Block Gram-Schmidt algorithms and their stability properties

Erin Carson[a,1,2,*], Kathryn Lund[1], Miroslav Rozložník[b,3], Stephen Thomas[c,2,4]

[a]*Faculty of Mathematics and Physics, Charles University, Prague, Czech Republic*
[b]*Institute of Mathematics, Czech Academy of Sciences, Prague, Czech Republic*
[c]*National Renewable Energy Laboratory, Boulder, Colorado, USA*

Abstract

Block Gram-Schmidt algorithms serve as essential kernels in many scientific computing applications, but for many commonly used variants, a rigorous treatment of their stability properties remains open. This work provides a comprehensive categorization of block Gram-Schmidt algorithms, particularly those used in Krylov subspace methods to build orthonormal bases one block vector at a time. Known stability results are assembled, and new results are summarized or conjectured for important communication-reducing variants. Additionally, new block versions of low-synchronization variants are derived, and their efficacy and stability are demonstrated for a wide range of challenging examples. Numerical examples are computed with a versatile Matlab package hosted at `https://github.com/katlund/BlockStab`, and scripts for reproducing all results in the paper are provided. Block Gram-Schmidt implementations in popular software packages are discussed, along with a number of open problems. An appendix containing all algorithms type-set in a uniform fashion is provided.

Keywords: Gram-Schmidt, block Krylov subspace methods, stability, loss of orthogonality
2000 MSC: 15-02, 15A23, 65-02, 65F05, 65F10, 65F25

*Corresponding author
Email addresses: `carson@karlin.mff.cuni.cz` (Erin Carson), `kathryn.d.lund@gmail.com` (Kathryn Lund), `miro@math.cas.cz` (Miroslav Rozložník), `stephethomas@gmail.com` (Stephen Thomas)

[1]Supported by Charles University PRIMUS project no. PRIMUS/19/SCI/11 and Charles University Research program no. UNCE/SCI/023.
[2]Supported by the Exascale Computing Project (17-SC-20-SC), a collaborative effort of the U.S. Department of Energy Office of Science and the National Nuclear Security Administration.
[3]Supported by the Academy of Sciences of the Czech Republic (RVO 67985840) and by the Grant Agency of the Czech Republic, Grant No. 20-01074S.
[4]National Renewable Energy Laboratory is operated by Alliance for Sustainable Energy, LLC, for the U.S. Department of Energy (DOE) under Contract No. DE-AC36-08GO28308.

1. Introduction

Given a matrix $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{m \times n}$, $m \gg n$, we treat block Gram-Schmidt (BGS) algorithms that return an "economic" QR factorization

$$\boldsymbol{\mathcal{X}} = \boldsymbol{\mathcal{Q}}\mathcal{R},$$

where $\boldsymbol{\mathcal{Q}} \in \mathbb{R}^{m \times n}$ has orthonormal columns which span the same space as the columns of $\boldsymbol{\mathcal{X}}$ and $\mathcal{R} \in \mathbb{R}^{n \times n}$ is upper triangular with positive diagonal entries.

There are many applications in which block Gram-Schmidt algorithms are more suitable than their single-vector counterparts. Block Krylov subspace methods (KSMs), based on underlying block Arnoldi/Lanczos procedures, are widely used for solving clustered eigenvalue problems (in which the block size is chosen based on the number of eigenvalues in a cluster) as well as linear systems with multiple right-hand sides. For seminal work in this area, see, e.g., Golub and Underwood [1], Stewart [2], and O'Leary [3]. Block approaches can also have an advantage from a performance standpoint, as they replace BLAS-2 (i.e., vector-wise) operations with the more cache-friendly BLAS-3 (i.e., matrix-wise) operations; see, e.g., the work of Baker, Dennis, and Jessup [4].

Block-partitioning strategies are also a key component in new algorithms designed to reduce communication in high-performance computing (HPC). The s-step (or, *communication-avoiding*) Arnoldi/GMRES algorithms are based on a block orthogonalization strategy in which *inter-block orthogonalization* is accomplished via a BGS method; for details see [5, Section 8] and the references therein. There is also recent work by Grigori, Moufawad, and Nataf on "enlarged" KSMs, which are a special case of block KSMs [6]; here the block size is based on a partitioning of the problem and is designed to reduce communication cost.

Implicit in all formulations of BGS is the choice of an *intra-block orthogonalization* scheme, to which we refer throughout as the `IntraOrtho` or "muscle" of a BGS "skeleton." An `IntraOrtho` is any routine that takes a tall and skinny matrix (i.e., block vector) $\boldsymbol{X} \in \mathbb{R}^{m \times s}$ and returns an economic QR factorization $\boldsymbol{Q}R = \boldsymbol{X}$, with $\boldsymbol{Q} \in \mathbb{R}^{m \times s}$ and $R \in \mathbb{R}^{s \times s}$. Popular choices include classical Gram-Schmidt (with reorthogonalization), modified Gram-Schmidt, Householder-based QR (`HouseQR`), Tall-Skinny QR (`TSQR`) [7], and Cholesky-based QR (`CholQR`). Backward stability analyses for all these algorithms are well established and understood, and they are foundational for the analysis of BGS methods; see, e.g., [8, 9, 10, 11, 12, 13], and especially the survey by Leon, Björck, and Gander [14]. To help distinguish between the primary problem $\boldsymbol{\mathcal{X}} = \boldsymbol{\mathcal{Q}}\mathcal{R}$ and the intermediate inputs and outputs of the `IntraOrtho`, we employ various combinations of boldface and calligraphic formatting, summarized in Table 1.

On the other hand, stability analysis for BGS, especially in the form of block Arnoldi, is either non-existent or applies only to a small set of special cases. We focus on the *loss of orthogonality* among the computed basis vectors as the primary measure of stability. To be precise, we let $\bar{\boldsymbol{\mathcal{Q}}} \in \mathbb{R}^{m \times n}$ and $\bar{\mathcal{R}} \in \mathbb{R}^{n \times n}$ denote the computed versions of $\boldsymbol{\mathcal{Q}} \in \mathbb{R}^{m \times n}$, $\mathcal{R} \in \mathbb{R}^{n \times n}$ for a given algorithm,

Table 1: Summary of formatting and notation.

	Size	Description
r	1×1	scalar; entry of vector or matrix
$\boldsymbol{x}$	$m \times 1$	column vector
R	$s \times s$	small square matrix
$\boldsymbol{X}$	$m \times s$	block vector
$\mathcal{R}$	$n \times n$	large square matrix; block entries denoted by R_{ij}
$\boldsymbol{\mathcal{X}}$	$m \times n$	concatenation of block vectors; block columns denoted by $\boldsymbol{X}_j$

where $\boldsymbol{\mathcal{Q}}\mathcal{R} = \boldsymbol{\mathcal{X}}$ is the QR decomposition of $\boldsymbol{\mathcal{X}}$ in exact arithmetic. We define *loss of orthogonality* as the quantity

$$\left\| I_n - \bar{\boldsymbol{\mathcal{Q}}}^T \bar{\boldsymbol{\mathcal{Q}}} \right\|, \tag{1}$$

for some norm $\|\cdot\|$. We take $\|\cdot\|$ to be the Euclidean norm, unless otherwise noted. This quantity often plays an important role in down-stream applications, such as the analysis of (block) GMRES/Arnoldi methods.

Letting ε denote the unit roundoff, we seek bounds on (1) in terms of $\mathcal{O}(\varepsilon)$ and $\kappa(\boldsymbol{\mathcal{X}})$, which denotes the 2-norm condition number of $\boldsymbol{\mathcal{X}}$, defined as the ratio between its largest and smallest singular values for a full-rank matrix and as infinity for the rank-deficient case. We say that an algorithm is *unconditionally stable* if (1) is bounded by $\mathcal{O}(\varepsilon)$ for any matrix $\boldsymbol{\mathcal{X}}$. In contrast, we say an algorithm is *conditionally stable* if the bound on (1) is in terms of $\kappa(\boldsymbol{\mathcal{X}})$ or only holds with some constraint on $\kappa(\boldsymbol{\mathcal{X}})$. Note that this use differs from the notion of "conditional (backward) stability" often used in the literature.

Another quantity that plays an important role in stability studies is the *relative residual*:

$$\frac{\left\| \bar{\boldsymbol{\mathcal{Q}}}\bar{\mathcal{R}} - \boldsymbol{\mathcal{X}} \right\|}{\|\boldsymbol{\mathcal{X}}\|}. \tag{2}$$

We take for granted that all algorithms considered here (with the exception of, perhaps, `CGSS+rpl`, `BCGSS+rpl`, `BMGS-CWY` and `BMGS-ICWY`) produce a residual bounded by $\mathcal{O}(\varepsilon)$.

A quantity that features strongly in the stability analysis of `CGS`-based algorithms is what we call the *relative Cholesky residual*,

$$\frac{\left\| \boldsymbol{\mathcal{X}}^T \boldsymbol{\mathcal{X}} - \bar{\mathcal{R}}^T \bar{\mathcal{R}} \right\|}{\|\boldsymbol{\mathcal{X}}\|^2}, \tag{3}$$

which measures how closely an algorithm approximates the Cholesky decomposition of $\boldsymbol{\mathcal{X}}^T \boldsymbol{\mathcal{X}}$.

Recent work by Barlow [15], as well as important results by Barlow and Smoktunowicz [16], Vanderstraeten [17], and Jalby and Philippe [18], appear to be the only rigorous stability treatments of BGS. Other important sources on BGS include the following highly influential texts, most of which use an algorithm derived from the block modified Gram-Schmidt skeleton, referred to here as `BMGS` and given as Algorithm 8:

- 1980: O'Leary [3] proposes block Conjugate Gradients (BCG) for solving linear systems with multiple right-hand sides. Although BCG is not necessarily derived from `BMGS`, O'Leary makes a recommendation relevant to this conversation, namely, that either a "QR algorithm or a modified Gram-Schmidt algorithm" be used as an `IntraOrtho`.

- 1993: Sadkane [19] uses a block Arnoldi (Algorithm 1 in that text) that is based on `BMGS` to compute the leading eigenpairs of large sparse unsymmetric matrices. The "QR method on CRAY2" is specified as the `IntraOrtho`, presumably a Householder-based QR (`HouseQR`) routine.

- 1996: Simoncini and Gallopoulos [20] use a block Arnoldi algorithm as a part of block GMRES but they do not provide pseudocode or specify what is used as the `IntraOrtho`.

- 2003: Saad [21] provides block Arnoldi routines based on both block Classical (Algorithm 6.22) and block Modified (Algorithm 6.23) Gram-Schmidt, with the `IntraOrtho` specified only as a "QR factorization," again presumably `HouseQR`.

- 2005: Morgan [22] uses Ruhe's variant of block Arnoldi (see, e.g., [21, Algorithm 6.24]) to implement block GMRES and block QMR and recommends full reorthogonalization (line (7) of Block-GMRES-DR) to increase stability, because good eigenvalue approximations are needed for deflation. Ruhe's block Arnoldi variant is not attuned for BLAS-3 operations.

- 2006: Baker, Dennis, and Jessup [4] propose a variant of block GMRES tuned to minimize memory movement. They use no reorthogonalization, and their block Arnoldi (lines 4-11 in Figure 1: B-LGMRES(m, k)) is based on `BMGS` with "QR factorization" specified as the `IntraOrtho` (probably a memory-sensitive version of `HouseQR`).

- 2007: Gutknecht [23] discusses a number of important theoretical properties of block KSMs, particularly the issue of rank deficiency among columns of computed basis vectors. Block Arnoldi (Algorithm 9.1) is based on `BMGS` and again, only a "QR factorization" is specified as the `IntraOrtho`, presumably `HouseQR`.

