Numerical evaluation of the Communication-Avoiding Lanczos algorithm

Magnus Gustafsson* James Demmel† Sverker Holmgren*

February 16, 2012

Abstract

The Lanczos algorithm is widely used for solving large sparse symmetric eigenvalue problems when only a few eigenvalues from the spectrum are needed. Due to sparse matrix-vector multiplications and frequent synchronization, the algorithm is communication intensive leading to poor performance on parallel computers and modern cache-based processors. The Communication-Avoiding Lanczos algorithm [Hoemmen; 2010] attempts to improve performance by taking the equivalence of s steps of the original algorithm at a time. The scheme is equivalent to the original algorithm in exact arithmetic but as the value of s grows larger, numerical roundoff errors are expected to have a greater impact. In this paper, we investigate the numerical properties of the Communication-Avoiding Lanczos (CA-Lanczos) algorithm and how well it works in practical computations. Apart from the algorithm itself, we have implemented techniques that are commonly used with the Lanczos algorithm to improve its numerical performance, such as semi-orthogonal schemes and restarting. We present results that show that CA-Lanczos is often as accurate as the original algorithm. In many cases, if the parameters of the s-step basis are chosen appropriately, the numerical behaviour of CA-Lanczos is close to the standard algorithm even though it is somewhat more sensitive to loosing mutual orthogonality among the basis vectors.

1 Introduction

Solving large sparse eigenvalue problems is an important and performance-critical task in a wide range of application areas. For Hermitian problems, the symmetric Lanczos algorithm [12] is often used, efficiently computing approximations to a few eigenvalues of a matrix by projecting the original problem on a lower-dimensional Krylov subspace, $\mathcal{K}_k(A, v) = \text{span}\left\{v, Av, A^2v, \ldots, A^{k-1}v\right\}$. In the iterative process, the Lanczos algorithm repeatedly computes the product of the sparse matrix with a dense vector (SpMV) and interleaves each of these SpMVs with BLAS 1 vector updates and inner products. On

*Division of Scientific Computing, Department of Information Technology, Uppsala University, Uppsala, Sweden (magnus.gustafsson@it.uu.se, sverker.holmgren@it.uu.se).

†Department of Electrical Engineering and Computer Sciences, University of California, Berkeley, CA, USA, (demmel@cs.berkeley.edu)

modern processor architectures, with deep cache hierarchies and limited memory bandwidth, the memory traffic incurred by the low utilization of data will be a performance bottleneck and the frequent global synchronization is a barrier for parallel scalability.

The Communication-Avoiding Lanczos algorithm (CA-Lanczos) [9] seeks to avoid the communication obstacles in the original Lanczos algorithm at several levels. One outer iteration of CA-Lanczos corresponds to a number of iterations (we denote it s) of the original Lanczos algorithm. In one outer iteration, a block of s SpMVs are computed together, leaving the opportunity to reuse data in the matrix between the products. Instead of orthogonalizing a single basis vector at a time using standard Gram-Schmidt, CA-Lanczos uses Block Gram-Schmidt (BGS) [20] and Tall Skinny QR (TSQR) [9], minimizing communication by sending as few messages as possible and by performing BLAS 3 operations instead of the BLAS 1 operations that are used in standard Gram-Schmidt orthogonalization.

The Lanczos algorithm in its original form uses a three term recurrence relation such that every computed basis vector is orthogonalized against just the previous two vectors. An issue with this is that mutual orthogonality among the basis vectors is eventually lost in finite precision arithmetic, leading to slow convergence and inaccurately computed eigenvalues [19, 18, 14, 15]. The immediate fix to this problem is to extend the algorithm to include all previous basis vectors in the orthogonalization. However, as the number of Lanczos iterations grows large, the computational effort to maintain full orthogonality among the basis vectors becomes increasingly demanding. To this end, several semi-orthogonal schemes have been developed that keep track of the loss of orthogonality and perform orthogonalization only when needed.

A related issue with the Lanczos algorithm is that the memory requirements grow with the number of vectors that we need to keep. With any reorthogonalization scheme we are forced to keep all vectors and for some computations we need not only the eigenvalues of the matrix but also the corresponding eigenvectors. For large matrices, we might not be able to store as many vectors as we need for the algorithm to converge. By restarting the method after a given number of iterations, discarding the vectors that do not contain the information we are after, we can alleviate these problems. We describe our implementation of an explicitly restarted Lanczos scheme, inspired by the method proposed by [7], adjusted for use with CA-Lanczos.

The main objective of this paper is to evaluate the numerical performance of CA-Lanczos with explicit restart and different reorthogonalization schemes that are often used with the Lanczos algorithm. We use matrices from various applications (the University of Florida Matrix collection [4] and show that CA-Lanczos is able to compute eigenvalues and eigenvectors to a given tolerance. We compare CA-Lanczos to the standard Lanczos algorithm and conclude that for the most part, CA-Lanczos is almost as accurate as standard Lanczos. We do not analyze the run time performance of CA-Lanczos in this paper, although we address some key performance aspects that motivate the use of CA-Lanczos.

This paper is organized as follows. First, in section 2, we give an overview of the previous work that has been done on the Lanczos algorithm in the light of communication avoiding and s-step methods as well as restarted methods. In section 3 we briefly describe the Lanczos algorithm followed by a discussion of its well-studied numerical properties in Section 4. The Communication-Avoiding Lanczos algorithm (CA-Lanczos) is outlined

in section 5, followed by some details regarding the semi-orthogonal techniques that we have implemented with the CA-Lanczos algorithm in section 6 and a discussion on explicit restarting of CA-Lanczos in section 7. Finally, in section 8 we present some numerical results, followed by concluding remarks in section 9.

2 Previous work

The Communication-Avoiding Lanczos algorithm was initially proposed by Hoemmen [9]. In his thesis, he discusses communication avoiding Krylov subspace methods in general and gives an in-depth analysis of a few chosen methods and their implementation. CA-Lanczos was derived and discussed in theory with regard to its expected numerical stability and performance. However, since eigenvalue problems were not the main focus of his thesis, CA-Lanczos was never implemented and analyzed in practical computations. This paper contributes to his work by assessing the numerical behaviour of the CA-Lanczos algorithm in different application problems.

The idea of combining several iterations of Krylov subspace methods in order to improve performance is not new. Kim and Chronopoulos [11] introduced s-step Lanczos and s-step Arnoldi methods, as well as s-step Conjugate Gradient [3]. However, their methods have shown to have stability problems [22], in particular when s grows large. Furthermore, their s-step Lanczos algorithm requires $s+1$ SpMVs in each outer iteration, as opposed to s SpMVs for an outer iteration of CA-Lanczos or s iterations of standard Lanczos. Thus, for large problems where the SpMV operation is the most time consuming part of the process, s-step Lanczos will loose much of its performance benefits. This is apparent in Gustafsson *et al.* [6], where the performance gains from avoiding synchronization was entirely consumed by the additional SpMV in each outer iteration.

Several schemes have been proposed for explicit restart of the Lanczos algorithm (see e.g. [10], [21], [23] and [7]). The explicitly restarted schemes all have in common that the restart vector is chosen from one of the Ritz vectors in each step. In contrast, Implicitly Restarted Lanczos [1] iteratively applies a polynomial filter to refine the restart vector. In this work we have chosen to implement a simple explicit scheme, although Implicitly Restarted Lanczos is regarded to be more stable and robust.

3 The symmetric Lanczos algorithm

Given a Hermitian $n \times n$ matrix A and a starting vector v, the Lanczos method [12] successively builds a basis of the Krylov subspace

$$\mathcal{K}_k(A, v) = \text{span}\left\{v, Av, A^2v, \ldots, A^{k-1}v\right\}. \tag{1}$$

Let $Q_m = [q_1, q_2, \ldots, q_m]$ be the orthonormal basis spanning $\mathcal{K}_m(A, q_1)$ after m steps of the Lanczos process, and let T_m denote the corresponding projection matrix of A onto $\mathcal{K}_m(A, q_1)$ such that

$$AQ_m = Q_m T_m + \beta_{m+1} q_{m+1} e_m^T. \tag{2}$$

Algorithm 1 The Lanczos Algorithm

1: $\beta_1 = \|v\|_2$, $q_0 = 0$, $q_1 = v/\beta_1$
2: **for** $j = 1$ to convergence **do**
3: $\quad w = Aq_j - \beta_j q_{j-1}$
4: $\quad \alpha_j = (w, q_j)$
5: $\quad w = w - \alpha_j q_j$
6: $\quad \beta_{j+1} = \|w\|_2$
7: $\quad q_{j+1} = w/\beta_{+1}$
8: **end for**

T_m is a real, symmetric and tridiagonal $m \times m$ matrix

$$T_m = \begin{pmatrix} \alpha_1 & \beta_2 & & & \\ \beta_2 & \alpha_2 & \beta_3 & & \\ & \ddots & \ddots & \ddots & \\ & & \beta_{m-1} & \alpha_{m-1} & \beta_m \\ & & & \beta_m & \alpha_m \end{pmatrix}. \tag{3}$$

Let θ_i denote the ith eigenvalue and s_i the corresponding eigenvector of T_m. Then (for all i) θ_i is an approximate eigenvalue (*Ritz value*) of A, and $y_i = Q_m s_i$ a corresponding approximate eigenvector (*Ritz vector*) of A. Since m is usually much smaller than n, and since T_m is tridiagonal, we can compute θ_i and s_i cheaply. The Ritz vectors y_i on the other hand are much more expensive to compute since they involve the entire basis matrix Q_m.

An outline of the symmetric Lanczos algorithm is presented in Algorithm 1. In k steps, the method computes k matrix-vector products and $O(k)$ BLAS 1 updates. The runtime performance will normally be dominated by the SpMV on line 3. In a parallel environment, also the inner products on line 4 and 6 will be cumbersome since they require global communication among the cores/processors. Furthermore, when the vectors are too large to fit in fast memory the BLAS 1 operations will generate excessive memory traffic. On modern architectures, this will lead to poor overall performance, since off-chip memory bandwidth is a scarce resource.

