we let the second variable be

$$\begin{aligned}\Xi_k^{i,j} &= \langle \overline{W}_{k-1}\boldsymbol{t}_{k-1}^i, A^{j-1}\boldsymbol{v}_k^1\rangle \\ &\overset{(33)}{=} \langle \overline{W}_{k-1}\boldsymbol{t}_{k-1}^i, \tilde{\boldsymbol{v}}_k^j + \overline{V}_{k-1}\boldsymbol{t}_{k-1}^j\rangle \\ &\overset{(10)}{=} \left(\boldsymbol{t}_{k-1}^i\right)^T \overline{M}_{k-1}\boldsymbol{t}_{k-1}^j.\end{aligned}$$

Finally, the third auxiliary variable is given by

$$\begin{aligned}\Psi_k^{i,j} &= (\boldsymbol{g}_k^s)^T\langle \overline{W}_k, A^{i+j-s-2}\boldsymbol{v}_{k+1}^1\rangle \\ &= \sum_{\iota=1}^{s} \boldsymbol{g}_k^{\iota,s}\langle \boldsymbol{w}_k^\iota, A^{i+j-s-2}\boldsymbol{v}_{k+1}^1\rangle \\ &\overset{(33)}{=} \sum_{\iota=1}^{s} \sigma_k^\iota \boldsymbol{g}_k^{\iota,s} \\ &\quad \langle (A^T)^{\iota-1}\boldsymbol{w}_k^1 - \overline{W}_{k-1}\boldsymbol{t}_{k-1}^\iota, A^{i+j-s-2}\boldsymbol{v}_{k+1}^1\rangle \\ &\overset{(32)}{=} \sum_{\iota=1}^{s} \sigma_k^\iota \boldsymbol{g}_k^{\iota,s}\Big[\langle \boldsymbol{w}_k^1, A^{i+j-s-3+\iota}\boldsymbol{v}_{k+1}^1\rangle \\ &\quad - \underbrace{\langle \overline{W}_{k-1}\boldsymbol{t}_{k-1}^\iota, \tilde{\boldsymbol{v}}_{k+1}^{i+j-s-1} + \overline{V}_k\boldsymbol{t}_k^{i+j-s-1}\rangle}_{=0}\Big] \\ &\overset{(A3)}{=} \sum_{\iota=1}^{s} \frac{\sigma_k^\iota \boldsymbol{g}_k^{\iota,s}\boldsymbol{b}_k^{i-s+\iota-1,j}}{\sigma_k^{i-s+\iota-1}} \\ &= \sum_{\substack{\iota=2s\\ +3-i-j}}^{s} \frac{\sigma_k^\iota \boldsymbol{g}_k^{\iota,s}\boldsymbol{b}_k^{i-s+\iota-1,j}}{\sigma_k^{i-s+\iota-1}}. \qquad \text{(A1)}\end{aligned}$$

Here, we entered in the last step the non-zero pattern of $\boldsymbol{b}_k^{i,j}$ from Eqn. (A2) in order to adjust the summation. In order to access only non-zero entries, the sum of both indices in $\boldsymbol{b}_k^{i-s+\iota-1,j}$ must be greater than or equal to $s+2$, i.e.,

$$(i-s+\iota-1)+j \geq s+2 \quad \Leftrightarrow \quad \iota \geq 2s+3-i-j.$$

Recurrence for $\boldsymbol{b}_k^{i,j}$. The following recurrence holds:

$$\begin{aligned}\boldsymbol{b}_k^{i,j} &= \langle \boldsymbol{w}_k^i, A^{j-1}\boldsymbol{v}_{k+1}^1\rangle \\ &\overset{(33)}{=} \sigma_k^i\langle (A^T)^{i-1}\boldsymbol{w}_k^1, A^{j-1}\boldsymbol{v}_{k+1}^1\rangle \\ &\quad - \sigma_k^i\langle \overline{W}_{k-1}\boldsymbol{t}_{k-1}^i, A^{j-1}\boldsymbol{v}_{k+1}^1\rangle \\ &\overset{(32)}{=} \sigma_k^i\langle \boldsymbol{w}_k^1, A^{i+j-2}\boldsymbol{v}_{k+1}^1\rangle \\ &\quad - \sigma_k^i\underbrace{\langle \overline{W}_{k-1}\boldsymbol{t}_{k-1}^i, \tilde{\boldsymbol{v}}_{k+1}^j + \overline{V}_k\boldsymbol{t}_k^j\rangle}_{=0} \\ &= \sigma_k^i\langle \boldsymbol{w}_k^1, A^{i+j-2}\boldsymbol{v}_{k+1}^1\rangle \\ &= \sigma_k^i\langle (A^T)^{s-1}\boldsymbol{w}_k^1, A^{i+j-s-1}\boldsymbol{v}_{k+1}^1\rangle \\ &\overset{(33)}{=} \frac{\sigma_k^i}{\sigma_k^s}\langle \boldsymbol{w}_k^s + \overline{W}_{k-1}\boldsymbol{t}_{k-1}^s, A^{i+j-s-1}\boldsymbol{v}_{k+1}^1\rangle \\ &\overset{(32)}{=} \frac{\sigma_k^i}{\sigma_k^s}\langle \boldsymbol{w}_k^s, A^{i+j-s-1}\boldsymbol{v}_{k+1}^1\rangle \\ &\quad + \frac{\sigma_k^i}{\sigma_k^s}\underbrace{\langle \overline{W}_{k-1}\boldsymbol{t}_{k-1}^s, \tilde{\boldsymbol{v}}_{k+1}^{i+j-s} + \overline{V}_k\boldsymbol{t}_k^{i+j-s}\rangle}_{=0} \\ &= \frac{\sigma_k^i}{\sigma_k^s}\langle A^T\boldsymbol{w}_k^s, A^{i+j-s-2}\boldsymbol{v}_{k+1}^1\rangle \\ &\overset{(30)}{=} \frac{\sigma_k^i}{\sigma_k^s}\langle \tilde{\boldsymbol{w}}_{k+1}^1 + \overline{W}_{k-1}\boldsymbol{e}_k^s \\ &\quad + \overline{W}_k\boldsymbol{g}_k^s, A^{i+j-s-2}\boldsymbol{v}_{k+1}^1\rangle \\ &= \frac{\sigma_k^i}{\sigma_k^s}\langle \tilde{\boldsymbol{w}}_{k+1}^1, A^{i+j-s-2}\boldsymbol{v}_{k+1}^1\rangle \\ &\quad + \frac{\sigma_k^i}{\sigma_k^s}\langle \overline{W}_{k-1}\boldsymbol{e}_k^s, A^{i+j-s-2}\boldsymbol{v}_{k+1}^1\rangle \\ &\quad + \frac{\sigma_k^i}{\sigma_k^s}(\boldsymbol{g}_k^s)^T\langle \overline{W}_k, A^{i+j-s-2}\boldsymbol{v}_{k+1}^1\rangle \\ &\qquad \text{cf. Step 13 in Algorithm 2} \\ &\overset{(32)}{=} \frac{\sigma_k^i}{\sigma_k^s}\langle \tilde{\boldsymbol{w}}_{k+1}^1, A^{i+j-s-2}\boldsymbol{v}_{k+1}^1\rangle \\ &\quad + \frac{\sigma_k^i}{\sigma_k^s}\underbrace{\langle \overline{W}_{k-1}\boldsymbol{e}_k^s, \tilde{\boldsymbol{v}}_{k+1}^{i+j-s-1} + \overline{V}_k\boldsymbol{t}_k^{i+j-s-1}\rangle}_{=0} \\ &\quad + \frac{\sigma_k^i}{\sigma_k^s}\Psi_k^{i,j} \\ &\overset{(A1)}{=} \frac{\sigma_k^i}{\sigma_k^s}\Bigg[\langle f_{k+1}\boldsymbol{w}_{k+1}^1, A^{i+j-s-2}\boldsymbol{v}_{k+1}^1\rangle \\ &\quad + \sum_{\substack{\iota=2s\\ +3-i-j}}^{s} \frac{\sigma_k^\iota \boldsymbol{g}_k^{\iota,s}\boldsymbol{b}_k^{i-s+\iota-1,j}}{\sigma_k^{i-s+\iota-1}}\Bigg]\end{aligned}$$