Our goal with the present work is to unify the literature on BGS algorithms in a common language to clarify what is known and what is unknown in terms of their numerical behavior. Our contributions are as follows:

- We provide a classification of existing variants of BGS algorithms within a unifying framework, based on a "skeleton-muscle" analogy first proposed by Hoemmen [24];

- We assemble existing stability bounds and note commonly used skeleton-muscle combinations for which analysis remains open;
- We derive a new block generalization of a low-synchronization algorithm of Świrydowicz, et al. [25];
- We give new insights into the mechanism by which stability is improved in block low-synchronization variants and also demonstrate the incompatibility between different classes of low-synchronization skeletons and muscles;
- We show how the results of Jalby and Philippe can be adapted to give bounds on the loss of orthogonality for `BMGS` combined with an unconditionally stable `IntraOrtho` (such as `HouseQR` or `TSQR`); and
- Using our versatile, publicly-available Matlab package developed to accompany this work, we perform extensive numerical experiments on various skeleton-muscle combinations for a variety of well-known test problems.

The rest of the paper proceeds as follows. In Section 2 we discuss the history and communication properties of all the BGS skeletons we have identified. Some of them are new, in particular, we derive a new block generalization of one of the low-synchronization algorithms by Świrydowicz, et al. [25]. Sections 3 and 4 treat stability properties in more detail. Section 3 contains heatmaps that allow for a quick comparison of many BGS variants all at once for a fixed matrix, while Section 4 focuses on one skeleton at a time and how its stability is affected across different `IntraOrtho`s and condition numbers. We give an overview of known mixed-precision BGS algorithms in Section 5. In Section 6, we examine which variants of BGS are implemented in well-known software. We conclude the survey in Section 7 and identify open problems and future directions. Appendix A and Appendix B contain pseudocode and Matlab scripts, respectively, for algorithms not included in the main text, and Appendix C contains Matlab command line calls for reproducing the plots in Sections 3 and 4.

2. Block Gram-Schmidt variants and a skeleton-muscle analogy

We assume that the matrix $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{m \times n}$, $m \gg n$ is partitioned into a set of p *block vectors*, each of size $m \times s$ $(n = ps)$, i.e.,

$$\boldsymbol{\mathcal{X}} = [\boldsymbol{X}_1 \,|\, \boldsymbol{X}_2 \,|\, \cdots \,|\, \boldsymbol{X}_p],$$

and that a block Gram-Schmidt (BGS) algorithm returns an "economic" QR factorization $\boldsymbol{\mathcal{X}} = \boldsymbol{\mathcal{Q}}\mathcal{R}$, where $\boldsymbol{\mathcal{Q}} = [\boldsymbol{Q}_1 \,|\, \boldsymbol{Q}_2 \,|\, \cdots \,|\, \boldsymbol{Q}_p] \in \mathbb{R}^{m \times n}$ has the same block-partitioned structure as $\boldsymbol{\mathcal{X}}$, and $\mathcal{R} \in \mathbb{R}^{n \times n}$ can also be thought of as a $p \times p$ matrix with matrix-valued entries of size $s \times s$.[5]

[5]To help remember what each index represents throughout the text, note that usually $m > n > p > s$, which is also an alphabetical ordering.

To simplify complex indexing, we first note that matrices like $\boldsymbol{\mathcal{Q}}$ and $\mathcal{R}$ have implicit tensor structure: $\boldsymbol{\mathcal{Q}}$ can be viewed as a third-order tensor, being a vector of block vectors, and $\mathcal{R}$ as a fourth-order tensor, being a matrix of matrices. None of the algorithms we consider requires further partitioning explicitly, so when we index $\boldsymbol{\mathcal{Q}}$ or $\mathcal{R}$, we are always referring to the corresponding contiguous block component, be it an $m \times s$ block vector or $s \times s$ block entry, respectively. This is best demonstrated by some examples:

$$\boldsymbol{\mathcal{Q}}_{1:j} = [\boldsymbol{Q}_1 \,|\, \cdots \,|\, \boldsymbol{Q}_j] \in \mathbb{R}^{m \times sj}.$$

where $1 : j = \{1, 2, \ldots, j\}$ is an indexing vector. Similarly,

$$\mathcal{R}_{1:j,k} = \begin{bmatrix} R_{1,k} \\ R_{2,k} \\ \vdots \\ R_{j,k} \end{bmatrix} \in \mathbb{R}^{sj \times s}.$$

For all Cholesky-based algorithms, we let $R = \texttt{chol}(A)$ denote an algorithm that takes a symmetric, numerically positive-definite matrix $A \in \mathbb{R}^{s \times s}$ and returns an upper triangular matrix $R \in \mathbb{R}^{s \times s}$.

We focus on scenarios where $\boldsymbol{\mathcal{X}}$ is a set of block vectors that may not all be available at once, as is the case in block Arnoldi or block KSMs, wherein the set of block vectors is built by successively applying an operator A to previously generated basis vectors. As such, we only consider BGS variants that do not require access to all of $\boldsymbol{\mathcal{X}}$ at once; in particular, we do not examine methods like CAQR from [26]. We further require that BGS algorithms work left-to-right– i.e., only information from the first k block vectors of $\boldsymbol{\mathcal{Q}}$ and $\boldsymbol{\mathcal{X}}$ is necessary for generating $\boldsymbol{Q}_{k+1}$– and that they feature BLAS-3 operations. We also note that p (the number of block vectors) may not be known a priori, as is the case in block KSMs, which will continue iterating until the specified convergence criterion is met.

Although we do not examine the performance of BGS methods in detail here, we do touch on their asymptotic communication properties, because they are crucial in a number of high-performance applications. In a simplified setting, the cost of an algorithm can be modeled in terms of the required computation, or number of floating point operations performed, and the required communication. By *communication*, we mean the movement of data, both between levels of the memory hierarchy in sequential implementations and between parallel processors in parallel implementations. It is well established that communication and, in particular, synchronization between parallel processors, is the dominant cost (in terms of both time and energy) in large-scale settings; see, e.g., Bienz et al. [27]. It is therefore of interest to understand the potential trade-offs between the numerical properties of loss of orthogonality and stability in finite precision and the cost of communication in terms of number of messages and number of words moved.

There are a multitude of viable BGS algorithms which can all be succinctly described via the skeleton-muscle analogy from Hoemmen's thesis [24]. We thoroughly wear out this metaphor and attempt to identify all viable skeleton and muscle options that have been considered before in the literature, as well as propose a few new ones.

Table 2: Acronyms for algorithms. For the block version of a Gram-Schmidt method, just add a “B” prefix.

`CGS`	classical Gram-Schmidt
`CGS-P`	`CGS`, Pythagorean variant
`CGS+`	`CGS`, run twice
`CGSI+`	`CGS` with inner reorthogonalization
`CGSI+LS`	`CGSI+`, low-synchronization variant
`CGSS+`	`CGS` with selective reorthogonalization
`CGSS+rpl`	`CGS` with selective reorthogonalization and replacement
`MGS`	modified Gram-Schmidt
`MGS+`	`MGS` run twice
`MGSI+`	`MGS` with inner reorthogonalization
`MGS-SVL`	`MGS` with Schreiber-Van-Loan reformulation
`MGS-LTS`	`MGS` with lower triangular solve
`MGS-ICWY`	`MGS` with inverse compact WY
`MGS-CWY`	`MGS` with compact WY
`HouseQR`	QR via Householder reflections
`GivensQR`	QR via Givens rotations
`TSQR`	Tall-Skinny QR
`CholQR`	Cholesky QR
`mCholQR`	mixed-precision Cholesky QR
`CholQR+`	`CholQR` run twice
`ShCholQR++`	shifted Cholesky QR with two stages of reorthogonalization

The analogy is best understood via the literal meanings of skeleton and muscle in mammals. Without muscles[6], we would be motionless bags of organs and bones. Muscles provide the tension needed to stand, propel forward, grasp objects, and push things. In much the same way, a BGS routine without an `IntraOrtho` specified is like a skeleton without muscles. However, just as we can still learn much about the human body by looking at bones, we can learn much from BGS skeletons without being distracted by the details of a specific `IntraOrtho`. In the following sections, we examine the history, development, and communication properties of two classes of BGS: classical and modified. Unless otherwise noted, single-column versions of block algorithms can be obtained by setting $s = 1$ and replacing $\texttt{IntraOrtho}\,(\cdot)$ with $\|\cdot\|$.

Before proceeding, some comments about notation are warranted: throughout the text, we will describe a skeleton with a specific muscle choice as $\texttt{BGS} \circ \texttt{IntraOrtho}$, where `BGS` is a general skeleton, `IntraOrtho` is a general muscle,

[6]...and tendons, ligaments, fasciae, bursae, etc. See, e.g., `https://en.wikipedia.org/wiki/Human_musculoskeletal_system`

and $\circ$ stands for composition. We also use our own naming system for algorithms, rather than what is proposed in their paper of origin, due to suffixes like "2" being used inconsistently to denote reorthogonalization, BLAS2-featuring, or simply a second version of an algorithm. In general, we use the suffix + to denote a reorthogonalized variant. A summary of acronyms used throughout the paper is provided in Table 2.

2.1. Block classical Gram-Schmidt skeletons

2.1.1. Block Classical Gram-Schmidt (`BCGS`*)*

The `BCGS` algorithm is a straightforward generalization of `CGS` obtained by replacing vectors with block vectors and norms with `IntraOrtho`s; see Algorithm 1. `BCGS` is especially attractive in massively parallel settings because it passes $O(p)$ fewer messages asymptotically than block Modified Gram-Schmidt (`BMGS`). See [24, Table 2.4] and also discussions in [5, 7, 26] for more details regarding the communication properties of `BCGS`.

Algorithm 1 $[\boldsymbol{\mathcal{Q}}, \mathcal{R}] = \texttt{BCGS}(\boldsymbol{\mathcal{X}})$

```
1: Allocate memory for $\boldsymbol{\mathcal{Q}}$ and $\mathcal{R}$
2: $[\boldsymbol{Q}_1, R_{11}] = \texttt{IntraOrtho}(\boldsymbol{X}_1)$
3: for $k = 1, \dots, p-1$ do
4:     $\mathcal{R}_{1:k,k+1} = \boldsymbol{\mathcal{Q}}_{1:k}^T \boldsymbol{X}_{k+1}$
5:     $\boldsymbol{W} = \boldsymbol{X}_{k+1} - \boldsymbol{\mathcal{Q}}_{1:k} \mathcal{R}_{1:k,k+1}$
6:     $[\boldsymbol{Q}_{k+1}, R_{k+1,k+1}] = \texttt{IntraOrtho}(\boldsymbol{W})$
7: end for
8: return $\boldsymbol{\mathcal{Q}} = [\boldsymbol{Q}_1, \dots, \boldsymbol{Q}_p]$, $\mathcal{R} = (R_{jk})$
```

By itself, `BCGS` is not often considered a viable skeleton, especially not prior to its use in communication-avoiding Krylov subspace methods [7]. This is likely due to the fact that `BCGS` inherits many of the bad stability properties of `CGS`. In particular, like `CGS`, the loss of orthogonality for `BCGS` can be worse than quadratic in terms of $\kappa(\boldsymbol{\mathcal{X}})$; see Section 4.1. To overcome this issue for `CGS`, Smoktunowicz et al. [28] introduced a variant of `CGS` in which the diagonal entries of the R-factor are computed via the Pythagorean theorem. The resulting algorithm (`CGS-P`) has loss of orthogonality bounded by $\mathcal{O}(\varepsilon)\,\kappa^2(\boldsymbol{\mathcal{X}})$ as long as $\mathcal{O}(\varepsilon)\,\kappa^2(\boldsymbol{\mathcal{X}}) < 1$ is satisfied. This steep constraint on condition number can be a limitation for many practical applications.

A similar correction is possible for `BCGS`, but a block generalization of the Pythagorean theorem, stated in the following theorem (see also [29]), is needed instead.