4 Orthogonality and convergence

A well-known and well-studied problem with the Lanczos algorithm is that mutual orthogonality among the basis vectors is eventually lost in finite precision arithmetic. Paige [14, 15] showed that loss of orthogonality occurs as soon as an eigenpair is close to convergence. If iterations are blindly continued with orthogonality lost, new Lanczos vectors will have components in the directions of already converged Ritz vectors and multiple copies of previously computed Ritz values will appear [21]. Furthermore, approximations of eigenvalues that are not true eigenvalues (*spurious* eigenvalues) of A may appear and mislead the results. For these reasons, lack of orthogonality among the basis vectors will slow down convergence and impair the accuracy.

Most practical computations using the Lanczos method require that the scheme is extended with some form of reorthogonalization of the basis vectors in addition to the

local orthogonalization of the three term recurrence. Which strategy is the most efficient will depend on the nature of the problem at hand and the requirements on the computed eigenvalues and eigenvectors. The crudest approach, in which every new basis vector is orthogonalized against all previous basis vectors, is referred to as *full orthogonalization*. This is the computationally most expensive option but it is also the most accurate since it maintains full orthogonality among all the basis vectors at all times. We consider full orthogonalization as the baseline result for accuracy but it is worth noting that full orthogonalization is often not viable in practice since it incurs a significant performance overhead when the number of Lanczos vectors grows large. Furthermore, orthogonality among the basis vectors to machine precision is not required for the Lanczos algorithm to be accurate and stable. It was shown by Paige [14, 15], Simon [19, 18] and Grcar [5] that it is sufficient for the orthogonalization error to be kept below $\sqrt{\epsilon_M}$, where ϵ_M is the machine roundoff error. This is further described in Section 6, where we discuss various *semi-orthogonal* techniques that are commonly used with the Lanczos algorithm.

5 The communication-avoiding symmetric Lanczos algorithm (CA-Lanczos)

The Krylov subspace sequence of vectors, $\{v, Av, A^2v, \ldots, A^sv\}$, gives good motivation for a scheme that breaks the data dependencies between iterations in the Lanczos algorithm and computes s SpMVs using a single kernel, potentially saving a factor of s reads of A from memory. The resulting method, referred to as the Communication-Avoiding Lanczos algorithm (CA-Lanczos), was derived in detail in [9] so we will only give a summary of it here.

5.1 Notation

We borrow the block-vector notation from [9], summarized as follows.

- Vectors and group of vectors:
  - v_k denotes a single vector
  - V_k denotes a group of s vectors in outer iteration k;
    e.g. $V_k = [v_{sk+1}, v_{sk+2}, \ldots, v_{sk+s}]$
  - $\mathfrak{V}_k$ denotes a collection of groups of vectors at outer iteration k;
    e.g. $\mathfrak{V}_k = [V_1, V_2, \ldots, V_k]$
- Groups of basis vectors:
  - Consider the matrix of basis vectors

    $$\underline{V}_k = [v_{sk+1}, v_{sk+2}, \ldots, v_{sk+s+1}].$$

  - We use the following short hand notation for groups of basis vectors, where underline implies an additional vector at the end and the accent means a shift

one step to the right.

$$
\begin{aligned}
V_k &= [v_{sk+1}, v_{sk+2}, \ldots, v_{sk+s}], \\
\underline{V}_k &= [V_k, v_{sk+s+1}], \\
\acute{V}_k &= [v_{sk+2}, \ldots, v_{sk+s}], \\
\underline{\acute{V}}_k &= [\acute{V}_k, v_{sk+s+1}].
\end{aligned}
$$

5.2 Computational kernels

In this section we describe the computational kernels on which the the CA-Lanczos algorithm relies.

5.2.1 Matrix powers kernel

The matrix powers kernel takes a sparse $n \times n$ matrix A, a dense vector of length n and a set of polynomials $p_0(z), p_1(z), \ldots, p_s(z)$, where $p_j(z)$ is of degree j, and computes

$$
\underline{V} = [v_1, v_2, \ldots, v_{s+1}] = [p_0(A)v, p_1(A)v, \ldots, p_s(A)v], \tag{4}
$$

where the vectors in $\underline{V}$ form a basis for the Krylov subspace

$$
\mathcal{K}_{s+1}(A, v) = \operatorname{span}\left\{v, Av, A^2v, \ldots, A^sv\right\}. \tag{5}
$$

Ideally, the matrix powers kernel requires just one message to be sent in parallel to compute all the vectors in (4). Furthermore, the matrix A and the vector v only need to be read once. This effectively leads to a factor of $\Theta(s)$ reduction in memory traffic and a factor of $\Theta(s)$ fewer messages in parallel.

By choosing the polynomial coefficients in (4), we can affect the numerical properties of the matrix powers kernel. The change of basis matrix $\underline{B}$, which is $s+1$ by s, should satisfy

$$
AV = \underline{VB}. \tag{6}
$$

For a detailed derivation and analysis of the different bases for the matrix powers kernel and the Leja orderings we refer to Hoemmen [9] and Carson *et al.* [2].

Monomial basis The simplest basis for the matrix powers kernel, the *monomial basis*, directly corresponds to the Krylov vector sequence with $p_k(A) = A^k$. We have

$$
\underline{V} = \left[v, Av, A^2v, \ldots, A^sv\right]. \tag{7}
$$

The corresponding change of basis matrix is

$$
\underline{B} = (e_2, e_3, \ldots, e_{s+1}). \tag{8}
$$

The monomial basis corresponds to taking s steps with the power method, which is inherently unstable since the basis vectors successively become more and more linearly dependent.

Newton basis The Newton basis applies polynomial shifts θ_i to the matrix powers kernel as follows

$$\underline{V} = \left[v, (A - \theta_1 I)v, (A - \theta_2 I)(A - \theta_1 I)v, \ldots, \prod_{i=1}^{s} (A - \theta_i I)v \right], \tag{9}$$

with the change of basis matrix

$$\underline{B} = \begin{pmatrix} \theta_1 & & & & \\ 1 & \theta_2 & & & \\ & \ddots & \ddots & & \\ & & 1 & \theta_s \\ & & & 1 \end{pmatrix}. \tag{10}$$

The shifts can be obtained from a set of approximate eigenvalues of A, ordered according to the *Leja ordering* for complex arithmetic, or the *modified Leja ordering* for real arithmetic. The approximate eigenvalues can be generated by taking a few steps with the original Lanczos algorithm if they are not available by other means.

5.2.2 Block Gram-Schmidt and Tall-Skinny QR

The original Lanczos algorithm uses Gram-Schmidt to orthogonalize the basis vectors one at a time. CA-Lanczos on the other hand, treats a block of s vectors simultaneously, and uses block Gram-Schmidt (BGS) to orthogonalize those s vectors with respect to the orthogonal basis of already computed Lanczos vectors [9, 20]. With a block size of s vectors, BGS saves a factor of $\Theta(s)$ in the number of messages sent in parallel and a factor of $\Theta(s)$ in the number of words transferred between slow and fast memory compared to standard Gram-Schmidt [9]. Throughout this work we have used Classical Gram-Schmidt in our implementation of BGS, although Modified Gram-Schmidt is considered to be more accurate. The motivation for this is that Classical Gram-Schmidt has more appealing parallel characteristics than Modified Gram-Schmidt [9, 20, 7]. However, both methods can be used interchangeably in the BGS kernel [9].

Using BGS to orthogonalize a block of vectors V_k against an orthogonal basis $\mathfrak{Q}_{k-1}$ makes the vectors in V_k orthogonal to those in $\mathfrak{Q}_{k-1}$, but the vectors in V_k are not made orthogonal to each other. We use Tall-skinny QR (TSQR) [9, 8] in order to obtain mutual orthogonality among the vectors in V_k, such that $V_k = Q_k R_{kk}$, where V_k and Q_k are $n \times m$ and R_{kk} is $m \times m$. TSQR has shown to be communication optimal for matrices where $n \gg m$ (i.e. many more rows than columns) [9, 13]. For an extensive discussion and analysis of TSQR, c.f. [9, 8].

If BGS and TSQR are used as described above to orthogonalize V_k against $\mathfrak{Q}_{k-1}$, it is not guaranteed that the generated basis vectors in Q_k are fully orthogonal to $\mathfrak{Q}_{k-1}$ [9, 20]. Those vectors for which the norm drops by more than a factor 2 in the orthogonalization step have to be reorthogonalized by doing a second pass of BGS [20]. If the vectors are still not orthogonal, we declare an orthogonalization fault (c.f. [20] for details on how to recover from such a situation).

Algorithm 2 BGS WITH TSQR AND REORTHOGONALIZATION

1: Compute $\rho(i) = ||V_k(:,i)||_2$, for $i = 1, \ldots, s$
2: $R_{1:k-1,k} = [Q_1, \ldots, Q_{k-1}]^* V_k$
3: $Y_k = V_k - [Q_1, \ldots, Q_{k-1}] R_{1:k-1,k}$
4: Compute the QR factorization $Y_k = Q_{Y_k} R_{Y_k}$ using TSQR
5: Compute $\sigma(i) = ||Y_k(:,i)||_2 = ||R_{Y_k}(:,i)||$, for $i = 1, \ldots, s$
6: **if** $\sigma(i)/\rho(i) < 0.5$ for any $i \in 1, \ldots, s$ **then**
7: $\quad R'_{1:k-1,k} = [Q_1, \ldots, Q_{k-1}]^* Y_k$
8: $\quad Z_k = Y_k - [Q_1, \ldots, Q_{k-1}] R'_{1:k-1,k}$
9: $\quad R_{1:k-1,k} = R_{1:k-1,k} + R'_{1:k-1,k}$
10: $\quad$ Compute the QR factorization $Z_k = Q_{Z_k} R_{Z_k}$ using TSQR
11: $\quad$ Compute $\tau(i) = ||Z_k(:,i)||_2 = ||R_{Z_k}(:,i)||$, for $i = 1, \ldots, s$
12: $\quad$ **if** $\tau(i)/\sigma(i) < 0.5$ for any $i \in 1, \ldots, s$ **then**
13: $\quad\quad$ Declare orthogonalization fault.
14: $\quad$ **else**
15: $\quad\quad Q_k = Q_{Z_k},\ R_{kk} = R_{Z_k}$
16: $\quad$ **end if**
17: **else**
18: $\quad Q_k = Q_{Y_k},\ R_{kk} = R_{Y_k}$
19: **end if**

5.3 Outline of the algorithm

The Communication-Avoiding Lanczos algorithm is presented in Algorithm 3. It makes use of the matrix powers kernel to compute s SpMVs (Sec. 5.2.1) and a combination of BGS and TSQR (Sec. 5.2.2) to orthogonalize s basis vectors in each iteration. The routine takes as input the matrix A and a vector v, and produces an orthogonal basis $\mathfrak{Q}_{\mathfrak{t}-\mathfrak{1}}$ and an upper Hessenberg (but approximately tridiagonal) projection matrix $\mathfrak{T}_{\mathfrak{t}-\mathfrak{1}}$.