and then the matrix $\boldsymbol{b}_k$ features the following structure:

$$\boldsymbol{b}_k = \begin{bmatrix} 0 & & & 0 \\ & \ddots & 0 & \beta_1 \\ & 0 & \ddots & \vdots \\ 0 & \beta_1 & \cdots & \beta_{s-1} \end{bmatrix}. \qquad \text{(A2)}$$

First, the anti-diagonal shape origins from the above equation:

$$\boldsymbol{b}_k^{i,j} = \sigma_k^i\langle \boldsymbol{w}_k^1, A^{i+j-2}\boldsymbol{v}_{k+1}^1\rangle, \qquad \text{(A3)}$$

with $i+j = \text{const}$, as the sum $i+j$ determines the values on the anti-diagonals in Eqn. (A2). Second, the non-zero

pattern comes from

$$\begin{aligned}
\boldsymbol{b}_k^{i,j} &= \langle \boldsymbol{w}_k^i, A^{j-1}\boldsymbol{v}_{k+1}^1\rangle = \langle (A^T)^{j-1}\boldsymbol{w}_k^i, \boldsymbol{v}_{k+1}^1\rangle \\
&= \sigma_k^i \langle (A^T)^{j-1}\tilde{\boldsymbol{w}}_k^i, \boldsymbol{v}_{k+1}^1\rangle \\
&\overset{(33)}{=} \sigma_k^i \langle (A^T)^{j-1}\left[(A^T)^{i-1}\boldsymbol{w}_k^1 - \overline{W}_{k-1}\boldsymbol{t}_{k-1}^i\right], \boldsymbol{v}_{k+1}^1\rangle \\
&= \sigma_k^i \langle (A^T)^{j+i-2}\boldsymbol{w}_k^1 - (A^T)^{j-1}\overline{W}_{k-1}\boldsymbol{t}_{k-1}^i, \boldsymbol{v}_{k+1}^1\rangle \\
&\overset{(33)}{=} \sigma_k^i \langle \tilde{\boldsymbol{w}}_k^{i+j-1} + \overline{W}_{k-1}\boldsymbol{t}_{k-1}^{i+j-1} \\
&\quad - (A^T)^{j-1}\overline{W}_{k-1}\boldsymbol{t}_{k-1}^i, \boldsymbol{v}_{k+1}^1\rangle \\
&= \sigma_k^i \Big[\langle \tilde{\boldsymbol{w}}_k^{i+j-1}, \boldsymbol{v}_{k+1}^1\rangle + \underbrace{\langle \overline{W}_{k-1}\boldsymbol{t}_{k-1}^{i+j-1}, \boldsymbol{v}_{k+1}^1\rangle}_{=0} \\
&\quad - \langle (A^T)^{j-1}\overline{W}_{k-1}\boldsymbol{t}_{k-1}^i, \boldsymbol{v}_{k+1}^1\rangle\Big] \\
&= \sigma_k^i \left[\langle \tilde{\boldsymbol{w}}_k^{i+j-1}, \boldsymbol{v}_{k+1}^1\rangle - \langle \overline{W}_{k-1}\boldsymbol{t}_{k-1}^i, A^{j-1}\boldsymbol{v}_{k+1}^1\rangle\right] \\
&\overset{(32)}{=} \sigma_k^i \Big[\langle \tilde{\boldsymbol{w}}_k^{i+j-1}, \boldsymbol{v}_{k+1}^1\rangle \\
&\quad - \underbrace{\langle \overline{W}_{k-1}\boldsymbol{t}_{k-1}^i, \tilde{\boldsymbol{v}}_{k+1}^j + \overline{V}_k\boldsymbol{t}_k^j\rangle}_{=0}\Big] \\
&= \sigma_k^i \cdot 0 = 0 \qquad \text{for } i+j-1 \le s.
\end{aligned}$$