Theorem 1. *Let full-rank block vectors $\boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{Z} \in \mathbb{R}^{n \times s}$ be such that $\boldsymbol{X} = \boldsymbol{Y} + \boldsymbol{Z}$ and $\boldsymbol{Y} \perp \boldsymbol{Z}$ in the block sense, i.e., the spaces spanned by the columns of $\boldsymbol{Y}$ and $\boldsymbol{Z}$ are orthogonal to each other. Then*

$$\boldsymbol{X}^T \boldsymbol{X} = \boldsymbol{Y}^T \boldsymbol{Y} + \boldsymbol{Z}^T \boldsymbol{Z}. \tag{4}$$

In particular, if $\boldsymbol{X} = \boldsymbol{Q}R$, $\boldsymbol{Y} = \boldsymbol{U}S$, and $\boldsymbol{Z} = \boldsymbol{V}T$ are economic QR decompositions (i.e., $\boldsymbol{Q}, \boldsymbol{U}, \boldsymbol{V} \in \mathbb{R}^{n\times s}$ are orthonormal, and $R, S, T \in \mathbb{R}^{s\times s}$ are upper triangular), then

$$R^T R = S^T S + T^T T. \tag{5}$$

We call the resulting algorithm BCGSP (i.e., `BCGS` with a block Pythagorean correction). Using Theorem 1, we can derive two `BCGS-P` variants, one based on (4), which we call `BCGS-PIP` (for "Pythagorean inner product"), and one based on (5), which we call `BCGS-PIO` (for "Pythagorean intra-orthogonalization"). These variants are shown in Algorithms 2 and 3, respectively. Stability analysis for both `BCGS-P` variants can be found in [29]; see also Section 4.1. We note that the communication requirements of `BCGS-P` can be made asymptotically comparable to those of `BCGS`, as long as the steps computing inner products or intra-orthogonalizations are coupled appropriately. We note that an algorithm equivalent to `BCGS-PIP` was developed in [30, Figure 2].

Algorithm 2 $[\boldsymbol{\mathcal{Q}}, \mathcal{R}] =$ `BCGS-PIP`$(\boldsymbol{\mathcal{X}})$

```
1: Allocate memory for $\boldsymbol{\mathcal{Q}}$ and $\mathcal{R}$
2: $[\boldsymbol{Q}_1, R_{11}] = \texttt{IntraOrtho}\,(\boldsymbol{X}_1)$
3: for $k = 1, \ldots, p-1$ do
4:     $\begin{bmatrix} \mathcal{R}_{1:k,k+1} \\ \mathcal{Z}_{k+1} \end{bmatrix} = [\boldsymbol{\mathcal{Q}}_{1:k}\ \boldsymbol{X}_{k+1}]^T \boldsymbol{X}_{k+1}$
5:     $R_{k+1,k+1} = \texttt{chol}(\mathcal{Z}_{k+1} - \mathcal{R}_{1:k,k+1}^T \mathcal{R}_{1:k,k+1})$
6:     $\boldsymbol{W} = \boldsymbol{X}_{k+1} - \boldsymbol{\mathcal{Q}}_{1:k} \mathcal{R}_{1:k,k+1}$
7:     $\boldsymbol{Q}_{k+1} = \boldsymbol{W} R_{k+1,k+1}^{-1}$
8: end for
9: return $\boldsymbol{\mathcal{Q}} = [\boldsymbol{Q}_1, \ldots, \boldsymbol{Q}_p]$, $\mathcal{R} = (R_{jk})$
```

Algorithm 3 $[\boldsymbol{\mathcal{Q}}, \mathcal{R}] =$ `BCGS-PIO`$(\boldsymbol{\mathcal{X}})$

```
1: Allocate memory for $\boldsymbol{\mathcal{Q}}$ and $\mathcal{R}$
2: $[\boldsymbol{Q}_1, R_{11}] = \texttt{IntraOrtho}\,(\boldsymbol{X}_1)$
3: for $k = 1, \ldots, p-1$ do
4:     $\mathcal{R}_{1:k,k+1} = \boldsymbol{\mathcal{Q}}_{1:k}^T \boldsymbol{X}_{k+1}$
5:     $\left[\sim, \begin{bmatrix} T_{k+1} & \\ & P_{k+1} \end{bmatrix}\right] = \texttt{IntraOrtho}\left(\begin{bmatrix} \boldsymbol{X}_{k+1} & \\ & \mathcal{R}_{1:k,k+1} \end{bmatrix}\right)$
6:     $R_{k+1,k+1} = \texttt{chol}(T_{k+1}^T T_{k+1} - P_{k+1}^T P_{k+1})$
7:     $\boldsymbol{W} = \boldsymbol{X}_{k+1} - \boldsymbol{\mathcal{Q}}_{1:k} \mathcal{R}_{1:k,k+1}$
8:     $\boldsymbol{Q}_{k+1} = \boldsymbol{W} R_{k+1,k+1}^{-1}$
9: end for
10: return $\boldsymbol{\mathcal{Q}} = [\boldsymbol{Q}_1, \ldots, \boldsymbol{Q}_p]$, $\mathcal{R} = (R_{jk})$
```

2.1.2. `BCGS` with reorthogonalization

It is well known that reorthogonalization can stabilize `CGS`; see, e.g., [31]. We note that there are essentially two reorthogonalization approaches: simply running the algorithm twice (CGS+) and performing each inner loop twice in a row (CGSI+, where the I stands for "inner loop"). While the two approaches differ in floating-point arithmetic, they satisfy the same error bounds and have the same asymptotic communication cost. In particular, they achieve $\mathcal{O}(\varepsilon)$ loss of orthogonality, formulated as the "twice is enough" principle in [32], and double the communication cost of CGS. A variant of CGSI+ that selectively chooses whether or not to run an inner loop twice is called CGSS+ (the extra 'S' for "selective"), which can reduce the total number of floating-point operations, usually at the expense of increased communication due to the norm computations used to determine which vectors to reorthogonalize. Many different selection criteria have been proposed; see, e.g., [33, 34, 14, 35, 17].

Reorthogonalization techniques can be easily generalized to block algorithms. BCGSI+ (Algorithm 4) has received increasing attention in recent years and was analyzed in detail by Barlow and Smoktunowicz in 2013 [16]. We explore some additional stability properties in Section 4.2.

As an aside, note that running an IntraOrtho twice on each vector in BCGS would generally not salvage orthogonality the way BCGS+ and BCGSI+ do. The problem is that even with an incredibly stable muscle, BCGS itself is unstable as a skeleton; see Section 4.1 for more details, in particular, the behavior of BCGS $\circ$ HouseQR.

Algorithm 4 $[\boldsymbol{\mathcal{Q}}, \mathcal{R}] =$ BCGSI+($\boldsymbol{\mathcal{X}}$)

```
1: Allocate memory for $\boldsymbol{\mathcal{Q}}$ and $\mathcal{R}$
2: $[\boldsymbol{Q}_1, R_{11}] =$ IntraOrtho $(\boldsymbol{X}_1)$
3: for $k = 1, \dots, p-1$ do
4:     $\mathcal{R}^{(1)}_{1:k,k+1} = \boldsymbol{\mathcal{Q}}^T_{1:k}\boldsymbol{X}_{k+1}$ % first BCGS step
5:     $\boldsymbol{W} = \boldsymbol{X}_{k+1} - \boldsymbol{\mathcal{Q}}_{1:k}\mathcal{R}^{(1)}_{1:k,k+1}$
6:     $[\widehat{\boldsymbol{Q}}, R^{(1)}_{k+1,k+1}] =$ IntraOrtho $(\boldsymbol{W})$
7:     $\mathcal{R}^{(2)}_{1:k,k+1} = \boldsymbol{\mathcal{Q}}^T_{1:k}\widehat{\boldsymbol{Q}}$ % second BCGS step
8:     $\boldsymbol{W} = \widehat{\boldsymbol{Q}} - \boldsymbol{\mathcal{Q}}_{1:k}\mathcal{R}^{(2)}_{1:k,k+1}$
9:     $[\boldsymbol{Q}_{k+1}, R^{(2)}_{k+1,k+1}] =$ IntraOrtho $(\boldsymbol{W})$
10:    $\mathcal{R}_{1:k,k+1} = \mathcal{R}^{(1)}_{1:k,k+1} + \mathcal{R}^{(2)}_{1:k,k+1}R^{(1)}_{k+1,k+1}$ % combine both steps
11:    $R_{k+1,k+1} = R^{(2)}_{k+1,k+1}R^{(1)}_{k+1,k+1}$
12: end for
13: return $\boldsymbol{\mathcal{Q}} = [\boldsymbol{Q}_1, \dots, \boldsymbol{Q}_p]$, $\mathcal{R} = (R_{jk})$
```

2.1.3. BCGS with selective replacement and reorthogonalization

Reorthogonalization is not enough to recover stability for some (pathologically bad) matrices. In response to such situations, Stewart [36] developed what we call CGS with Selective Reorthogonalization and Replacement (CGSS+rpl, Algorithm 15), which replaces low-quality vectors with random ones of the same magnitude. He also developed a block variant (`BCGSS+rpl`, Algorithm 5), designed to work specifically with `CGSS+rpl` as its muscle.

One of the primary reasons Stewart developed this variant was to account for *orthogonalization faults*, which may occur when applying an `IntraOrtho` twice to the same block vector, possibly causing the norm of some columns to drop drastically due to extreme cancellation error and therefore lose orthogonality with respect to the columns of previously computed orthogonal block vectors. Hoemmen describes this situation as "naive reorthogonalization" in Section 2.4.7 of his thesis [24] and provides a theoretical example in Appendix C.2 therein.

Both algorithms are highly technical and are perhaps best understood via the body of MATLAB code accompanying this paper, especially the functions `cgs_step_sror` and `bcgs_step_sror`, which are reproduced in Appendix B[7]. These routines are in fact the core of the algorithms and perform a careful reorthogonalization step that checks the quality of the reorthogonalization via a series of before-and-after norm comparisons and replaces reorthogonalized vectors with random ones if the quality is not satisfactory. The subroutine `cgs_step_sror` additionally replaces identically zero vectors with random ones of small norm, an essential feature that is not explicitly mentioned in Stewart's paper [36] but is present in his MATLAB implementations. This feature is key in ensuring that `CGSS+rpl` and `BCGSS+rpl` can handle severely rank-deficient matrices, i.e., that they return a full-rank, orthogonal $\boldsymbol{\mathcal{Q}}$ while passing rank deficiency onto $\mathcal{R}$.

A major downside to `CGSS+rpl` is that it can be as expensive as `HouseQR`, due to the many norm computations and the potential for more than two orthogonalization steps per vector. `BCGSS+rpl` suffers the same fate and is generally ill-suited to high-performance contexts, because the frequent column-wise norm computations create communication bottlenecks.

It is also possible to formulate such an algorithm based on `MGS`; see, e.g., [24, Algorithm 13]). We do not explore this further here, however, because there is little practical benefit from the communication point-of-view.

Algorithm 5 $[\boldsymbol{\mathcal{Q}}, \mathcal{R}] =$ `BCGSS+rpl`$(\boldsymbol{\mathcal{X}}, \delta)$

```
1: Allocate memory for $\boldsymbol{\mathcal{Q}}$ and $\mathcal{R}$
2: $[\boldsymbol{Q}_1, \sim, R_{11}] = \texttt{bcgs\_step\_sror}(\boldsymbol{0}, \boldsymbol{X}_1, \delta)$
3: for $k = 1, \dots, p-1$ do
4:     $[\boldsymbol{Q}_{k+1}, \mathcal{R}_{1:k,k+1}, R_{k+1,k+1}] = \texttt{bcgs\_step\_sror}(\boldsymbol{\mathcal{Q}}_{1:k}, \boldsymbol{X}_{k+1}, \delta)$
5: end for
6: return $\boldsymbol{\mathcal{Q}} = [\boldsymbol{Q}_1, \dots, \boldsymbol{Q}_s]$, $\mathcal{R} = (R_{jk})$
```

[7]These routines are precisely Stewart's original implementations, accessed in February 2020 at a cached version of `ftp://ftp.umiacs.umd.edu/pub/stewart/reports/Contents.html`. It is unclear if this page will remain accessible in the future. Because the full algorithms are not printed in the paper [36], we have therefore incorporated the original implementations into our package, to facilitate and preserve accessibility.