In order to specify line 14 of Algorithm 3 we need to clarify some notation and rewrite the R-factors and the matrix $\underline{B}_k$. This derivation was done in detail by Hoemmen [9], but we summarize it in this section. We write the R-factors as follows

$$\underline{R}_{k-1,k} = \begin{pmatrix} e_{s+1} & \acute{\underline{R}}_{k-1,k} \end{pmatrix},$$
$$\underline{R}_k = \begin{pmatrix} e_1 & \acute{\underline{R}}_k \end{pmatrix},$$

where the last row of $\underline{R}_{k-1,k}$ and the first row of $\underline{R}_k$ are the same. The $s+1$ by $s+1$ matrix $\underline{R}_k$ can be further decomposed into

$$\underline{R}_k = \begin{pmatrix} R_k & z_k \\ 0 & \rho_k \end{pmatrix}, \quad \tilde{\rho}_k = R_k(s,s), \tag{11}$$

and similarly, $R_{k-1,k}$ is the s by s principal sub matrix of the $s+1$ by $s+1$ matrix $\underline{R}_{k-1,k}$. The $s+1$ by s change-of-basis matrix $\underline{B}_k$ is decomposed into

$$\underline{B}_k = \begin{pmatrix} B_k \\ b_k e_s^T \end{pmatrix}. \tag{12}$$

Algorithm 3 The CA-Lanczos Algorithm

1: $\beta_0 = \|v\|_2$, $q_1 = v/\beta_0$
2: **for** $k = 0$ to $t-1$ **do**
3: Fix basis conversion matrix $\underline{B}_k$
4: Compute $\underline{\acute{V}}_k$ from q_{sk+1} and A using matrix powers kernel
5: **if** $k = 0$ **then**
6: Compute the QR factorization $\underline{V}_0 = \underline{Q}_0 \underline{R}_0$ using TSQR
7: $\underline{\mathfrak{Q}}_0 = \underline{Q}_k$
8: $\underline{T}_0 = \underline{R}_0 \underline{B}_0 R_0^{-1}$, $\underline{\mathfrak{T}}_0 = T_0$
9: **else**
10: $\underline{\acute{R}}_{k-1,k} = \underline{Q}_{k-1}^* \underline{\acute{V}}_k$
11: $\underline{\acute{V}}_k = \underline{\acute{V}}_k - \underline{Q}_{k-1} \underline{\acute{R}}_{k-1,k}$
12: Compute the QR factorization $\underline{\acute{V}}_k = \underline{\acute{Q}}_k \underline{\acute{R}}_k$ using TSQR
13: $\underline{\mathfrak{Q}}_k = \left[\underline{\mathfrak{Q}}_{k-1}, \underline{\acute{Q}}_k\right]$
14: Compute T_k (Eq. 13) and $\beta_{s(k+1)+1}$ (Eq. 14)
15: $\underline{\mathfrak{T}}_k = \begin{pmatrix} \mathfrak{T}_{k-1} & \beta_{sk+1} e_{s(k-1)} e_1^T \\ \beta_{sk+1} e_1 e_{s(k-1)}^T & T_k \\ 0_{1,s(k-1)} & \beta_{s(k+1)+1} e_s^T \end{pmatrix}$
16: **end if**
17: **end for**

Then we get the following expression for T_k

$$T_k = R_k B_k R_k^{-1} + \tilde{\rho}_k^{-1} b_k z_k e_s^T - \beta_{sk+1} e_1 e_s^T R_{k-1,k} R_k^{-1}, \tag{13}$$

and an expression for $\beta_{s(k+1)+1} = \underline{T}_k(s+1, s)$

$$\beta_{s(k+1)+1} = \frac{\rho_k}{\tilde{\rho}_k} b_k. \tag{14}$$

6 Semi-orthogonal techniques with CA-Lanczos

We have implemented and evaluated two semi-orthogonal techniques in conjuction with CA-Lanczos, *selective orthogonalization* [16] and *periodic orthogonalization* [17, 18]. Both of these methods are commonly used with the standard Lanczos algorithm and the discussion in this section will cast some light on the differences and pitfalls that we might occur when implementing them for CA-Lanczos.

The key idea behind the semi-orthogonal Lanczos schemes is to monitor the loss of orthogonality among the basis vectors and restore orthogonality when needed. The event of restoring mutual orthogonality among the basis vectors is referred to as an orthogonalization break. Several efficient strategies for estimating the loss of orthogonality have been proposed previously (see e.g. [16, 17]). We use two different techniques; in selective orthogonalization we estimate the error bounds on the convergence of each Ritz pair and in periodic orthogonalization we estimate the *level of orthogonality* of the current basis vectors in each step by a simple recurrence update.

6.1 Periodic orthogonalization

Periodic orthogonalization [5] is a simple and straightforward technique for maintaining semi-orthogonality among the Lanczos vectors. Orthogonalization is done by performing full orthogonalization but it is deferred until the orthogonalization error exceeds a given threshold.

The need for orthogonalization is assessed by monitoring the level of orthogonality. At step j this is defined as follows [18]

$$\kappa_j \equiv \max_{1 \le k < j} |q_j^* q_k| \approx \max_{1 \le k < j} |\omega_{j,k}|, \tag{15}$$

where the approximate entries $\omega_{j,k}$ are computed using a simple recurrence presented by Simon [18, 19] (also discussed in [17])

$$\begin{aligned} \omega_{j+1,k} &= \left(\tilde{\omega} + \text{sign}\left(\tilde{\omega}\right) 2\epsilon_M ||A||\right) / \beta_j, \\ \tilde{\omega} &= \beta_{k+1}\omega_{j,k+1} + \left(\alpha_k - \alpha_j\right)\omega_{j,k} + \beta_k \omega_{j,k-1} - \beta_j \omega_{j-1,k}, \\ &\qquad \text{for } 1 \le k < j, \end{aligned} \tag{16}$$

using

$$\begin{aligned} \omega_{j,k} &= \omega_{k,j}, && k = 1, \ldots, j \\ \omega_{k,0} &\equiv 0, && k = 1, \ldots, j \\ \omega_{k,k} &= 1, && k = 1, \ldots, j \\ \omega_{k,k-1} &= q_k^* q_{k-1} = O(\epsilon_M), && k = 2, \ldots, j \end{aligned}$$

The recurrence in (16) requires just a few arithmetic operations in each step and depends only on already computed scalar quantities.

In summary, the algorithm is described by the following steps:

1. Do Lanczos iterations with local orthogonalization. In each iteration (every outer iteration of CA-Lanczos), update $\omega_{j,k}$ and compute the level of orthogonality (15).

2. If the level of orthogonality exceeds $\sqrt{\epsilon_M}$, reorthogonalize the current block of Lanczos vectors against all previous blocks of vectors and update $\omega_{j,k}$ to reflect the explicit orthogonalization. In standard Lanczos we have to reorthogonalize the current Lanczos vector (q_{j+1}) as well as the one before it (q_j) against all previously computed Lanczos vectors, whereas in CA-Lanczos we only have to reorthogonalize the current block and the last vector in the previous block.

3. Repeat step 1 - 2 until convergence.

6.2 Selective orthogonalization

In selective orthogonalization, each basis vector is made orthogonal to the Ritz vectors that have (nearly) converged in the previous iterations. We can estimate the convergence of a Ritz pair by computing an error bound of the corresponding residual norm [16, 17]

$$||Ay_i - \theta_i y_i||_2 = \beta_{j+1} |e_j^T s_i| = \beta_{j+1} |s_{j,i}|. \tag{17}$$

This is inexpensive to compute since for each i it only involves the scalars β_{j+1} and $s_{j,i}$, where $s_{j,i}$ is the bottom element of eigenvector i of the projection matrix (T_j in Lanczos, $\mathfrak{T}_k$ in CA-Lanczos). In the standard Lanczos algorithm, T_j is tridiagonal by definition so we can compute its eigenvectors efficiently. In CA-Lanczos on the other hand, $\mathfrak{T}_k$ is only approximately tridiagonal and stored as a full matrix. This makes the eigenvalue decomposition of $\mathfrak{T}_k$ slightly more expensive to compute than that of T_j. The total cost might be lower though, since we only need to compute the eigenvectors every outer iteration of CA-Lanczos. Furthermore, we might do as well with the eigendecomposition of just the tridiagonal part of $\mathfrak{T}_k$ for computing (17). This performance aspect has not been studied in this work, since performance is not the main subject of this paper.

In summary, the Lanczos algorithm with selective orthogonalization proceeds as follows [16]:

1. Do Lanczos iterations with local orthogonalization. In each iteration (every outer iteration of CA-Lanczos), estimate the convergence of each Ritz pair.

2. If there are newly converged Ritz pairs, compute *all* converged Ritz vectors (i.e. the previously computed Ritz vectors need to be recomputed).

3. In the subsequent Lanczos iterations, every new basis vector is orthogonalized locally and made orthogonal to all converged Ritz vectors.

4. Repeat steps 2-3 until convergence.

6.3 Comparison of the cost of an orthogonalization break

Upon an orthogonalization break with periodic orthogonalization, orthogonality is restored by reorthogonalizing the current block of vectors against all the previous vectors in the Krylov basis. This operation becomes increasingly expensive as the number of Lanczos vectors grows but no additional arithmetic operations are required on a break. On the other hand, when an orthogonalization break occurs with selective orthogonalization, a significant amount of extra arithmetic is introduced by the explicit regeneration of the set of converged Ritz vectors. As the number of converged Ritz vectors increases, the cost of computing the Ritz vectors increase, as does the orthogonalization against the converged set that needs to be done in each step of the Lanczos algorithm.