Recurrence for $\boldsymbol{c}_k^{i,j}$. The matrix $\boldsymbol{c}_k$, whose entries are the elements $\boldsymbol{c}_k^{i,j}$, features the following non-zero pattern:

$$\boldsymbol{c}_k = \left[\begin{array}{ccc} \vdots & & \vdots \\ 0 & \cdots & 0 \\ \hline \boldsymbol{c}_k^{s,1} & \cdots & \boldsymbol{c}_k^{s,s} \end{array}\right]. \tag{A4}$$

We prove the above structure by induction. If $\boldsymbol{c}_0 := 0_{s,s}$, then it is sufficient to focus on the induction step from $k-1$ to k. Assuming the structure holds for $\boldsymbol{c}_k^{i,j}$, we derive

$$\begin{aligned}
\boldsymbol{c}_k^{i,j} &= \langle \boldsymbol{w}_{k-1}^i, A\boldsymbol{v}_k^j\rangle = \langle A^T\boldsymbol{w}_{k-1}^i, \boldsymbol{v}_k^j\rangle \\
&= \sigma_{k-1}^i \langle A^T\tilde{\boldsymbol{w}}_{k-1}^i, \boldsymbol{v}_k^j\rangle \\
&\overset{(33)}{=} \sigma_{k-1}^i \langle A^T\left[(A^T)^{i-1}\boldsymbol{w}_{k-1}^1 - \overline{W}_{k-2}\boldsymbol{t}_{k-2}^i\right], \boldsymbol{v}_k^j\rangle \\
&= \sigma_{k-1}^i \langle (A^T)^i\boldsymbol{w}_{k-1}^1 - A^T\overline{W}_{k-2}\boldsymbol{t}_{k-2}^i, \boldsymbol{v}_k^j\rangle \\
&\overset{(33)}{=} \sigma_{k-1}^i \big[\langle \tilde{\boldsymbol{w}}_{k-1}^{i+1} + \overline{W}_{k-2}\boldsymbol{t}_{k-2}^{i+1}, \boldsymbol{v}_k^j\rangle \\
&\quad - \langle A^T\overline{W}_{k-2}\boldsymbol{t}_{k-2}^i, \boldsymbol{v}_k^j\rangle\big] \\
&= \sigma_{k-1}^i \big[\langle \tilde{\boldsymbol{w}}_{k-1}^{i+1}, \boldsymbol{v}_k^j\rangle + \underbrace{\langle \overline{W}_{k-2}\boldsymbol{t}_{k-2}^{i+1}, \boldsymbol{v}_k^j\rangle}_{=0} \\
&\quad - \langle \overline{W}_{k-2}\boldsymbol{t}_{k-2}^i, A\boldsymbol{v}_k^j\rangle\big] \\
&\overset{(32)}{=} \sigma_{k-1}^i \Big[\langle \tilde{\boldsymbol{w}}_{k-1}^{i+1}, \boldsymbol{v}_k^j\rangle \\
&\quad - \sigma_k^j \langle \overline{W}_{k-2}\boldsymbol{t}_{k-2}^i, A\big[A^{j-1}\boldsymbol{v}_k^1 - \overline{V}_{k-1}\boldsymbol{t}_{k-1}^j\big]\rangle\Big] \\
&\overset{(32)}{=} \sigma_{k-1}^i \Big[\langle \tilde{\boldsymbol{w}}_{k-1}^{i+1}, \boldsymbol{v}_k^j\rangle - \sigma_k^j \langle \overline{W}_{k-2}\boldsymbol{t}_{k-2}^i, \\
&\quad \boldsymbol{v}_k^{j+1} + \overline{V}_{k-1}\boldsymbol{t}_{k-1}^{j+1} - A\overline{V}_{k-1}\boldsymbol{t}_{k-1}^j\rangle\Big] \\
&= \sigma_{k-1}^i \langle \tilde{\boldsymbol{w}}_{k-1}^{i+1}, \boldsymbol{v}_k^j\rangle \\
&\quad - \sigma_{k-1}^i\sigma_k^j \underbrace{\langle \overline{W}_{k-2}\boldsymbol{t}_{k-2}^i, \boldsymbol{v}_k^{j+1} + \overline{V}_{k-1}\boldsymbol{t}_{k-1}^{j+1}\rangle}_{=0} \\
&\quad + \sigma_{k-1}^i\sigma_k^j \langle \overline{W}_{k-2}\boldsymbol{t}_{k-2}^i, A\overline{V}_{k-1}\boldsymbol{t}_{k-1}^j\rangle \\
&\overset{(34)}{=} \sigma_{k-1}^i \left[\langle \tilde{\boldsymbol{w}}_{k-1}^{i+1}, \boldsymbol{v}_k^j\rangle + \sigma_k^j \big(\boldsymbol{t}_{k-2}^i\big)^T \boldsymbol{c}_{k-1}\boldsymbol{t}_{k-1}^j\right] \\
&= \sigma_{k-1}^i \cdot 0 = 0
\end{aligned}$$