2.1.4. One-sync `BCGSI+` *(*`BCGSI+LS`*)*

Recently, a number of low-synchronization variants of Gram-Schmidt have been proposed [25]. Here, a *synchronization point* refers to a global reduction which requires $\log P$ time to complete for P processors or MPI ranks. This occurs with operations such as inner products and norms because large vectors are often stored and operated on in a distributed fashion.

`CGSI+` (if implemented like Algorithm 4 with $s = 1$) has up to four synchronization points per column, while Algorithm 3 from [25]– which we denote here as `CGSI+LS`, where "LS" stands for "low-synchronization"– has just one per column. See Algorithm 6. Note that it is formulated a bit differently from the presentation in [25]; in particular, lines 21-23 introduce a single final synchronization point necessary for orthogonalizing the last column.

Algorithm 6 $[\boldsymbol{Q}, R] =$ `CGSI+LS`$(\boldsymbol{X})$

```
 1: Allocate memory for $\boldsymbol{Q}$ and $R$
 2: $\boldsymbol{u} = \boldsymbol{x}_1$
 3: for $k = 2, \dots s$ do
 4:     if $k = 2$ then
 5:         $[r_{k-1,k-1}^2 \ \rho] = \boldsymbol{u}^T [\boldsymbol{u} \ \boldsymbol{x}_k]$
 6:     else if $k > 2$ then
 7:         $\begin{bmatrix} \boldsymbol{w} & \boldsymbol{z} \\ \omega & \zeta \end{bmatrix} = [\boldsymbol{Q}_{1:k-2} \ \boldsymbol{u}]^T [\boldsymbol{u} \ \boldsymbol{x}_k]$
 8:         $[r_{k-1,k-1}^2 \ \rho] = [\omega \ \zeta] - \boldsymbol{w}^T [\boldsymbol{w} \ \boldsymbol{z}]$
 9:     end if
10:     $r_{k-1,k} = \rho / r_{k-1,k-1}$
11:     if $k = 2$ then
12:         $\boldsymbol{q}_{k-1} = \boldsymbol{u} / r_{k-1,k-1}$
13:     else if $k > 2$ then
14:         $R_{1:k-2,k-1} = R_{1:k-2,k-1} + \boldsymbol{w}$
15:         $R_{1:k-2,k} = \boldsymbol{z}$
16:         $\boldsymbol{q}_{k-1} = (\boldsymbol{u} - \boldsymbol{Q}_{1:k-2}\boldsymbol{w}) / r_{k-1,k-1}$
17:     end if
18:     $\boldsymbol{u} = \boldsymbol{x}_k - \boldsymbol{Q}_{1:k-1} R_{1:k-1,k}$
19: end for
20: $\begin{bmatrix} \boldsymbol{w} \\ \omega \end{bmatrix} = [\boldsymbol{Q}_{1:s-1} \ \boldsymbol{u}]^T \boldsymbol{u}$
21: $r_{s,s}^2 = \omega - \boldsymbol{w}^T \boldsymbol{w}$
22: $R_{1:s-1,s} = R_{1:s-1,s} + \boldsymbol{w}$
23: $\boldsymbol{q}_s = (\boldsymbol{u} - \boldsymbol{Q}_{1:s-1}\boldsymbol{w}) / r_{s,s}$
24: return $\boldsymbol{Q} = [\boldsymbol{q}_1, \dots, \boldsymbol{q}_s]$, $R = (r_{jk})$
```

Note that `CGSI+LS` differs from Cholesky-based QR approaches such as `CholQR`, which has a single synchronization point for the entire algorithm. Reorthogonalized `CholQR` (`CholQR+`) [12] and shifted and reorthogonalized `CholQR` (`ShCholQR++`) [13] require two and three total synchronization points, respectively. See Algorithms 20-22 to compare details.

The low-synchronization algorithms from [25] are based on two ideas. The first is to compute a strictly lower triangular matrix L (i.e., with zeros on the diagonal) one row or block of rows at a time in a single global reduction to account for all the inner products needed in the current iteration. Each row $L_{k-1,1:k-2} = (\boldsymbol{Q}_{1:k-2}^T \boldsymbol{q}_{k-1})^T$ is obtained one at a time within the current step. The second idea is to lag the normalization step and merge it into this single reduction.

Ruhe [37] observed that `MGS` and `CGS` could be interpreted as Gauss-Seidel and Gauss-Jacobi iterations, respectively, for solving the normal equations where the associated orthogonal projector is given as

$$I - \boldsymbol{Q}_{1:k-1} T_{1:k-1,1:k-1} \boldsymbol{Q}_{1:k-1}^T, \text{ for } T_{1:k-1,1:k-1} \approx (\boldsymbol{Q}_{1:k-1}^T \boldsymbol{Q}_{1:k-1})^{-1}$$

For `CGSI+LS`, the matrix T would be given iteratively by the following (originally deduced in [25]):

$$T_{1:k-1,1:k-1} = I - L_{1:k-1,1:k-1} - L_{1:k-1,1:k-1}^T.$$

Another way to think of what `CGSI+LS` does per step is the following: one can split the auxiliary matrix $T_{1:k-1,1:k-1}$ into two parts and apply them across two iterations. The lower triangular matrix $I - L_{1:k-1,1:k-1}$ is applied first, followed by a lagged correction

$$R_{1:k-2,k-1} = R_{1:k-2,k-1} + \boldsymbol{w}, \quad \boldsymbol{w} = -L_{k-1,1:k-2}^T,$$

which is a delayed reorthogonalization step in the next iteration; see line 14 in Algorithm 6, but note that normalization there is delayed. Thus, the notion of reorthogonalization is modified and occurs "on-the-fly," as opposed to requiring a complete second pass of the algorithm. We emphasize, however, that T and L are not computed explicitly in Algorithm 6.

A block generalization of Algorithm 6 is rather straightforward; see Algorithm 7. One only has to be careful with the diagonal entries of $\mathcal{R}$, which are now $s \times s$ matrices. In particular, line 5 from Algorithm 6 must be replaced with a Cholesky factorization, and instead of dividing by r_{k-1} in lines 10, 12, and 16, it is necessary to invert either $R_{k-1,k-1}$ or its transpose; compare with lines 10, 12, and 16 of Algorithm 7. Note that line 10 in particular is tricky: by combining previous quantities, it holds that

$$P = \boldsymbol{U}^T \boldsymbol{X}_k \text{ or } \boldsymbol{U}^T \big(I - \boldsymbol{\mathcal{Q}}_{1:k-2} \boldsymbol{\mathcal{Q}}_{1:k-2}^T\big) \boldsymbol{X}_k,$$

so we must apply $R_{k-1,k-1}^{-T}$ from the left in order to properly scale the $\boldsymbol{U}$ "hidden" in P.

Our block generalization in Algorithm 7 is essentially the same as [30, Figure 3]. There are no existing theoretical studies on the stability of `BCGSI+LS`. However, it is clear that unlike `BCGSI+`, `BCGSI+LS` is not guaranteed to exhibit $\mathcal{O}(\varepsilon)$ loss of orthogonality even with an unconditionally stable `IntraOrtho`, likely due to the delayed operations; see Sections 3 and 4.4.

Algorithm 7 $[\boldsymbol{\mathcal{Q}}, \mathcal{R}] = \texttt{BCGSI+LS}(\boldsymbol{\mathcal{X}})$

```
 1: Allocate memory for $\boldsymbol{\mathcal{Q}}$ and $\mathcal{R}$
 2: $\boldsymbol{U} = \boldsymbol{X}_1$
 3: for $k = 2, \ldots p$ do
 4:     if $k = 2$ then
 5:         $[R_{k-1,k-1}^T R_{k-1,k-1} \; P] = \boldsymbol{U}^T [\boldsymbol{U} \; \boldsymbol{X}_k]$
 6:     else if $k > 2$ then
 7:         $\begin{bmatrix} \boldsymbol{W} & \boldsymbol{Z} \\ \Omega & Z \end{bmatrix} = [\boldsymbol{\mathcal{Q}}_{1:k-2} \; \boldsymbol{U}]^T [\boldsymbol{U} \; \boldsymbol{X}_k]$
 8:         $[R_{k-1,k-1}^T R_{k-1,k-1} \; P] = [\Omega \; Z] - \boldsymbol{W}^T [\boldsymbol{W} \; \boldsymbol{Z}]$
 9:     end if
10:     $R_{k-1,k} = R_{k-1,k-1}^{-T} P$
11:     if $k = 2$ then
12:         $\boldsymbol{Q}_{k-1} = \boldsymbol{U} R_{k-1,k-1}^{-1}$
13:     else if $k > 2$ then
14:         $\mathcal{R}_{1:k-2,k-1} = \mathcal{R}_{1:k-2,k-1} + \boldsymbol{W}$
15:         $\mathcal{R}_{1:k-2,k} = \boldsymbol{Z}$
16:         $\boldsymbol{Q}_{k-1} = (\boldsymbol{U} - \boldsymbol{\mathcal{Q}}_{1:k-2} \boldsymbol{W}) R_{k-1,k-1}^{-1}$
17:     end if
18:     $\boldsymbol{U} = \boldsymbol{X}_k - \boldsymbol{\mathcal{Q}}_{1:k-1} \mathcal{R}_{1:k-1,k}$
19: end for
20: $\begin{bmatrix} \boldsymbol{W} \\ \Omega \end{bmatrix} = [\boldsymbol{\mathcal{Q}}_{1:s-1} \; \boldsymbol{U}]^T \boldsymbol{U}$
21: $R_{s,s}^T R_{s,s} = \Omega - \boldsymbol{W}^T \boldsymbol{W}$
22: $\mathcal{R}_{1:s-1,s} = \mathcal{R}_{1:s-1,s} + \boldsymbol{W}$
23: $\boldsymbol{Q}_s = (\boldsymbol{U} - \boldsymbol{\mathcal{Q}}_{1:s-1} \boldsymbol{W}) R_{s,s}^{-1}$
24: return $\boldsymbol{\mathcal{Q}} = [\boldsymbol{Q}_1, \ldots, \boldsymbol{Q}_s]$, $\mathcal{R} = (R_{jk})$
```

2.2. Block modified Gram-Schmidt skeletons

2.2.1. Block Modified Gram-Schmidt (BMGS)

Much like BCGS, BMGS is obtained through a straightforward replacement of the column vectors of MGS with block vectors; see Algorithm 8. An initial stability study can be found in [18], and the algorithm was perhaps first considered in a high-performance setting by Vital [38].

A quick comparison between Algorithms 1 and 8 reveals that BMGS has many more synchronization points than BCGS and therefore suffers from higher communication demands. Specifically, BCGS has just one (block) inner product per block vector, whereas BMGS splits this up into k smaller (still block) inner products for the $(k+1)$st block vector. At the same time, it is precisely this strategy that generally makes BMGS more stable than BCGS, which we discuss in more detail in Section 4.5.