7 Explicit restart

The main objective of a restarted Lanczos scheme is to reduce the memory requirements and orthogonalization effort when the number of iterations grows large. In the explicitly restarted scheme that we have implemented, based on the discussion by Hernandez *et al.* [7], this is accomplished by repeatedly computing new m-step factorizations (m is referred to as the restart length) and refine the initial vector at each restart such that the iterations converge towards the eigenpairs that we are interested in. The initial vector for each restart is generated based on the information available from the most recent factorization, for example the Ritz vector corresponding to the largest or the smallest non-converged Ritz value [7]. Unlike the implementation proposed by Hernandez *et al.* [7] we

Algorithm 4 EXPLICTLY RESTARTED CA-LANCZOS

Let m: restart length (multiple of s for CA-Lanczos)
$k = 0$
while $k <$ number of wanted eigenvalues **do**
  Take m Lanczos steps (Algorithm 1 or 3) with initial vector v_{k+1}, $AV_m = V_m T_m$
  Compute eigenpairs of T_m, $T_m y_i = y_i \theta_i$
  Compute residual norm estimates, $\tau_i = \beta_{m+1} |e_m^* y_i|$
  for each converged eigenpair θ_i, y_i **do**
    $k = k + 1$
    $E_k = \theta_i$
    $V_k = V_m y_i$
  end for
end while

keep the restart length fixed between restarts. Otherwise, in particular for large values of s, there may be significant variances in the number of iterations for the inner Lanczos kernel to do which would lead to slow or unpredictible convergence properties. In our implementation, we need to keep a total of $m + n_w$ Lanczos vectors in memory, where n_w is the number of wanted eigenvalues specified by the user.

At each restart, the approximate residual norm of every Ritz pair is computed and those Ritz pairs for which the residual norm fulfills the tolerance are *locked*. In order to lock a Ritz pair, we compute the Ritz vector and save it together with its corresponding Ritz value. After k Ritz pairs have been locked, we denote the full vector basis

$$V_{k+m} = [\ V_k \mid V_m\ ] = \left[\ V_{1:k}^{(l)} \mid V_{k+1:k+m}^{(a)}\ \right] \tag{18}$$

where the superscript (l) denotes the set of locked vectors and (a) the set of active vectors that have not yet converged [7]. In all subsequent restarts, the computed Lanczos vectors are kept orthogonal against the k converged ones (the converged vectors are deflated) and no further modifications are done to the converged Ritz pairs. In every iteration of the restart loop, the inner Lanczos routine operates on the $m - k$ active Ritz pairs

The choice of restart length is highly problem dependent and has a significant impact on the overall performance. If the restart length is too short, iterations will converge very slowly (if at all) since we loose a little bit of information every time we throw vectors away. If it is too large, orthogonality may be lost or lead to severe performance degradation if the number of vectors to orthogonalize against is very large.

8 Numerical experiments

In this section, we assess the numerical behaviour of the CA-Lanczos algorithm with respect to convergence, orthogonality and accuracy of the computed eigenvalues. We compare the efficiency of the orthogonalization techniques that were discussed in Section 6 and evaluate the accuracy and stability of CA-Lanczos with explicit restart, described in Section 7.

Throughout the numerical experiments we have tracked the convergence of the Ritz pairs and the orthogonalization error. The convergence of a Ritz pair is described by its

residual norm, which (for Ritz pair i) is computed as $||Ay_i - \theta_i y_i||_2 / ||\theta_i y_i||_2$, where θ_i is the ith eigenvalue and y_i the corresponding Ritz vector of A. The orthogonalization error is defined as the maximum of $||I - Q_m^* Q_m||_F$. For the semi-orthogonal schemes we have also tracked the number of times iterations are interrupted in order to reorthogonalize the basis vectors, an event referred to as an orthogonalization break.

We have selected a few matrices with different properties for evaluation of our implementation, ranging from simple diagonal matrices to complex matrices from various applications. All matrices are symmetric positive definite and real but they differ in structure, condition number and eigenvalue distribution. For each matrix we have performed two experiments. In the first experiment we analyze the convergence behaviour of the CA-Lanczos algorithm, for different lengths of the s-step basis and for different orthogonalization strategies. We let the iterations go way beyond the point where the largest Ritz pair has converged, in order to see how well orthogonality is preserved. In the second experiment we evaluate the numerical performance of the explicitly restarted scheme. We iterate until the 10 largest eigenvalues have converged to a tolerance on the relative residual norm of 10^{-8}, using a restart length of 60 iterations. In all experiments we have used a vector of all ones as a starting vector.

8.1 Diagonal matrices

As a basic test case for verification and analysis, we use diagonal matrices with equally spaced eigenvalues. By choosing the range over which we distribute the eigenvalues, we can vary the condition numbers of the matrices. These simple matrices are also a good starting point since we can easily verify the correctness of the computed eigenvalues.

Convergence and orthogonality Convergence- and orthogonality results for a diagonal matrix of size 500×500 with condition number 100 are shown in Figures 3 and 4 for the monomial basis and the Newton basis respectively. The relative errors in the computed eigenvalues are given in Table 1 and in Table 2 we list the number of orthogonalization breaks using selective and periodic orthogonalization for each basis. Comparing the results for the two bases, it is immediately apparent that the Newton basis manages much better than the monomial basis does in this case. For the monomial basis, the residual norm and the orthogonalization error grows exponentially with s. For large values of s, the error in the largest eigenvalue grows out of control except with full orthogonalization where we still get an error of $\sim 10^{-7}$. With the Newton basis on the other hand, orthogonality is well preserved with full orthogonalization as well as periodic orthogonalization. With selective orthogonalization, orthogonality among the Lanczos vectors slowly diminishes and for $s \geq 8$ the final error in the largest eigenvalue is unacceptable. Local orthogonalization quickly loses orthogonality, but still maintains the accuracy of the converged eigenvalues.

Restart In Figure 5 we present convergence- and orthogonality results for the explicitly restarted scheme, computing the 10 largest eigenvalues of a diagonal matrix with condition number $1.0 \cdot 10^4$. The dimension of this matrix is 5000. As a reference, we also plot the specified tolerance at $1.0 \cdot 10^{-8}$. Clearly, the monomial basis suffers from bad scaling since the orthogonalization error is significantly larger than for the corresponding

Table 1: Error in the largest computed eigenvalue of a 500×500 diagonal matrix with condition number 100 after 480 iterations.

	Lanczos	$s = 4$	$s = 8$	$s = 12$	$s = 16$
Monomial basis					
Local	1.7e-15	1.7e-14	1.3e-11	3.0e-09	1.2e-04
Full	1.3e-15	5.0e-15	5.6e-13	7.4e-10	2.2e-07
Selective	2.1e-15	3.1e-15	4.0e-14	4.0e-10	2.1e-02
Periodic	3.4e-15	5.1e-15	3.6e-14	1.5e-04	5.6e-04
Newton basis					
Local	8.5e-16	7.1e-16	8.8e-15	3.2e-14	2.8e-16
Full	2.1e-15	2.6e-15	8.7e-15	7.5e-15	1.8e-15
Selective	1.4e-16	4.7e-15	3.9e-02	2.4e-02	1.2e-02
Periodic	2.4e-15	2.3e-15	2.1e-15	4.3e-16	5.4e-15

Table 2: Total number of orthogonalization breaks for a diagonal matrix of dimension 500 with condition number 100 after 480 iterations.

	Selective orth.		**Periodic orth.**	
	Monomial	Newton	Monomial	Newton
Lanczos	185	185	21	21
$s = 4$	77	77	19	19
$s = 8$	41	42	15	17
$s = 12$	18	29	5	16
$s = 16$	19	23	3	15

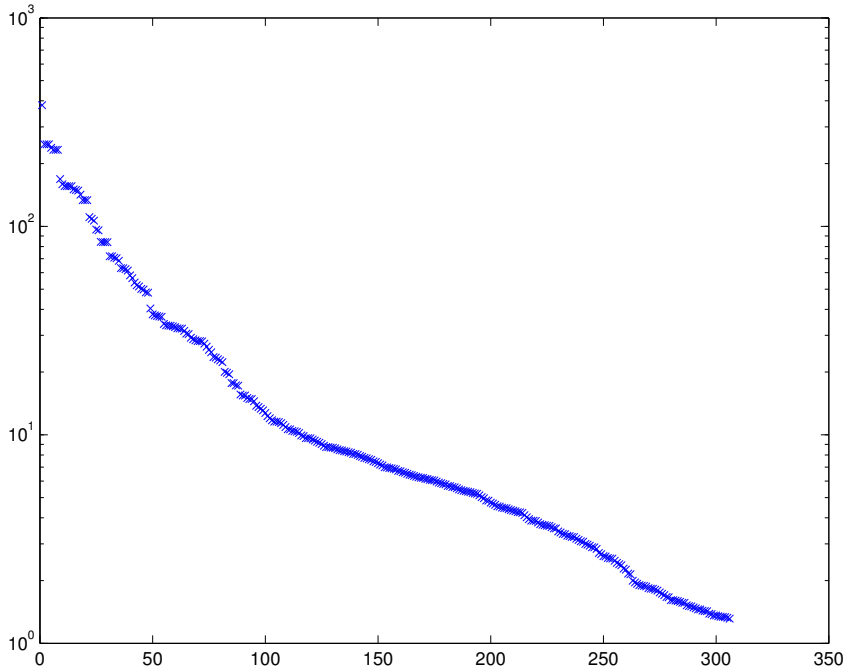


Figure 1: Spectrum of mesh2e1.

error for the Newton basis, in particular for $s = 8$. For $s = 12$, the monomial basis fails to converge in 200 restarts, whereas Newton basis converges in 81 restarts (Fig. 5(d)).

8.2 Matrices from applications

We use two different matrices from the University of Florida Sparse Matrix Collection [4], mesh2e1 and nos5. Both of these matrices are of moderate size such that we can compute the eigenvalues directly for reference.