for $i < s$ by induction hypotheses $\boldsymbol{c}_{k-1}^{i,j} = 0$ for all $i < s$. Furthermore,

$$\begin{aligned}
\boldsymbol{c}_k^{s,j} &= \langle \boldsymbol{w}_{k-1}^s, A\boldsymbol{v}_k^j\rangle \\
&\overset{(32)}{=} \sigma_k^j \langle \boldsymbol{w}_{k-1}^s, A\left[A^{j-1}\boldsymbol{v}_k^1 - \overline{V}_{k-1}\boldsymbol{t}_{k-1}^j\right]\rangle \\
&\overset{(35),(36)}{=} \sigma_k^j \left[\boldsymbol{b}_{k-1}^{s,j+1} - \boldsymbol{d}_{k-1}^{s,\bullet}\boldsymbol{t}_{k-1}^j\right]
\end{aligned}$$

holds where $\boldsymbol{d}_{k-1}^{s,\bullet}$ denotes the s-th row of $\boldsymbol{d}_{k-1}$. When computing $\boldsymbol{c}_k^{s,j}$ recursively, the additional value of $\boldsymbol{b}_{k-1}^{s,s+1}$ is unavoidably required, but which we also determine in the above recursive fashion.

Recurrence for $\boldsymbol{d}_k^{i,j}$. Next, we find

$$\begin{aligned}
\boldsymbol{d}_k^{i,j} &= \langle \boldsymbol{w}_k^i, A\boldsymbol{v}_k^j\rangle \\
&= \sigma_k^i\sigma_k^j \Big[\big\langle (A^T)^{i-1}\boldsymbol{w}_k^1 \\
&\quad - \overline{W}_{k-1}\boldsymbol{t}_{k-1}^i, A\left[A^{j-1}\boldsymbol{v}_k^1 - \overline{V}_{k-1}\boldsymbol{t}_{k-1}^j\right]\big\rangle\Big] \\
&= \sigma_k^i\sigma_k^j \Big[\big\langle (A^T)^{i-1}\boldsymbol{w}_k^1 \\
&\quad - \overline{W}_{k-1}\boldsymbol{t}_{k-1}^i, A^j\boldsymbol{v}_k^1 - A\overline{V}_{k-1}\boldsymbol{t}_{k-1}^j\big\rangle\Big] \\
&\overset{(35)}{=} \sigma_k^i\sigma_k^j \Big[\langle \boldsymbol{w}_k^1, A^{i+j-1}\boldsymbol{v}_k^1\rangle \\
&\quad - \Omega_k^{i+1,j} - \Xi_k^{i,j+1} + \big(\boldsymbol{t}_{k-1}^i\big)^T \boldsymbol{d}_{k-1}\boldsymbol{t}_{k-1}^j\Big].
\end{aligned}$$