Algorithm 8 $[\mathcal{Q}, \mathcal{R}] = \texttt{BMGS}(\mathcal{X})$

```
1:  Allocate memory for $\mathcal{Q}$ and $\mathcal{R}$
2:  $[Q_1, R_{11}] = \texttt{IntraOrtho}(X_1)$
3:  for $k = 1, \dots, p-1$ do
4:      $W = X_{k+1}$
5:      for $j = 1, \dots, k$ do
6:          $R_{j,k+1} = Q_j^T W$
7:          $W = W - Q_j R_{j,k+1}$
8:      end for
9:      $[Q_{k+1}, R_{k+1,k+1}] = \texttt{IntraOrtho}(W)$
10: end for
11: return $\mathcal{Q} = [Q_1, \dots, Q_p]$, $\mathcal{R} = (R_{jk})$
```

2.2.2. Low-sync BMGS variants

Although increases in floating-point operations or communication often correlate with better stability properties, several new low-synchronization variants of MGS have been developed recently that appear to challenge this maxim. We describe all four such variants here and show how they can be turned into block algorithms. A full stability analysis for most of these algorithms remains open, but we outline a path forward in Section 4.6.

Low-sync MGS variants. Recall that CGS (Algorithm 1 with $s = 1$) has two synchronization points per vector, namely the inner product in line 4 and the norm (IntraOrtho) in line 6, while MGS (Algorithm 8 with $s = 1$) has a linearly increasing number of $k + 1$ synchronization points per vector, where k increases linearly with the vector index. The core steps of these algorithms can be expressed as the application of a projector or a sequence of projectors on the "next" vector:

$$\texttt{CGS}: \quad (I - Q_{1:k} Q_{1:k}^T) x_{k+1} \tag{6}$$

$$\texttt{MGS}: \quad (I - q_k q_k^T) \cdots (I - q_1 q_1^T) x_{k+1} \tag{7}$$

It is well known that the application of the single projector (6) is unstable, but breaking it up like in (7) improves stability. A third option would be to compute CGS's projector slightly differently, for example, by "cushioning" it with a correction matrix $C_k \in \mathbb{R}^{s \times s}$:

$$(I - \boldsymbol{Q}_{1:k} C_k \boldsymbol{Q}_{1:k}^T)\boldsymbol{x}_{k+1}.$$

An algorithm built from such projectors would then communicate like CGS. The challenge, then, is to find C_k so that orthogonality is lost like $\mathcal{O}(\varepsilon)\,\kappa(\boldsymbol{\mathcal{X}})$ or better. The matrix T from Section 2.1.4 would be one such candidate. The T matrix is an approximation of the inverse $(\boldsymbol{Q}_{1:k}^T \boldsymbol{Q}_{1:k})^{-1}$ from particular normal equations derived by Ruhe [37].

Recently, four such algorithms have been developed. The first two (`MGS-SVL` and `MGS-LTS`) only seek to eliminate the increasing number of inner products and have two synchronization points per vector; the latter two (`MGS-ICWY` and `MGS-CWY`) use a technique called "normalization lagging," introduced by Kim and Chronopoulos (1992) [39], in order to bring the number of synchronization points per vector down to one:

- `MGS-SVL` [15, Func. 3.1]: Barlow calls this variant "MGS2." We prefer the suffix "SVL" for "Schreiber and Van Loan," on whose work [40] the development of this algorithm is largely based. See Algorithm 16 in Appendix A.
- `MGS-LTS` [25, Alg. 4]: This variant is not assigned a name in [25]; it is designated only as "Algorithm 4." We refer to it with the suffix "LTS" in this note, which stands for "Lower Triangular Solve," due to how the correction matrix is handled. See Algorithm 17 in Appendix A.
- `MGS-CWY` [25, Alg. 6]: The suffix here stands for "Compact WY," and the algorithm turns out to be the one-sync version of `MGS-SVL`. See Algorithm 18 in Appendix A.
- `MGS-ICWY` [25, Alg. 5]: The suffix here stands for "Inverse Compact WY," referring to an alternative formulation of the `MGS` projector (7). It turns out that `MGS-ICWY` is the one-sync version of `MGS-LTS`. See Algorithm 19 in Appendix A.

The differences between the four algorithms are highlighted in red.

The forms of the projectors for these variants are quite similar, although the correction matrix itself is different for all algorithms, especially in floating-point arithmetic:

$$\texttt{MGS-SVL}, \texttt{MGS-CWY}: \quad (I - \boldsymbol{Q}_{1:k} T_{1:k,1:k}^T \boldsymbol{Q}_{1:k}^T)\boldsymbol{x}_{k+1} \tag{8}$$

$$\texttt{MGS-LTS}, \texttt{MGS-ICWY}: \quad (I - \boldsymbol{Q}_{1:k} T_{1:k,1:k}^{-T} \boldsymbol{Q}_{1:k}^T)\boldsymbol{x}_{k+1} \tag{9}$$

The projector for `MGS-CWY` has the same form as that of `MGS-SVL`, because both represent the product of rank-one elementary projectors as a recursively constructed triangular matrix T. The same is true for `MGS-ICWY` and `MGS-LTS`; see, e.g., Walker [41] and the technical report by Sun [42]. To see how `MGS-CWY` and `MGS-SVL` share the same form of projector directly from the algorithms, start with `MGS-CWY`, and assume $k > 1$. Then from line 14 of Algorithm 18, we have that

$$\begin{aligned}
\boldsymbol{u} &= \boldsymbol{w} - \boldsymbol{Q}_{:,1:k} R_{1:k,k+1} \\
&= \boldsymbol{w} - \boldsymbol{Q}_{:,1:k} T^T_{1:k,1:k} \begin{bmatrix} \boldsymbol{r} \\ \rho / r_{k,k} \end{bmatrix}, \text{ from line 12} \\
&= \boldsymbol{w} - \boldsymbol{Q}_{:,1:k} T^T_{1:k,1:k} \begin{bmatrix} \boldsymbol{Q}^T_{1:k-1} \boldsymbol{w} \\ \boldsymbol{u}^T \boldsymbol{w} / r_{k,k} \end{bmatrix}, \text{ from line 9} \\
&= \boldsymbol{x}_{k+1} - \boldsymbol{Q}_{:,1:k} T^T_{1:k,1:k} \boldsymbol{Q}^T_{1:k} \boldsymbol{x}_{k+1}, \text{ from lines 13 and 5.}
\end{aligned}$$

By similar means, `MGS-LTS` and `MGS-ICWY` can be related.

We also note that `MGS-SVL` and `MGS-CWY` use only matrix-matrix multiplication to apply their correction matrices, while `MGS-LTS` and `MGS-ICWY` use lower triangular solves. Depending on how these kernels are implemented, one variant may be preferred over another in practice. The T^T matrix comes from the $(2, 2)$ block of the transposed Householder matrix U^T described by Barlow; see [15], equation (2.17) on page 1262.

Block generalizations. Block generalizations of both `MGS-SVL` (Algorithm 9) and `MGS-LTS` (Algorithm 10) are quite straightforward. In [15], `BMGS-SVL` appears as `MGS3` and `BMGS_H` with `IntraOrtho`s `MGS-SVL` (Algorithm 16) and `HouseQR` (i.e., Householder-based QR, as in, e.g., [43]), respectively. `BMGS-LTS` as a BGS variant is new, as far as we are aware.

One quirk about these low-sync variants is that, unlike `BCGS` or `BMGS`, they require an `IntraOrtho` that produces a T-factor, in addition to $\boldsymbol{Q}$ and R. For algorithms that do not explicitly produce such a factor, one must assume that $T = I$. This correction matrix is necessary to ensuring the stability of the overall algorithm. Furthermore, there is a kind of compatibility requirement between T-producing `IntraOrtho`s and low-sync skeletons: combinations like `BMGS` $\circ$ `MGS-SVL` or `BMGS-SVL` $\circ$ `MGS-LTS`, for example, may not produce the expected stability behavior. We explore this issue further in Section 3.

It is also possible to derive block generalizations of `MGS-CWY` and `MGS-ICWY` that are still one-sync and mostly stable; see [30, Figure 3]. Instead of the implied square root in line 7 of Algorithms 18 and 19, a Cholesky factorization is needed to recover the block diagonal entry $R_{k,k}$. The computation of $\boldsymbol{Q}_k$ is then completed by inverting the $s \times s$ upper triangular matrix $R_{k,k}$. As with `BCGSI+LS` (cf. Section 2.1.4), one has to take care about when to apply $R^{-1}_{k,k}$ or $R^{-T}_{k,k}$: line 12 of both Algorithms 11 and 12 require applying $R^{-T}_{k,k}$ to properly scale the "hidden" $\boldsymbol{U}$ in P. Note that both the Cholesky factorization and matrix inverse can be computed locally, because s is small. The final orthogonalization in lines 16-17 of Algorithms 18 and 19 can be replaced by an `IntraOrtho` or another Cholesky factorization; we opt for an `IntraOrtho` in line 16 of Algorithms 11 and 12.

For further discussion of the stability properties of these low-sync block methods, see Section 4.6.

Algorithm 9 $[\mathcal{Q}, \mathcal{R}, \mathcal{T}] = \texttt{BMGS-SVL}(\mathcal{X})$

```
1: Allocate memory for $\mathcal{Q}$, $\mathcal{R}$, and $\mathcal{T}$
2: $[Q_1, R_{11}, T_{11}] = \texttt{IntraOrtho}(X_1)$
3: for $k = 1, \ldots, p-1$ do
4:     $\mathcal{R}_{1:k,k+1} = \mathcal{T}_{1:k,1:k}^T (\mathcal{Q}_{1:k}^T X_{k+1})$
5:     $W = X_{k+1} - \mathcal{Q}_{1:k} \mathcal{R}_{1:k,k+1}$
6:     $[Q_{k+1}, R_{k+1,k+1}, T_{k+1,k+1}] = \texttt{IntraOrtho}(W)$
7:     $\mathcal{T}_{1:k,k+1} = -\mathcal{T}_{1:k,1:k} (\mathcal{Q}_{1:k}^T Q_{k+1}) T_{k+1,k+1}$
8: end for
9: return $\mathcal{Q} = [Q_1, \ldots, Q_p]$, $\mathcal{R} = (R_{jk})$, $\mathcal{T} = (T_{jk})$
```

Algorithm 10 $[\mathcal{Q}, \mathcal{R}, \mathcal{T}] = \texttt{BMGS-LTS}(X)$

```
1: Allocate memory for $\mathcal{Q}$, $\mathcal{R}$, and $\mathcal{T}$
2: $[Q_1, R_{11}, T_{11}] = \texttt{IntraOrtho}(X_1)$
3: for $k = 1, \ldots, p-1$ do
4:     $\mathcal{R}_{1:k,k+1} = \mathcal{T}_{1:k,1:k}^{-T} (\mathcal{Q}_{1:k}^T X_{k+1})$
5:     $W = X_{k+1} - \mathcal{Q}_{1:k} \mathcal{R}_{1:k,k+1}$
6:     $[Q_{k+1}, R_{k+1,k+1}, T_{k+1,k+1}] = \texttt{IntraOrtho}(W)$
7:     $\mathcal{T}_{1:k,k+1} = (\mathcal{Q}_{1:k}^T Q_{k+1}) T_{k+1,k+1}$
8: end for
9: return $\mathcal{Q} = [Q_1, \ldots, Q_p]$, $\mathcal{R} = (R_{jk})$, $\mathcal{T} = (T_{jk})$
```

2.2.3. Dynamic `BMGS` *(*`DBMGS`*)*

Dynamic `BMGS` is essentially `BMGS` with variable block sizes determined adaptively in order to reduce the condition numbers of block vectors to be orthogonalized. Vanderstraeten first proposed `DBMGS` in 2000, particularly with `MGS` in mind as the `IntraOrtho` [17]. The communication costs associated with this approach can be expected to be somewhere between that of `MGS` (i.e., with a block size of 1) and that of `BMGS` using the specified maximum block size, depending on the numerical properties of the input data. We will not devote further discussion to `DBMGS` here, because we assume that the block partitioning of $\mathcal{X}$ is fixed a priori. However, a stability analysis of `DBMGS` would greatly inform that of adaptive s-step algorithms [44].