8.2.1 mesh2e1

This matrix represents a structural problem developed by NASA. It has a dimension of 306 with 2018 nonzeros and a condition number of ~ 400. The spectrum of mesh2e1 is given in Figure 1. The largest eigenvalue is well separated from the others, but then there are several clusters and multiples of eigenvalues. We expect this to lead to fast convergence to the largest eigenvalue, but difficulties to find the right multiplicities of all eigenvalues.

Convergence and orthogonality For mesh2e1 the convergence results are shown in Figures 6 and 7. Table 3 lists the final errors in the largest eigenvalue for each basis and value of s. The Newton basis with full and periodic orthogonalization yields a relative residual norm that is kept at machine accuracy level for all values of s except for $s = 16$, but with local or selective orthogonalization, orthogonality is lost to the extent that the converged result is destroyed. Surprisingly, the monomial basis is more stable than with the Newton basis, although the monomial basis does not converge down to the same level of accuracy as Newton base does when it converges. This behaviour is probably

Table 3: Error in the largest computed eigenvalue of matrix mesh2e1 after 144 iterations.

	Lanczos	$s = 4$	$s = 8$	$s = 12$	$s = 16$
Monomial basis					
Local	1.5e-16	3.5e-10	1.0e-01	8.6e-05	5.7e-10
Full	7.5e-16	1.0e-15	2.0e-14	2.7e-14	8.1e-13
Selective	4.4e-16	1.0e-15	2.2-14	3.2e-14	7.5e-12
Periodic	1.5e-16	9.0e-16	2.2e-14	2.7e-14	2.2e-12
Newton basis					
Local	1.3e-15	1.1e-04	8.9e-03	1.9e-08	1.5e-03
Full	6.0e-16	3.4e-15	4.5e-16	3.1e-15	7.1e-06
Selective	1.2e-15	1.9e-02	2.3e-04	6.3e-05	4.2e-06
Periodic	3.0e-16	1.0e-15	2.1e-15	1.2e-15	1.2e-06

Table 4: Total number of orthogonalization breaks for the mesh2e1 matrix after 144 iterations.

	Selective orth.		**Periodic orth.**	
	Monomial	Newton	Monomial	Newton
Lanczos	46	46	29	29
$s = 4$	19	24	26	26
$s = 8$	14	15	17	17
$s = 12$	11	11	11	11
$s = 16$	8	8	9	9

due bad estimates of the Ritz values in the Newton matrix powers kernel. In Table 4 we list the count of orthogonalization events for selective and periodic orthogonalization. Interestingly enough, the number of orthogonalizations shrinks as s increases but as is clear from Figure 7, for $s = 12$ and larger the orthogonalization is insufficient to control the error.

Restart In this example, the restarted scheme converged towards the 10 largest Ritz pairs already after the first restart (with a restart length of 60 Lanczos iterations). Thus, there are no convergence histories to present for this experiment.

8.2.2 nos5

The matrix nos5 is a finite element matrix approximating the shape of a building. Its dimension is 468 and it has 5172 nonzero elements. The condition number is $\sim 1.1 \cdot 10^4$. Figure 2 shows the spectrum of nos5. The large eigenvalues are very close together, with no well separated outliers. However, in the small end of the spectrum, the eigenvalues are much more distinguished. We expect this to yield slower and smoother convergence towards the larger eigenvalues than what was the case for mesh2e1 in the previous example.

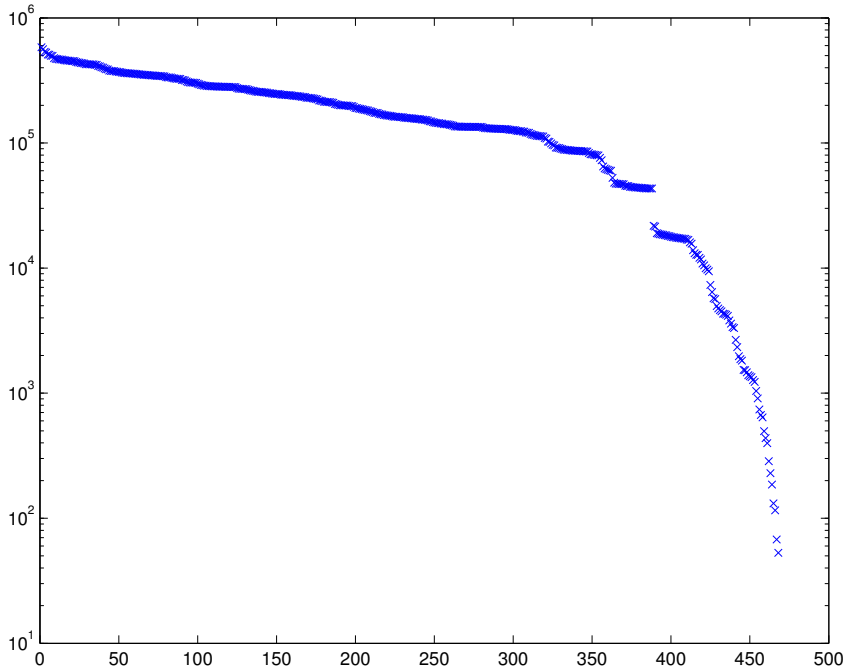


Figure 2: Spectrum of nos5.

Convergence and orthogonality The convergence and orthogonality results of the nos5 matrix are presented in Figures 8 and 9 for the monomial and Newton basis respectively. The orthogonality behaviour is about the same for both bases, but the monomial basis scales much worse with s than the Newton basis does. However, in all cases the converged eigenpairs remain stable, so the choice of basis only affects the accuracy of the result. Tables 5 shows the numerical error in the largest eigenvalue after 144 iterations, and 6 shows the number of orthogonalization breaks. Here too, the number of orthogonalization events decreases as s increases, but in this case we do not have any problems with the orthogonalization intervals.

Restart Figure 10 shows the results for the explicit restart scheme, computing the 10 largest Ritz values and the corresponding Ritz vectors of nos5. We see that the Newton basis converges even for larger values of s, whereas the monomial basis fails to converge for $s = 8$. However, the residual norm estimate indicates that all 10 Ritz pairs have converged so the restarted scheme terminates after 5 iterations with 4 erroneous Ritz estimates.

9 Conclusions and future work

The numerical behaviour of the Lanczos algorithm is known to be very complex and problem-dependent, and has been an active area of research for decades. With this paper we have investigated whether CA-Lanczos can be used as a substitute for the Lanczos algorithm without sacrificing numerical accuracy. In general, we conclude that CA-Lanczos can indeed replace the Lanczos algorithm, as long as care is taken to make appropriate choices for the length and basis of the matrix powers kernel. In our comparison of bases

Table 5: Error in the largest computed eigenvalue of matrix nos5 after 144 iterations.

	Lanczos	$s = 4$	$s = 8$	$s = 12$	$s = 16$
Monomial basis					
Local	1.0e-15	8.4e-15	1.2e-12	2.0e-05	2.4e-08
Full	6.0e-16	1.2e-14	1.4e-12	2.1e-10	2.4e-08
Selective	1.4e-15	6.6e-15	1.4e-12	2.1e-10	2.4e-08
Periodic	8.0e-16	1.5e-14	1.4e-12	2.1e-10	2.4e-08
Newton basis					
Local	4.0e-16	1.4e-14	1.5e-14	1.2e-13	1.3e-10
Full	1.0e-15	6.2e-15	5.2e-15	6.8e-15	4.0e-15
Selective	2.2e-15	1.5e-14	4.8e-15	8.6e-15	1.3e-14
Periodic	6.0e-16	1.1e-14	7.6e-15	4.6e-15	1.8e-14

Table 6: Total number of orthogonalization breaks for the nos5 matrix after 144 iterations.

	Selective orth.		**Periodic orth.**	
	Monomial	Newton	Monomial	Newton
Lanczos	15	15	22	22
$s = 4$	15	15	15	15
$s = 8$	7	11	15	15
$s = 12$	6	9	10	10
$s = 16$	7	5	8	8

for the matrix powers kernel, we see that the overall numerical performance of the Newton basis is generally better than the monomial basis, which is expected [9, 2].

Furthermore, we see that the choice of orthogonalization strategy is important for accuracy as well as convergence, in particular for larger values of s. Our evaluation of the semi-orthogonal schemes shows that periodic orthogonalization manages to preserve orthogonality much better than selective orthogonalization, without exceptions. The semi-orthogonal schemes are prone to loose accuracy for large values of s, since when the need for reorthogonalization arises, there might be many iterations more to go until the next possibility to break.

The explicitly restarted scheme we have demonstrated here fails to be stable for large values of s. If and when this happens is dependent on the problem and to some extent on the matrix powers basis. A more stable restarted scheme, such as the implicitly restarted Lanczos method [1] or the thick restart Lanczos method [23] might be better off to cope with these difficulties. We leave for future work to investigate the implementation of these schemes.

References

[1] D. Calvetti, L. Reichel, and D. C. Sorensen. An implicitly restarted Lanczos method for large symmetric eigenvalue problems. *Electronic Transactions on Numerical Analysis*, 2:1–21, 1994.

[2] E. Carson, N. Knight, and J. Demmel. Avoiding communication in two-sided Krylov subspace methods. Technical report, EECS Department, University of California, Berkeley, 2011.

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

[4] T. A. Davis and Y. Hu. The University of Florida sparse matrix collection.

[5] J. Grcar. *Analyses of the Lanczos algorithm and of the approximation problem in Richardson's method*. PhD thesis, University of Illinois at Urbana Champaign, 1981.

[6] M. Gustafsson, K. Kormann, and S. Holmgren. Communication-efficient algorithms for numerical quantum dynamics. In *Applied Parallel Computing. State of the Art in Scientific Computing 10th International Workshop, PARA 2010, Revised Selected Papers*, volume 7133-7134 of *Lecture Notes in Computer Science*. Springer Berlin / Heidelberg, 2012.

[7] V. Hernandez, J. Roman, and A. Tomas. Evaluation of several variants of explicitly restarted Lanczos eigensolvers and their parallel implementations. In M. Daydé, J. Palma, Á. Coutinho, E. Pacitti, and J. Lopes, editors, *High Performance Computing for Computational Science - VECPAR 2006*, volume 4395 of *LNCS*, pages 403–416. Springer Berlin / Heidelberg, 2007.