We notice that the given expressions

$$\Omega_k^{i,j} = \left(\boldsymbol{t}_{k-1}^i\right)^T \overline{M}_{k-1} \boldsymbol{t}_{k-1}^j,$$
$$\Xi_k^{i,j} = \left(\boldsymbol{t}_{k-1}^i\right)^T \overline{M}_{k-1} \boldsymbol{t}_{k-1}^j$$

cannot be evaluated for both $\Omega_k^{s+1,j}$ and $\Xi_k^{i,s+1}$ as the variable $\boldsymbol{t}_{k-1}^{s+1}$ is undefined. Therefore, we need to derive alternative recursions for the special cases $\Omega_k^{s+1,j}$ and $\Xi_k^{i,s+1}$ as follows:

$$\begin{aligned}
\Omega_k^{s+1,j} &:= \langle \boldsymbol{w}_k^1, A^s \overline{V}_{k-1} \boldsymbol{t}_{k-1}^j \rangle \\
&= \langle (A^T)^{s-1} \boldsymbol{w}_k^1, A \overline{V}_{k-1} \boldsymbol{t}_{k-1}^j \rangle \\
&\overset{(33)}{=} \langle \frac{\boldsymbol{w}_k^s}{\sigma_k^s} + \overline{W}_{k-1} \boldsymbol{t}_{k-1}^s, A \overline{V}_{k-1} \boldsymbol{t}_{k-1}^j \rangle \\
&= \frac{1}{\sigma_k^s} \left(\overline{V}_{k-1}^T A^T \boldsymbol{w}_k^s \right)^T \boldsymbol{t}_{k-1}^j \\
&\quad + \langle \overline{W}_{k-1} \boldsymbol{t}_{k-1}^s, A \overline{V}_{k-1} \boldsymbol{t}_{k-1}^j \rangle \\
&= \frac{1}{\sigma_k^s} \left(\overline{W}_{k-1}^T A \boldsymbol{v}_k^s \right)^T \boldsymbol{t}_{k-1}^j \\
&\quad + \langle \overline{W}_{k-1} \boldsymbol{t}_{k-1}^s, A \overline{V}_{k-1} \boldsymbol{t}_{k-1}^j \rangle
\end{aligned}$$

after Kim and Chronopoulos (1992, p. 366). Then, using (34) and (35), we arrive at

$$\Omega_k^{s+1,j} = \frac{\left(\boldsymbol{c}_k^{\bullet,s}\right)^T \boldsymbol{t}_{k-1}^j}{\sigma_k^s} + \left(\boldsymbol{t}_{k-1}^s\right)^T \boldsymbol{d}_{k-1} \boldsymbol{t}_{k-1}^j,$$

where $\boldsymbol{c}_k^{\bullet,s}$ is the s-th column of $\boldsymbol{c}_k$. Using (A4), we finally find

$$\Omega_k^{s+1,j} = \frac{\boldsymbol{c}_k^{s,s} \boldsymbol{t}_{k-1}^{s,j}}{\sigma_k^s} + \left(\boldsymbol{t}_{k-1}^s\right)^T \boldsymbol{d}_{k-1} \boldsymbol{t}_{k-1}^j.$$