Algorithm 11 $[\boldsymbol{\mathcal{Q}}, \mathcal{R}, \mathcal{T}] =$ `BMGS-CWY`$(\boldsymbol{\mathcal{X}})$

```
1:  Allocate memory for $\boldsymbol{\mathcal{Q}}$ and $\mathcal{R}$
2:  $\mathcal{T} = I_n$
3:  $\boldsymbol{U} = \boldsymbol{X}_1$
4:  for $k = 1, \dots, p-1$ do
5:      $\boldsymbol{W} = \boldsymbol{X}_{k+1}$
6:      if $k = 1$ then
7:          $\begin{bmatrix} R_{k,k}^T R_{k,k} & P \end{bmatrix} = \boldsymbol{U}^T [\boldsymbol{U} \;\; \boldsymbol{W}]$ % recover $R_{k,k}$ with chol
8:      else if $k > 1$ then
9:          $\begin{bmatrix} \boldsymbol{T} & \boldsymbol{R} \\ R_{k,k}^T R_{k,k} & P \end{bmatrix} = [\boldsymbol{\mathcal{Q}}_{1:k-1} \;\; \boldsymbol{U}]^T [\boldsymbol{U} \;\; \boldsymbol{W}]$ % recover $R_{k,k}$ with chol
10:         $\mathcal{T}_{1:k-1,k} = -\mathcal{T}_{1:k-1,1:k-1}(\boldsymbol{T} R_{k,k}^{-1})$
11:     end if
12:     $\mathcal{R}_{1:k,k+1} = \mathcal{T}_{1:k,1:k}^T \begin{bmatrix} \boldsymbol{R} \\ R_{k,k}^{-T} P \end{bmatrix}$
13:     $\boldsymbol{Q}_k = \boldsymbol{U} R_{k,k}^{-1}$
14:     $\boldsymbol{U} = \boldsymbol{W} - \boldsymbol{\mathcal{Q}}_{:,1:k} \mathcal{R}_{1:k,k+1}$
15: end for
16: $[\boldsymbol{Q}_p, R_{p,p}] =$ IntraOrtho $(\boldsymbol{U})$
17: return $\boldsymbol{\mathcal{Q}} = [\boldsymbol{Q}_1, \dots, \boldsymbol{Q}_p]$, $\mathcal{R} = (R_{jk})$, $\mathcal{T} = (T_{jk})$
```

Algorithm 12 $[\boldsymbol{\mathcal{Q}}, \mathcal{R}, \mathcal{T}] =$ `BMGS-ICWY`$(\boldsymbol{\mathcal{X}})$

```
1:  Allocate memory for $\boldsymbol{\mathcal{Q}}$ and $\mathcal{R}$
2:  $\mathcal{T} = I_n$
3:  $\boldsymbol{U} = \boldsymbol{X}_1$
4:  for $k = 1, \dots, p-1$ do
5:      $\boldsymbol{W} = \boldsymbol{X}_{k+1}$
6:      if $k = 1$ then
7:          $\begin{bmatrix} R_{k,k}^T R_{k,k} & P \end{bmatrix} = \boldsymbol{U}^T [\boldsymbol{U} \;\; \boldsymbol{W}]$ % recover $R_{k,k}$ with chol
8:      else if $k > 1$ then
9:          $\begin{bmatrix} \boldsymbol{T} & \boldsymbol{R} \\ R_{k,k}^T R_{k,k} & P \end{bmatrix} = [\boldsymbol{\mathcal{Q}}_{1:k-1} \;\; \boldsymbol{U}]^T [\boldsymbol{U} \;\; \boldsymbol{W}]$ % recover $R_{k,k}$ with chol
10:         $\mathcal{T}_{1:k-1,k} = \boldsymbol{T} R_{k,k}^{-1}$
11:     end if
12:     $\mathcal{R}_{1:k,k+1} = \mathcal{T}_{1:k,1:k}^{-T} \begin{bmatrix} \boldsymbol{R} \\ R_{k,k}^{-T} P \end{bmatrix}$
13:     $\boldsymbol{Q}_k = \boldsymbol{U} R_{k,k}^{-1}$
14:     $\boldsymbol{U} = \boldsymbol{W} - \boldsymbol{\mathcal{Q}}_{:,1:k} \mathcal{R}_{1:k,k+1}$
15: end for
16: $[\boldsymbol{Q}_p, R_{p,p}] =$ IntraOrtho $(\boldsymbol{U})$
17: return $\boldsymbol{\mathcal{Q}} = [\boldsymbol{Q}_1, \dots, \boldsymbol{Q}_p]$, $\mathcal{R} = (R_{jk})$, $\mathcal{T} = (T_{jk})$
```

3. Skeleton stability in terms of muscle stability: overview

Viewing BGS algorithms through the skeleton-muscle framework begs the following questions:

(Q.1) If we use an unconditionally stable muscle, what is the best a skeleton can do?

(Q.2) What are the minimum requirements on the muscle such that a given skeleton is stable?

To answer either question, it will be helpful to keep in mind the stability properties of different muscles, summarized in Table 3. Additionally, we summarize known and conjectured results for block algorithms in Table 4. For the specific assumptions on the muscles that lead to these stability bounds, see the exposition in Section 4.

Table 3: Upper bounds on the loss of orthogonality in the $\bar{\boldsymbol{Q}}$ factor for various `IntraOrtho`s, along with proof references. Note that all conditions have hidden constants in terms of polynomials of the dimensions m and s. A superscript dagger † indicates that the result is conjectured based on numerical observations, but not yet proven.

`IntraOrtho`	$\left\lVert I - \bar{\boldsymbol{Q}}^T \bar{\boldsymbol{Q}} \right\rVert_2$	Assumption on $\kappa(\boldsymbol{X})$	Reference(s)
`CGS`	$\mathcal{O}(\varepsilon)\,\kappa^{n-1}(\boldsymbol{X})$	$\mathcal{O}(\varepsilon)\,\kappa(\boldsymbol{X}) < 1$	[45]
`CGS-P`	$\mathcal{O}(\varepsilon)\,\kappa^2(\boldsymbol{X})$	$\mathcal{O}(\varepsilon)\,\kappa^2(\boldsymbol{X}) < 1$	[28]
`CholQR`	$\mathcal{O}(\varepsilon)\,\kappa^2(\boldsymbol{X})$	$\mathcal{O}(\varepsilon)\,\kappa^2(\boldsymbol{X}) < 1$	[12]
`MGS`	$\mathcal{O}(\varepsilon)\,\kappa(\boldsymbol{X})$	$\mathcal{O}(\varepsilon)\,\kappa(\boldsymbol{X}) < 1$	[8]
`MGS-SVL`	$\mathcal{O}(\varepsilon)\,\kappa(\boldsymbol{X})$	$\mathcal{O}(\varepsilon)\,\kappa(\boldsymbol{X}) < 1$	[15]
`MGS-LTS`	$\mathcal{O}(\varepsilon)\,\kappa(\boldsymbol{X})^\dagger$	$\mathcal{O}(\varepsilon)\,\kappa(\boldsymbol{X}) < 1^\dagger$	conjecture [25]
`MGS-CWY`	$\mathcal{O}(\varepsilon)\,\kappa(\boldsymbol{X})^\dagger$	$\mathcal{O}(\varepsilon)\,\kappa(\boldsymbol{X}) < 1^\dagger$	conjecture [25]
`MGS-ICWY`	$\mathcal{O}(\varepsilon)\,\kappa(\boldsymbol{X})^\dagger$	$\mathcal{O}(\varepsilon)\,\kappa(\boldsymbol{X}) < 1^\dagger$	conjecture [25]
`CholQR+`	$\mathcal{O}(\varepsilon)$	$\mathcal{O}(\varepsilon)\,\kappa^2(\boldsymbol{X}) < 1$	[12]
`CGSI+`	$\mathcal{O}(\varepsilon)$	$\mathcal{O}(\varepsilon)\,\kappa(\boldsymbol{X}) < 1$	[31, 16, 46]
`CGSS+`	$\mathcal{O}(\varepsilon)$	$\mathcal{O}(\varepsilon)\,\kappa(\boldsymbol{X}) < 1$	[33, 34]
`CGSI+LS`	$\mathcal{O}(\varepsilon)^\dagger$	$\mathcal{O}(\varepsilon)\,\kappa(\boldsymbol{X}) < 1^\dagger$	conjecture [25]
`MGS+`	$\mathcal{O}(\varepsilon)$	$\mathcal{O}(\varepsilon)\,\kappa(\boldsymbol{X}) < 1$	[18, 47]
`MGSI+`	$\mathcal{O}(\varepsilon)$	$\mathcal{O}(\varepsilon)\,\kappa(\boldsymbol{X}) < 1$	[34, 48]
`ShCholQR++`	$\mathcal{O}(\varepsilon)$	$\mathcal{O}(\varepsilon)\,\kappa(\boldsymbol{X}) < 1$	[13]
`CGSS+rpl`	$\mathcal{O}(\varepsilon)^\dagger$	none†	conjecture [36]
`HouseQR`	$\mathcal{O}(\varepsilon)$	none	[10, Sec. 19.3]
`GivensQR`	$\mathcal{O}(\varepsilon)$	none	[10, Sec. 19.6]
`TSQR`	$\mathcal{O}(\varepsilon)$	none	[26, 49]

To demonstrate stability properties for multiple skeleton-muscle combinations at once, we build heat-maps for a number of test matrices $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{m \times ps}$, where $m = 10000$ (number of rows), $p = 50$ (number of block vectors), and

Table 4: Upper bounds on the loss of orthogonality in the $\bar{\boldsymbol{\mathcal{Q}}}$ factor for BGS skeletons **composed with unconditionally stable `IntraOrtho`s**, along with proof references. Note that all conditions have hidden constants in terms of the dimensional parameters m, n, and s. A superscript dagger † indicates that the result is conjectured but lacks a rigorous proof.

`BGS`	$\left\lVert I - \bar{\boldsymbol{\mathcal{Q}}}^T \bar{\boldsymbol{\mathcal{Q}}} \right\rVert_2$	Assumption on $\kappa(\boldsymbol{\mathcal{X}})$	Reference(s)
`BCGS`	$\mathcal{O}(\varepsilon)\,\kappa^{n-1}(\boldsymbol{\mathcal{X}})^\dagger$	$\mathcal{O}(\varepsilon)\,\kappa(\boldsymbol{\mathcal{X}}) < 1^\dagger$	conjecture
`BCGS-P`	$\mathcal{O}(\varepsilon)\,\kappa^2(\boldsymbol{\mathcal{X}})$	$\mathcal{O}(\varepsilon)\,\kappa^2(\boldsymbol{\mathcal{X}}) < 1$	[29]
`BMGS`	$\mathcal{O}(\varepsilon)\,\kappa(\boldsymbol{\mathcal{X}})$	$\mathcal{O}(\varepsilon)\,\kappa(\boldsymbol{\mathcal{X}}) < 1$	[18], here
`BMGS-SVL`	$\mathcal{O}(\varepsilon)\,\kappa(\boldsymbol{\mathcal{X}})$	$\mathcal{O}(\varepsilon)\,\kappa(\boldsymbol{\mathcal{X}}) < 1$	[15]
`BMGS-LTS`	$\mathcal{O}(\varepsilon)\,\kappa(\boldsymbol{\mathcal{X}})^\dagger$	$\mathcal{O}(\varepsilon)\,\kappa(\boldsymbol{\mathcal{X}}) < 1^\dagger$	conjecture, here
`BMGS-CWY`	$\mathcal{O}(\varepsilon)\,\kappa^2(\boldsymbol{\mathcal{X}})^\dagger$	$\mathcal{O}(\varepsilon)\,\kappa^2(\boldsymbol{\mathcal{X}}) < 1^\dagger$	conjecture, here
`BMGS-ICWY`	$\mathcal{O}(\varepsilon)\,\kappa^2(\boldsymbol{\mathcal{X}})^\dagger$	$\mathcal{O}(\varepsilon)\,\kappa^2(\boldsymbol{\mathcal{X}}) < 1^\dagger$	conjecture, here
`BCGSI+`	$\mathcal{O}(\varepsilon)$	$\mathcal{O}(\varepsilon)\,\kappa(\boldsymbol{\mathcal{X}}) < 1$	[16]
`BCGSS+rpl`	$\mathcal{O}(\varepsilon)^\dagger$	none†	conjecture [36]
`BCGSI+LS`	$\mathcal{O}(\varepsilon)\,\kappa^2(\boldsymbol{\mathcal{X}})^\dagger$	$\mathcal{O}(\varepsilon)\,\kappa^2(\boldsymbol{\mathcal{X}}) < 1^\dagger$	conjecture, here