[8] M. Hoemmen. A communication-avoiding, hybrid-parallel, rank-revealing orthogonalization method. In *Parallel Distributed Processing Symposium (IPDPS), 2011 IEEE International*, pages 966 –977, 2011.

[9] M. F. Hoemmen. *Communication-Avoiding Krylov Subspace Methods*. PhD thesis, EECS Department, University of California, Berkeley, April 2010.

[10] W. Karush. An iterative method for finding characteristic vectors of a symmetric matrix. *Pacific Journal of Mathematics*, 1(2):233–248, 1951.

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

[12] C. Lanczos. 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, 1950.

[13] M. Mohiyuddin, M. Hoemmen, J. Demmel, and K. Yelick. Minimizing communication in sparse matrix solvers. In *Proceedings of SC09*, November 2009.

[14] C. C. Paige. Computational variants of the Lanczos method for the eigenproblem. *IMA Journal of Applied Mathematics*, 10(3):373–381, 1972.

[15] C. C. Paige. Error analysis of the Lanczos algorithm for tridiagonalizing a symmetric matrix. *IMA Journal of Applied Mathematics*, 18(3):341–349, 1976.

[16] B. N. Parlett and D. S. Scott. The Lanczos algorithm with selective orthogonalization. *Mathematics of Computation*, 33(145):217–238, 1979.

[17] A. Ruhe. Lanczos method. In Z. Bai, J. Demmel, J. Dongarra, A. Ruhe, and H. van der Vorst, editors, *Templates for the Solution of Algebraic Eigenvalue Problems: A Practical Guide*. SIAM, Philadelphia, 2000.

[18] H. D. Simon. Analysis of the symmetric Lanczos algorithm with reorthogonalization methods. *Linear Algebra and its Applications*, 61:101–131, 1984.

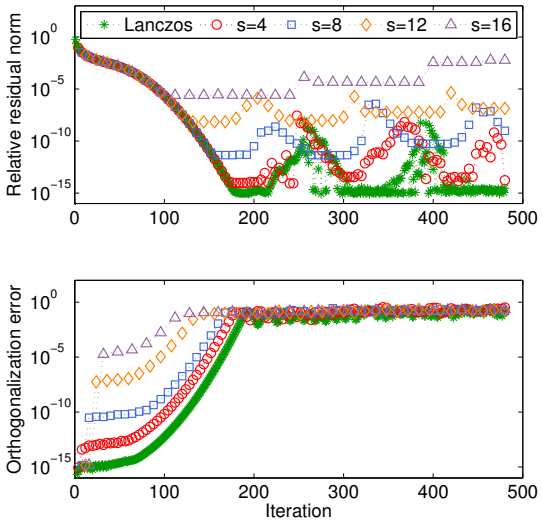
[19] H. D. Simon. The Lanczos algorithm with partial reorthogonalization. *Mathematics of Computation*, 42(165):115–142, 1984.

[20] G. W. Stewart. Block Gram-Schmidt orthogonalization. *SIAM Journal on Scientific Computing*, 31(1):761–775, 2008.

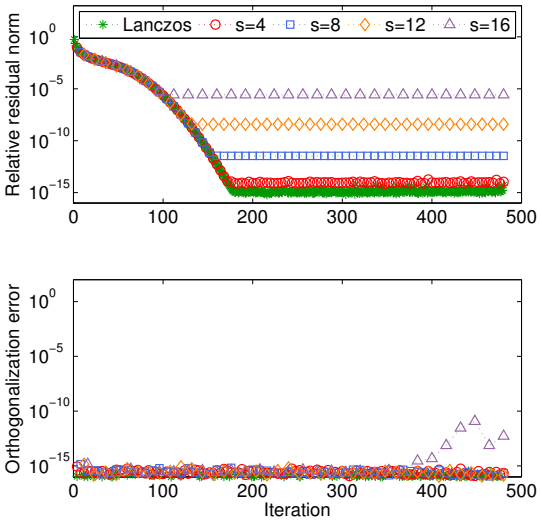
[21] M. Szularz, J. Weston, and M. Clint. Explicitly restarted Lanczos algorithms in an MPP environment. *Parallel Computing*, 25(5):613–631, 1999.

[22] H. A. van der Vorst. *Iterative Krylov Methods for Large Linear Systems*. Cambridge University Press, 2003.

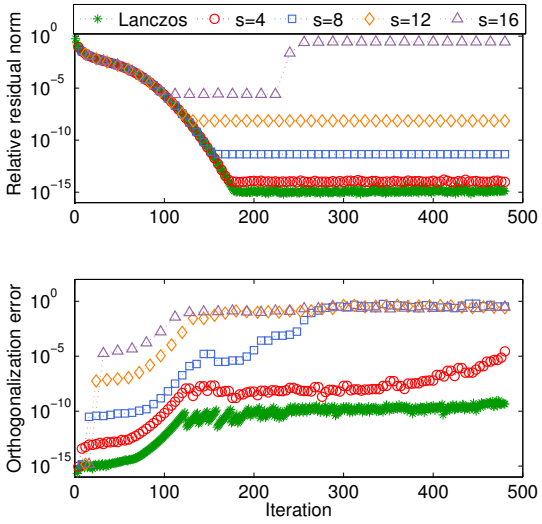
[23] K. Wu and S. Horst. Thick-restart Lanczos method for large symmetric eigenvalue problems. *SIAM Journal on Matrix Analysis and Applications*, 22(2):602–616, 2000.



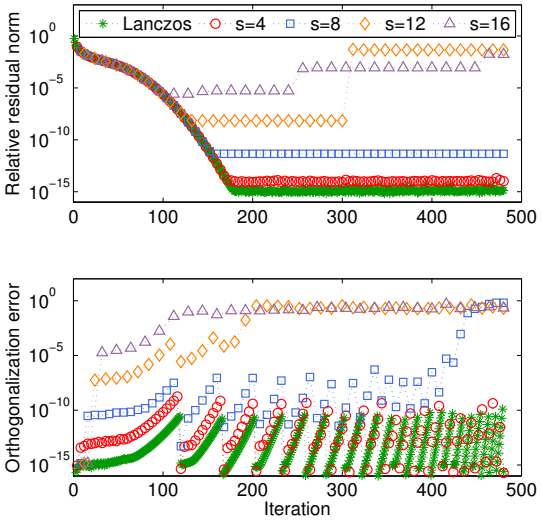
(a) Local orthogonalization



(b) Full orthogonalization

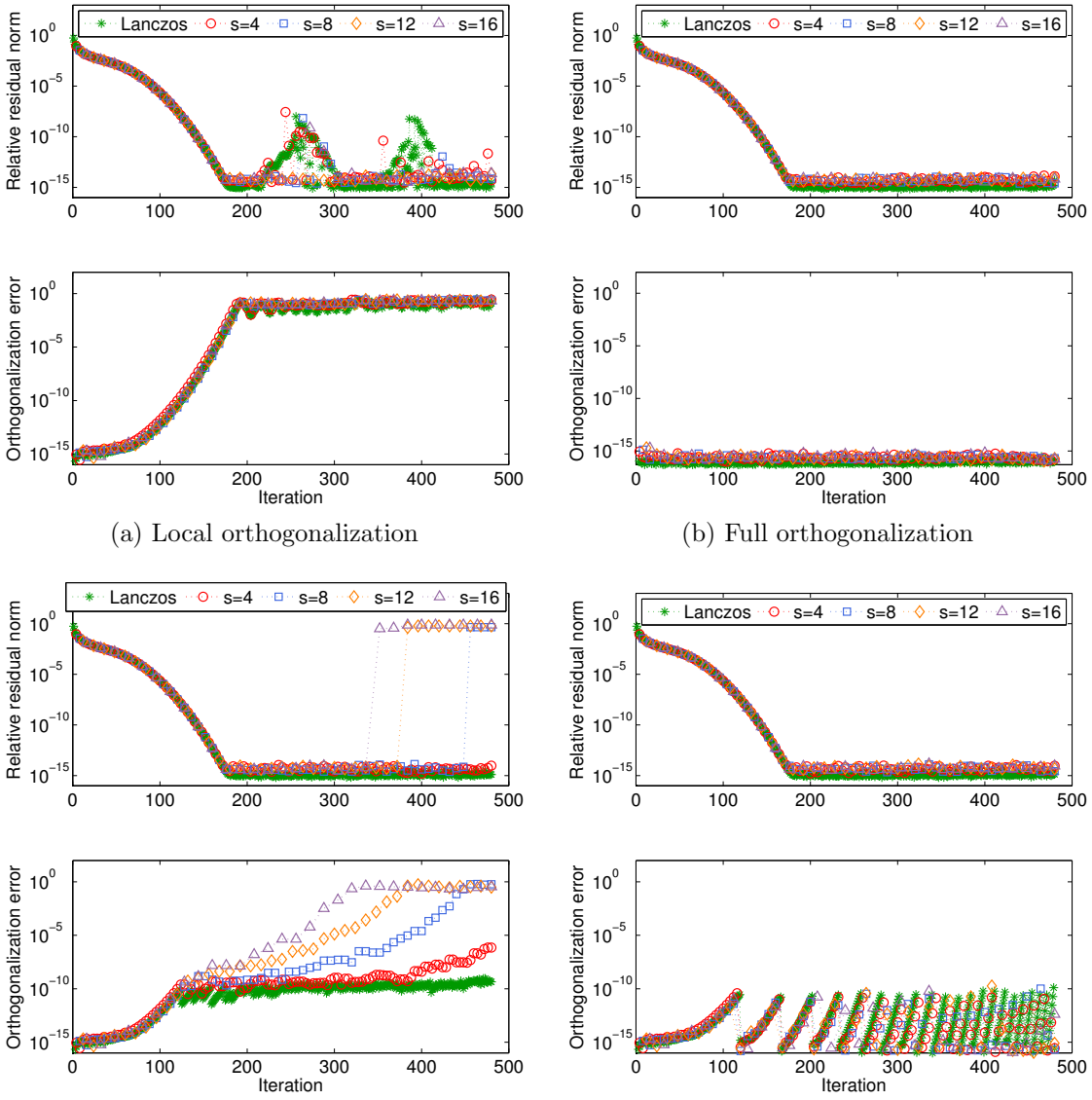


(c) Selective orthogonalization



(d) Periodic orthogonalization

Figure 3: Convergence- and orthogonality results for a 500×500 diagonal matrix with condition number 100, running for 480 iterations. The Monomial basis was used in the matrix powers kernel.