Similarly, we derive

$$\begin{aligned}
\Xi_k^{i,s+1} &:= \langle \overline{W}_{k-1} \boldsymbol{t}_{k-1}^i, A^s \boldsymbol{v}_k^1 \rangle \\
&\overset{(32)}{=} \langle \overline{W}_{k-1} \boldsymbol{t}_{k-1}^i, A \left[\frac{\boldsymbol{v}_k^s}{\sigma_k^s} + \overline{V}_{k-1} \boldsymbol{t}_{k-1}^s \right] \rangle \\
&= \frac{\langle \overline{W}_{k-1} \boldsymbol{t}_{k-1}^i, A \boldsymbol{v}_k^s \rangle}{\sigma_k^s} \\
&\quad + \langle \overline{W}_{k-1} \boldsymbol{t}_{k-1}^i, A \overline{V}_{k-1} \boldsymbol{t}_{k-1}^s \rangle \\
&\overset{(34),(35)}{=} \frac{\left(\boldsymbol{t}_{k-1}^i\right)^T \boldsymbol{c}_k^{\bullet,s}}{\sigma_k^s} + \left(\boldsymbol{t}_{k-1}^i\right)^T \boldsymbol{d}_{k-1} \boldsymbol{t}_{k-1}^s \\
&\overset{(A4)}{=} \frac{\boldsymbol{t}_{k-1}^{s,i} \boldsymbol{c}_k^{s,s}}{\sigma_k^s} + \left(\boldsymbol{t}_{k-1}^i\right)^T \boldsymbol{d}_{k-1} \boldsymbol{t}_{k-1}^s.
\end{aligned}$$

Recurrence for $\overline{M}_k^{i,j}$. Finally, we have

$$\begin{aligned}
\overline{M}_k^{i,j} &= \langle \boldsymbol{w}_k^i, \boldsymbol{v}_k^j \rangle \\
&= \sigma_k^i \sigma_k^j \big\langle (A^T)^{i-1} \boldsymbol{w}_k^1 \\
&\quad - \overline{W}_{k-1} \boldsymbol{t}_{k-1}^i, A^{j-1} \boldsymbol{v}_k^1 - \overline{V}_{k-1} \boldsymbol{t}_{k-1}^j \big\rangle \\
&= \sigma_k^i \sigma_k^j \Big[\langle \boldsymbol{w}_k^1, A^{i+j-2} \boldsymbol{v}_k^1 \rangle \\
&\quad - \langle \boldsymbol{w}_k^1, A^{i-1} \overline{V}_{k-1} \boldsymbol{t}_{k-1}^j \rangle \\
&\quad - \langle \overline{W}_{k-1} \boldsymbol{t}_{k-1}^i, A^{j-1} \boldsymbol{v}_k^1 \rangle \\
&\quad + \langle \overline{W}_{k-1} \boldsymbol{t}_{k-1}^i, \overline{V}_{k-1} \boldsymbol{t}_{k-1}^j \rangle \Big] \\
&\overset{(29)}{=} \sigma_k^i \sigma_k^j \Big[\langle \boldsymbol{w}_k^1, A^{i+j-2} \boldsymbol{v}_k^1 \rangle - \Omega_k^{i,j} \\
&\quad - \Xi_k^{i,j} + \left(\boldsymbol{t}_{k-1}^i\right)^T \overline{M}_{k-1} \boldsymbol{t}_{k-1}^j \Big] \\
&= \sigma_k^i \sigma_k^j \Big[\langle \boldsymbol{w}_k^1, A^{i+j-2} \boldsymbol{v}_k^1 \rangle \\
&\quad - \left(\boldsymbol{t}_{k-1}^i\right)^T \overline{M}_{k-1} \boldsymbol{t}_{k-1}^j \Big], \qquad \text{(A5)}
\end{aligned}$$

where the last equation follows from

$$\Omega_k^{i,j} = \Xi_k^{i,j} = \left(\boldsymbol{t}_{k-1}^i\right)^T \overline{M}_{k-1} \boldsymbol{t}_{k-1}^j. \qquad \text{(A6)}$$

Received: 15 April 2014
Revised: 17 October 2014