$s = 10$ (columns per block vector). Additional matrix properties are given in Table 5. All heat-map plots are run in MATLAB 2019b on the r3d3 login node of the Sněhurka cluster, which consists of an Intel Xeon Processor E5-2620 with 15MB cache 2.00 GHz and operating system 18.04.4 LTS.[8] Our code is publicly available at `https://github.com/katlund/BlockStab`. Our test problems include:

- `rand_uniform`: The entries of $\boldsymbol{\mathcal{X}}$ are drawn randomly from the uniform distribution via the `rand` command in MATLAB.
- `rand_normal`: The entries of $\boldsymbol{\mathcal{X}}$ are drawn randomly from the normal distribution via the `randn` command in MATLAB.
- `rank_def`: $\boldsymbol{\mathcal{X}}$ is first generated like `rand_normal`. Then the first block vector is set to 100 times the last block vector, i.e., $\boldsymbol{X}_1 = 100\boldsymbol{X}_p$, to ensure that the matrix is numerically rank-deficient and badly scaled.
- `laeuchli`: $\boldsymbol{\mathcal{X}}$ is a Läuchli matrix of the form
$$\boldsymbol{\mathcal{X}} = \begin{bmatrix} 1 & 1 & \cdots & 1 \\ \eta & & & \\ & \eta & & \\ & & \ddots & \\ & & & \eta \end{bmatrix}, \quad \eta \in (\varepsilon, \sqrt{\varepsilon}),$$
where η is drawn randomly from a scaled uniform distribution. This matrix is interesting, because columns are only barely linearly independent.

[8] `http://cluster.karlin.mff.cuni.cz/`.

- `monomial`: A diagonal $m \times m$ operator A with evenly distributed eigenvalues in $(\frac{1}{10}, 10)$ is defined, and p vectors $\boldsymbol{v}_k$, $k = 1, \ldots, p$, are randomly generated from the uniform distribution and normalized. The matrix $\boldsymbol{\mathcal{X}}$ is then defined as the concatenation of p block vectors

$$\boldsymbol{X}_k = [\boldsymbol{v}_k \,|\, A\boldsymbol{v}_k \,|\, \cdots \,|\, A^{s-1}\boldsymbol{v}_k].$$

- `s-step`: $\boldsymbol{\mathcal{X}}$ is built similarly to `monomial`, but now $\boldsymbol{v}_k$ is the normalized version of $A^{s-1}\boldsymbol{v}_{k-1}$.
- `newton`: $\boldsymbol{\mathcal{X}}$ is built like `s-step` but with Newton polynomials instead of monomials.
- `stewart`: $\boldsymbol{\mathcal{X}} = \boldsymbol{\mathcal{U}}\Sigma\mathcal{V}^T$, where $\boldsymbol{\mathcal{U}} \in \mathbb{R}^{m\times n}$ and $\mathcal{V} \in \mathbb{R}^{n\times n}$ are random unitary matrices and the diagonal of $\Sigma \in \mathbb{R}^{n\times n}$ is the geometric sequence from 1 to 10^{-20}. The first and 25th columns of $\boldsymbol{\mathcal{X}}$ are the same, and the 35th column is identically zero.
- `stewart_extreme`: $\boldsymbol{\mathcal{X}} = \boldsymbol{\mathcal{U}}\Sigma\mathcal{V}^T$, where $\boldsymbol{\mathcal{U}} \in \mathbb{R}^{m\times n}$ and $\mathcal{V} \in \mathbb{R}^{n\times n}$ are random unitary matrices and the first half of the diagonal of Σ is the geometric sequence from 1 to 10^{-10}, while the second half is identically zero.

Table 5: Matrix properties. Note that singular values are computed via `svd` after $\boldsymbol{\mathcal{X}}$ is explicitly formed. When exact values are known, which is the case with the smallest singular value of `rank_def`, and all the singular values of `stewart` and `stewart_extreme`, those values are listed instead.

Matrix ID	σ_1	σ_n	$\kappa(\boldsymbol{\mathcal{X}})$
`rand_uniform`	1.12e+03	2.25e+01	4.96e+01
`rand_normal`	1.22e+02	7.77e+01	1.46e+00
`rank_def`	1.02e+04	0	`Inf`
`laeuchli`	2.24e+01	2.02e-11	1.11e+12
`monomial`	2.32e+08	3.04e-04	7.63e+11
`s-step`	2.08e+01	3.21e-17	6.50e+17
`newton`	3.15e+00	7.77e-04	4.06e+03
`stewart`	1	1e-20	1e+20
`stewart_extreme`	1	0	`Inf`

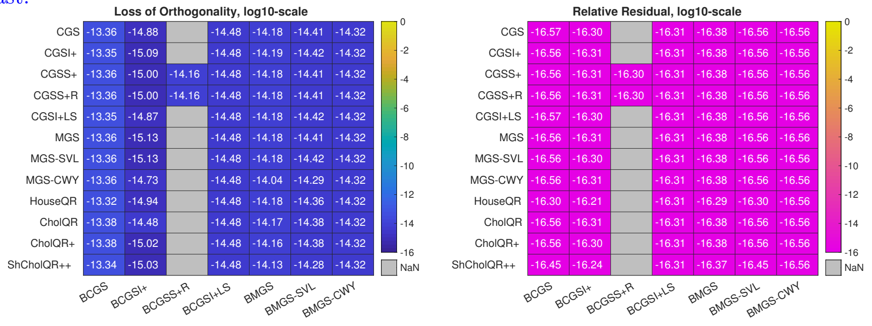
Note that while `stewart` and `stewart_extreme` are taken from [36], matrices like these have been used to study stability properties in Gram-Schmidt algorithms perhaps as early as Hoffman [34]. However, as we shall see, only Stewart's algorithm `BCGSS+rpl` and a couple variations of `BCGSI+` are consistently robust enough to reliably factor these matrices. We therefore believe the names `stewart` and `stewart_extreme` are well earned.

For each matrix, two heat-maps are produced – one for loss of orthogonality and one for relative residual – allowing for a quick and straightforward comparison of all methods at once. The colorbars indicate the value range. Note that `BCGSS+rpl` is only compatible with `CGSS+` and `CGSS+rpl`, so a `NaN` is returned for all other combinations. Note also that `BCGSS+rpl` and `CGSS+rpl` are denoted as 'BCGSS+R' and 'CGSS+R', respectively, in the axis labels. To demonstrate the effect of replacement, `CGSS+` has a replacement tolerance $\delta = 0$, while `CGSS+rpl` has $\delta = 100$, which Stewart [36] notes is relatively "aggressive." `BMGS-SVL` and `BMGS-LTS` have similar behavior, so we only include the former; likewise for `BMGS-CWY` and `BMGS-ICWY`. Recall that `BCGSI+LS` does not require an `IntraOrtho`, and `BMGS-CWY` only at the last step, so the reported values are identical for their columns. A `NaN` value is also returned when algorithms using Cholesky (`CholQR`, `CholQR+`, `ShCholQR++`, `BCGSI+LS`, and `BMGS-CWY`) encounter a matrix that is not numerically positive definite, because MATLAB's built-in `chol` throws an error.

3.1. *`rand_uniform`* and *`rand_normal`*

These tests serve primarily as sanity checks. In comparing Figures 1 and 2, there is little difference overall among the algorithms for either matrix. The only noticeable differences arise for `rand_uniform` (Figure 1) where we see that `BCGS` algorithms lose the most orthogonality, while `BCGSI+` the least; indeed, there is an order of magnitude difference between the two columns. Other small differences can be seen for the muscles `CGSI+LS` and `MGS-CWY` in Figure 2; for all methods they perform slightly worse.

Figure 1: Measurements for the `rand_uniform` matrix. There is relatively little difference between various algorithms; `BCGS` algorithms lose the most orthogonality, while `BCGSI+` the least.



Loss of Orthogonality, log10-scale

	BCGS	BCGSI+	BCGSS+R	BCGSI+LS	BMGS	BMGS-SVL	BMGS-CWY
CGS	-13.36	-14.88	▨	-14.48	-14.18	-14.41	-14.32
CGSI+	-13.35	-15.09	▨	-14.48	-14.19	-14.42	-14.32
CGSS+	-13.36	-15.00	-14.16	-14.48	-14.18	-14.41	-14.32
CGSS+R	-13.36	-15.00	-14.16	-14.48	-14.18	-14.41	-14.32
CGSI+LS	-13.35	-14.87	▨	-14.48	-14.18	-14.42	-14.32
MGS	-13.36	-15.13	▨	-14.48	-14.18	-14.41	-14.32
MGS-SVL	-13.36	-15.13	▨	-14.48	-14.18	-14.42	-14.32
MGS-CWY	-13.36	-14.73	▨	-14.48	-14.04	-14.29	-14.32
HouseQR	-13.32	-14.94	▨	-14.48	-14.18	-14.36	-14.32
CholQR	-13.38	-14.48	▨	-14.48	-14.17	-14.38	-14.32
CholQR+	-13.38	-15.02	▨	-14.48	-14.16	-14.38	-14.32
ShCholQR++	-13.34	-15.03	▨	-14.48	-14.13	-14.28	-14.32

Relative Residual, log10-scale

	BCGS	BCGSI+	BCGSS+R	BCGSI+LS	BMGS	BMGS-SVL	BMGS-CWY
CGS	-16.57	-16.30	▨	-16.31	-16.38	-16.56	-16.56
CGSI+	-16.56	-16.31	▨	-16.31	-16.38	-16.56	-16.56
CGSS+	-16.56	-16.31	-16.30	-16.31	-16.38	-16.56	-16.56
CGSS+R	-16.56	-16.31	-16.30	-16.31	-16.38	-16.56	-16.56
CGSI+LS	-16.57	-16.30	▨	-16.31	-16.38	-16.56	-16.56
MGS	-16.56	-16.31	▨	-16.31	-16.38	-16.56	-16.56
MGS-SVL	-16.56	-16.30	▨	-16.31	-16.38	-16.56	-16.56
MGS-CWY	-16.56	-16.31	▨	-16.31	-16.38	-16.56	-16.56
HouseQR	-16.30	-16.21	▨	-16.31	-16.29	-16.30	-16.56
CholQR	-16.56	-16.31	▨	-16.31	-16.38	-16.56	-16.56
CholQR+	-16.56	-16.30	▨	-16.31	-16.38	-16.56	-16.56
ShCholQR++	-16.45	-16.24	▨	-16.31	-16.37	-16.45	-16.56

3.2. *`rank_def`*

In Figure 3, we see that only the skeletons with reorthogonalization are robust enough to handle the matrix. It is further interesting to note that `BCGSI+` performs nearly the same regardless of `IntraOrtho`.