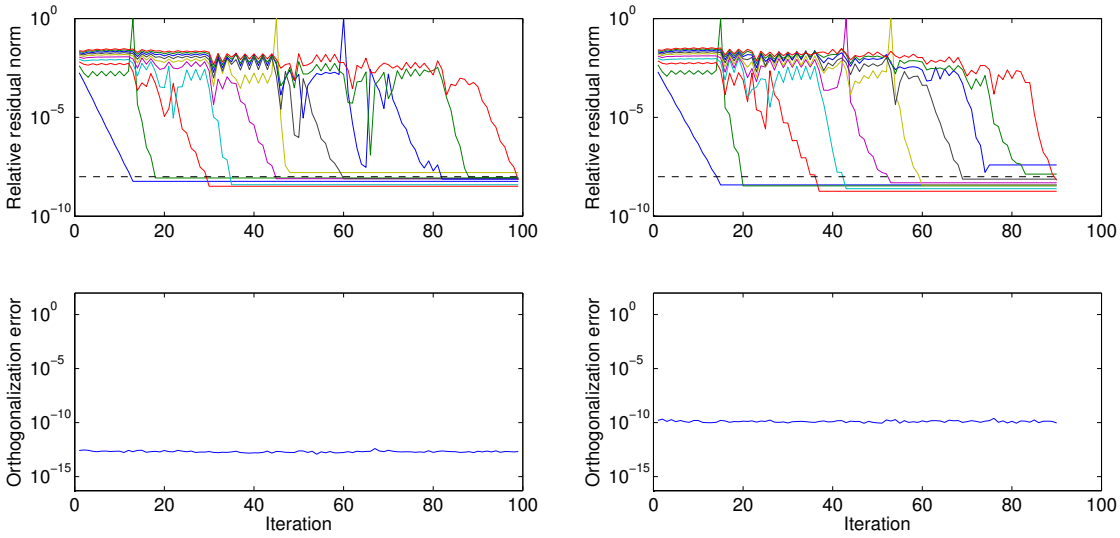
(a) Local orthogonalization

(b) Full orthogonalization

(c) Selective orthogonalization

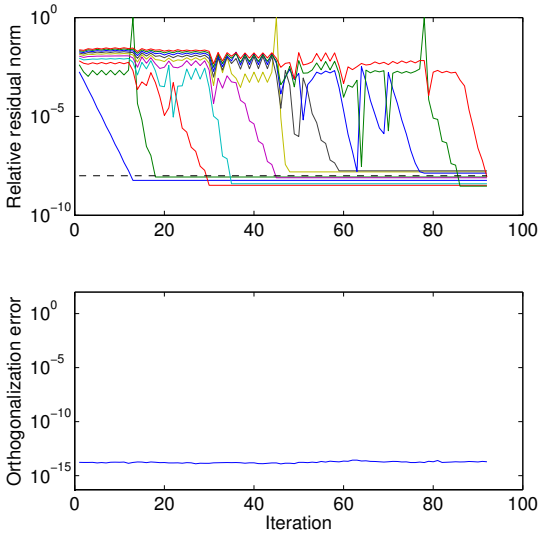
(d) Periodic orthogonalization

Figure 4: Convergence- and orthogonality results for a 500×500 diagonal matrix with condition number 100, running for 480 iterations. The Newton basis was used in the matrix powers kernel.

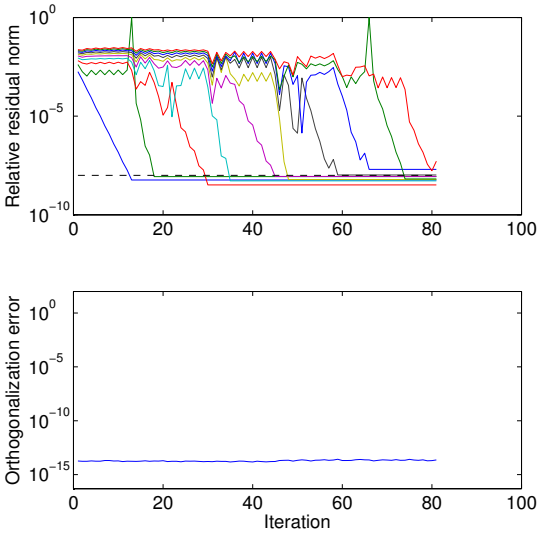


(a) Monomial basis, $s = 4$

(b) Monomial basis, $s = 8$

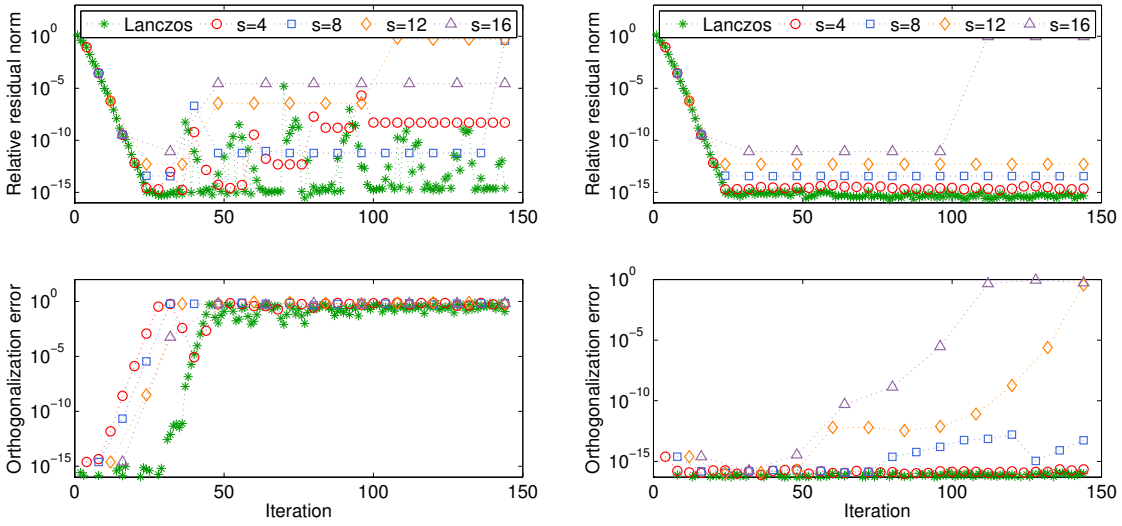


(c) Newton basis, $s = 4$



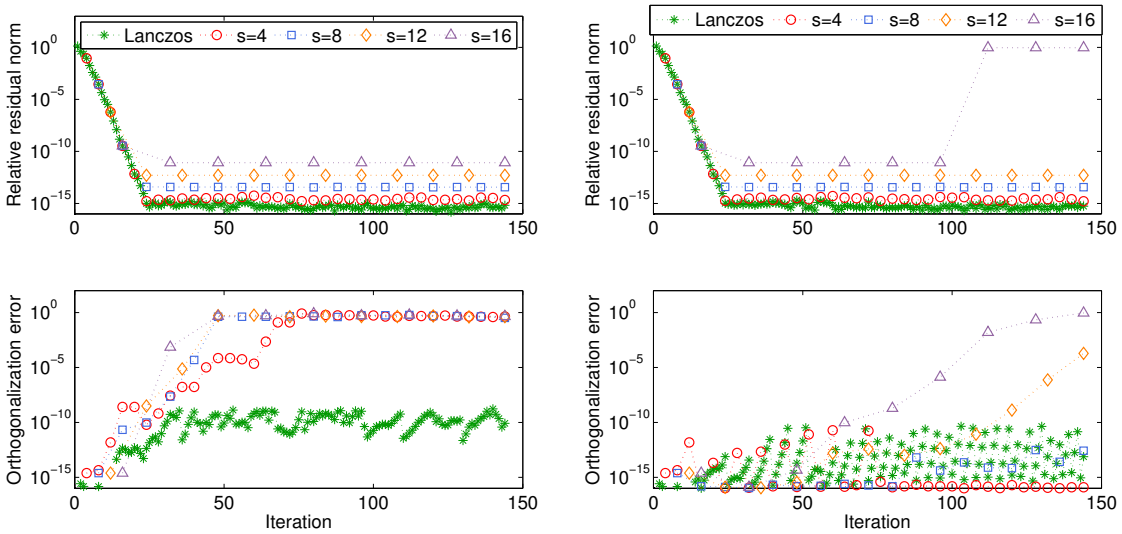
(d) Newton basis, $s = 12$

Figure 5: Convergence- and orthogonality results for a 5000×5000 diagonal matrix with condition number $1.0 \cdot 10^4$, computing the 10 largest eigenvalues. Full orthogonalization has been used in all runs. The dashed line indicates the tolerance of the relative residual norm at $1.0 \cdot 10^{-8}$.



(a) Local orthogonalization

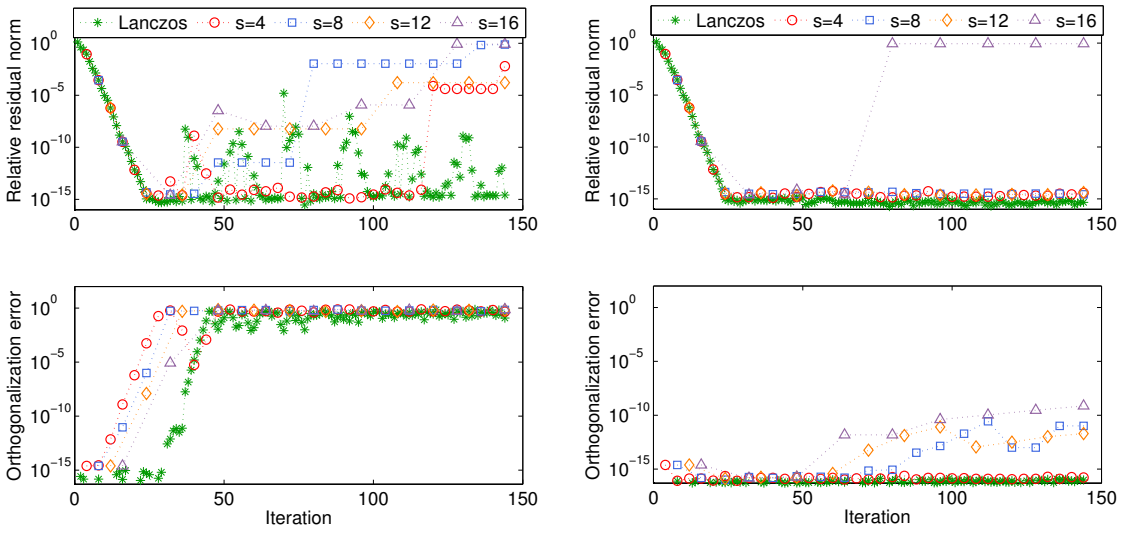
(b) Full orthogonalization



(c) Selective orthogonalization

(d) Periodic orthogonalization

Figure 6: Convergence- and orthogonality results for mesh2e1, running for 144 iterations. The Monomial basis was used in the matrix powers kernel.



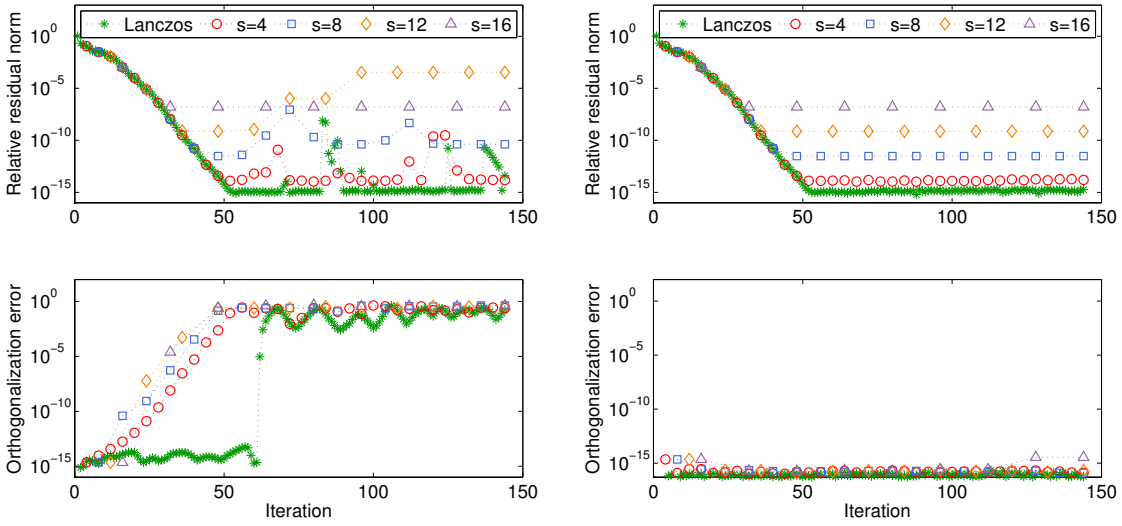
(a) Local orthogonalization

(b) Full orthogonalization

(c) Selective orthogonalization

(d) Periodic orthogonalization

Figure 7: Convergence- and orthogonality results for mesh2e1, running for 144 iterations. The Newton basis was used in the matrix powers kernel.



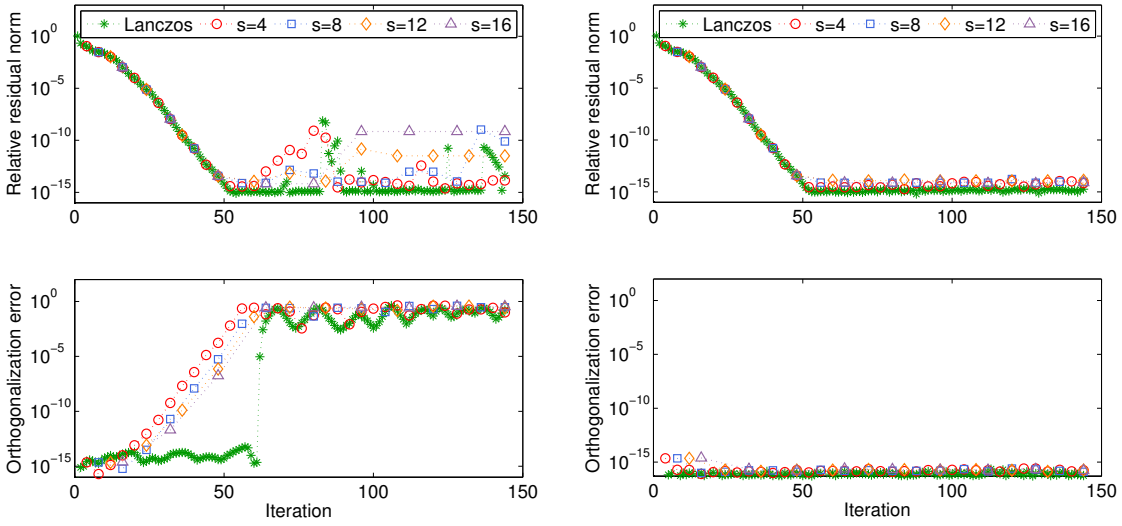
(a) Local orthogonalization

(b) Full orthogonalization

(c) Selective orthogonalization

(d) Periodic orthogonalization

Figure 8: Convergence- and orthogonality results for nos5, running for 144 iterations. The Monomial basis was used in the matrix powers kernel.



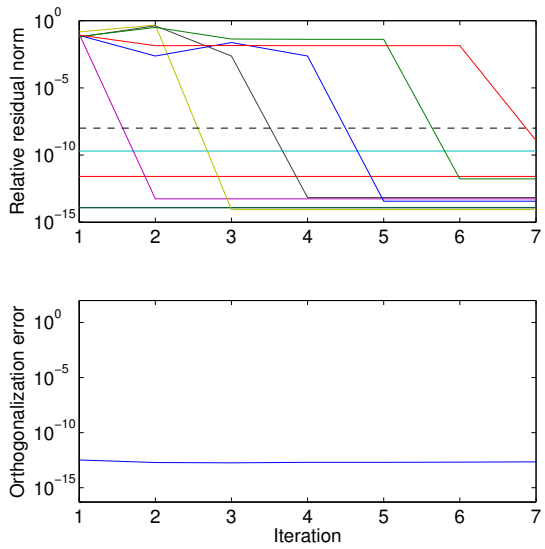
(a) Local orthogonalization

(b) Full orthogonalization

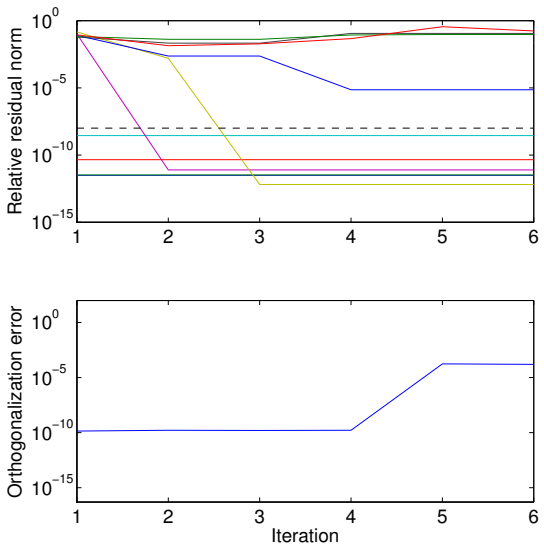
(c) Selective orthogonalization

(d) Periodic orthogonalization

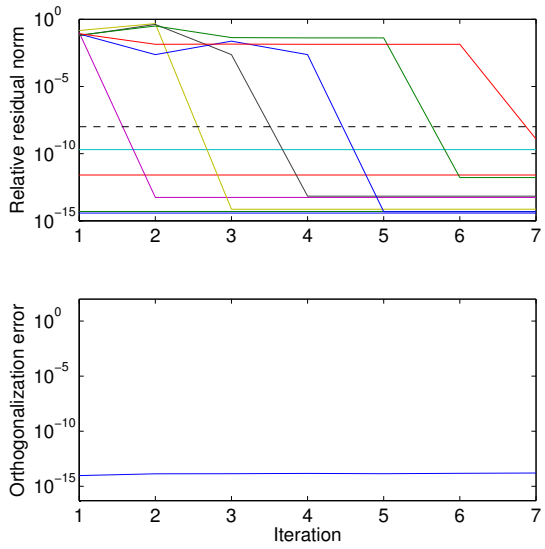
Figure 9: Convergence- and orthogonality results for nos5, running for 144 iterations. The Newton basis was used in the matrix powers kernel.



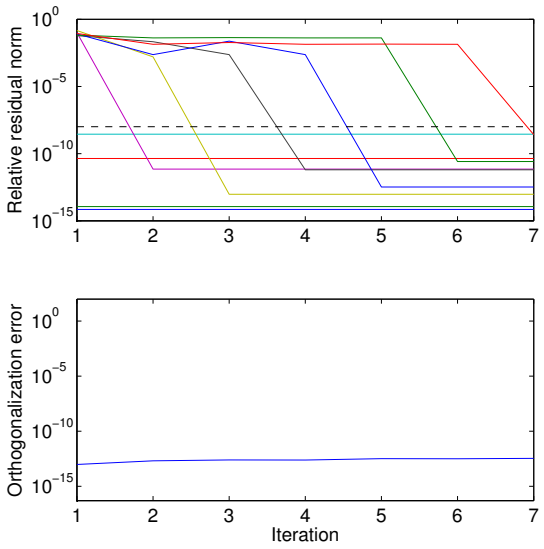
(a) Monomial basis, $s = 4$



(b) Monomial basis, $s = 8$



(c) Newton basis, $s = 4$



(d) Newton basis, $s = 8$

Figure 10: Convergence- and orthogonality results for the nos5 matrix, computing the 10 largest eigenvalues. Full orthogonalization has been used in all runs. The dashed line indicates the tolerance of the relative residual norm at $1.0 \cdot 10^{-8}$.