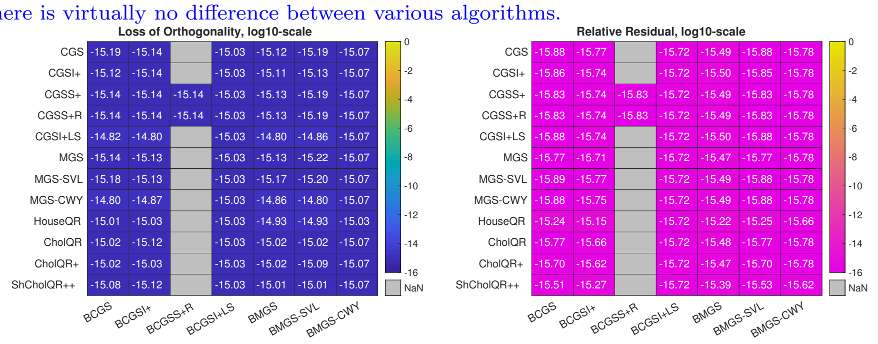


Figure 2: Measurements for the `rand_normal` matrix. For this very well-conditioned problem, there is virtually no difference between various algorithms.

Loss of Orthogonality, log10-scale

	BCGS	BCGSI+	BCGSS+R	BCGSI+LS	BMGS	BMGS-SVL	BMGS-CWY
CGS	-15.19	-15.14		-15.03	-15.12	-15.19	-15.07
CGSI+	-15.12	-15.14		-15.03	-15.11	-15.13	-15.07
CGSS+	-15.14	-15.14	-15.14	-15.03	-15.13	-15.19	-15.07
CGSS+R	-15.14	-15.14	-15.14	-15.03	-15.13	-15.19	-15.07
CGSI+LS	-14.82	-14.80		-15.03	-14.80	-14.86	-15.07
MGS	-15.14	-15.13		-15.03	-15.13	-15.22	-15.07
MGS-SVL	-15.18	-15.13		-15.03	-15.17	-15.20	-15.07
MGS-CWY	-14.80	-14.87		-15.03	-14.86	-14.80	-15.07
HouseQR	-15.01	-15.03		-15.03	-14.93	-14.93	-15.03
CholQR	-15.02	-15.12		-15.03	-15.02	-15.02	-15.07
CholQR+	-15.02	-15.03		-15.03	-15.02	-15.09	-15.07
ShCholQR++	-15.08	-15.12		-15.03	-15.01	-15.01	-15.07

Relative Residual, log10-scale

	BCGS	BCGSI+	BCGSS+R	BCGSI+LS	BMGS	BMGS-SVL	BMGS-CWY
CGS	-15.88	-15.77		-15.72	-15.49	-15.88	-15.78
CGSI+	-15.86	-15.74		-15.72	-15.50	-15.85	-15.78
CGSS+	-15.83	-15.74	-15.83	-15.72	-15.49	-15.83	-15.78
CGSS+R	-15.83	-15.74	-15.83	-15.72	-15.49	-15.83	-15.78
CGSI+LS	-15.88	-15.74		-15.72	-15.50	-15.88	-15.78
MGS	-15.77	-15.71		-15.72	-15.47	-15.77	-15.78
MGS-SVL	-15.89	-15.77		-15.72	-15.49	-15.88	-15.78
MGS-CWY	-15.88	-15.75		-15.72	-15.49	-15.88	-15.78
HouseQR	-15.24	-15.15		-15.72	-15.22	-15.25	-15.66
CholQR	-15.77	-15.66		-15.72	-15.48	-15.77	-15.78
CholQR+	-15.70	-15.62		-15.72	-15.47	-15.70	-15.78
ShCholQR++	-15.51	-15.27		-15.72	-15.39	-15.53	-15.62

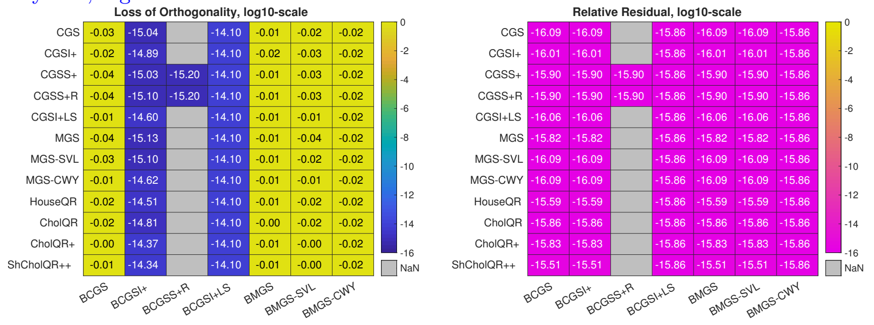


Figure 3: Measurements for the `rank_def` matrix. Only skeletons which use reorthogonalization maintain a reasonable loss of orthogonality; for other skeletons, orthogonality is completely lost, regardless of `IntraOrtho`.

Loss of Orthogonality, log10-scale

	BCGS	BCGSI+	BCGSS+R	BCGSI+LS	BMGS	BMGS-SVL	BMGS-CWY
CGS	-0.03	-15.04		-14.10	-0.01	-0.02	-0.02
CGSI+	-0.02	-14.89		-14.10	-0.02	-0.03	-0.02
CGSS+	-0.04	-15.03	-15.20	-14.10	-0.01	-0.03	-0.02
CGSS+R	-0.04	-15.10	-15.20	-14.10	-0.01	-0.03	-0.02
CGSI+LS	-0.01	-14.60		-14.10	-0.01	-0.01	-0.02
MGS	-0.04	-15.13		-14.10	-0.01	-0.04	-0.02
MGS-SVL	-0.03	-15.10		-14.10	-0.01	-0.02	-0.02
MGS-CWY	-0.01	-14.62		-14.10	-0.01	-0.01	-0.02
HouseQR	-0.02	-14.51		-14.10	-0.01	-0.02	-0.02
CholQR	-0.02	-14.81		-14.10	-0.00	-0.02	-0.02
CholQR+	-0.00	-14.37		-14.10	-0.01	-0.00	-0.02
ShCholQR++	-0.01	-14.34		-14.10	-0.01	-0.00	-0.02

Relative Residual, log10-scale

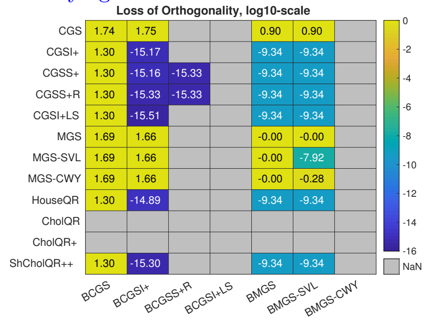
	BCGS	BCGSI+	BCGSS+R	BCGSI+LS	BMGS	BMGS-SVL	BMGS-CWY
CGS	-16.09	-16.09		-15.86	-16.09	-16.09	-15.86
CGSI+	-16.01	-16.01		-15.86	-16.01	-16.01	-15.86
CGSS+	-15.90	-15.90	-15.90	-15.86	-15.90	-15.90	-15.86
CGSS+R	-15.90	-15.90	-15.90	-15.86	-15.90	-15.90	-15.86
CGSI+LS	-16.06	-16.06		-15.86	-16.06	-16.06	-15.86
MGS	-15.82	-15.82		-15.86	-15.82	-15.82	-15.86
MGS-SVL	-16.09	-16.09		-15.86	-16.09	-16.09	-15.86
MGS-CWY	-16.09	-16.09		-15.86	-16.09	-16.09	-15.86
HouseQR	-15.59	-15.59		-15.86	-15.59	-15.59	-15.86
CholQR	-15.86	-15.86		-15.86	-15.86	-15.86	-15.86
CholQR+	-15.83	-15.83		-15.86	-15.83	-15.83	-15.86
ShCholQR++	-15.51	-15.51		-15.86	-15.51	-15.51	-15.86

3.3. `laeuchli`

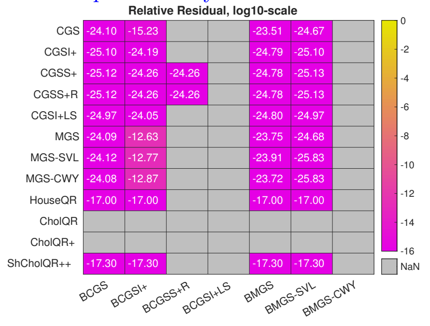
The Läuchli matrix reveals many interesting nuances. We first note in Figure 4 that for all algorithms that run to completion, except `BCGSI+` with `CGS`, `MGS`, `MGS-SVL`, and `MGS-CWY`, the relative residual exceeds double precision. This is due to the nature of the Läuchli matrix itself: because entries are either zero or close to ε, there is a high rate of cancellation. Incidentally, this same cancellation is the reason that neither `CholQR` nor `CholQR+` are viable `IntraOrthos`, and `BCGSI+LS` and `BMGS-CWY` are not viable skeletons: $\boldsymbol{X}_1^T \boldsymbol{X}_1$ is within $\mathcal{O}(\varepsilon)$ of a matrix of all ones, which is not strictly positive definite and therefore violates the requirements in MATLAB's `chol`.

The other interesting point relates to the correction matrix T of `MGS-SVL`. Both `BCGSI+` and `BMGS` suffer a total loss of orthogonality with `MGS-SVL`, but `BMGS-SVL` does not, since the `BMGS-SVL` skeleton was designed to incorporate this T matrix from the `MGS-SVL` muscle. There is an easy fix for `BCGSI+` and `BMGS`: we can incorporate the T output of `MGS-SVL` into both algorithms by replacing every action of $\boldsymbol{Q}_k$ with $\boldsymbol{Q}_k T_{kk}$. The second loss of orthogonality table in Figure 4 demonstrates the outcome, with a "T" suffix denoting the altered algorithms. With the T-fix, both `BCGSI+T`$\circ$`MGS-SVL` and `BMGS-T`$\circ$`MGS-SVL` perform as well as `BMGS-SVL` $\circ$ `MGS-SVL`; in fact, `BCGSI+T` $\circ$ `MGS-SVL` matches the others in residual too. (Note that `BCGS` with the T-fix would reproduce `BMGS-SVL`.) We note also that, even though `MGS-SVL` and `MGS-CWY` are based on the same projector, `BMGS-SVL` does not work with the muscle `MGS-CWY`; orthogonality is completely lost.

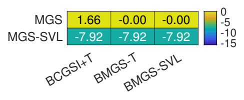
Figure 4: Measurements for the `laeuchli` matrix. Due to the high rate of cancellation in this matrix, skeletons and muscles that rely on MATLAB's `chol` fail. The bottom-most heatmap shows that the loss of orthogonality can be improved in `BCGSI+` and `BMGS` coupled with `MGS-SVL` by modifying the skeletons to make use of the T matrix produced by `MGS-SVL`.



	BCGS	BCGSI+	BCGSS+R	BCGSI+LS	BMGS	BMGS-SVL	BMGS-CWY
CGS	1.74	1.75			0.90	0.90	
CGSI+	1.30	-15.17			-9.34	-9.34	
CGSS+	1.30	-15.16	-15.33		-9.34	-9.34	
CGSS+R	1.30	-15.33	-15.33		-9.34	-9.34	
CGSI+LS	1.30	-15.51			-9.34	-9.34	
MGS	1.69	1.66			-0.00	-0.00	
MGS-SVL	1.69	1.66			-0.00	-7.92	
MGS-CWY	1.69	1.66			-0.00	-0.28	
HouseQR	1.30	-14.89			-9.34	-9.34	
CholQR							
CholQR+							
ShCholQR++	1.30	-15.30			-9.34	-9.34	



	BCGS	BCGSI+	BCGSS+R	BCGSI+LS	BMGS	BMGS-SVL	BMGS-CWY
CGS	-24.10	-15.23			-23.51	-24.67	
CGSI+	-25.10	-24.19			-24.79	-25.10	
CGSS+	-25.12	-24.26	-24.26		-24.78	-25.13	
CGSS+R	-25.12	-24.26	-24.26		-24.78	-25.13	
CGSI+LS	-24.97	-24.05			-24.80	-24.97	
MGS	-24.09	-12.63			-23.75	-24.68	
MGS-SVL	-24.12	-12.77			-23.91	-25.83	
MGS-CWY	-24.08	-12.87			-23.72	-25.83	
HouseQR	-17.00	-17.00			-17.00	-17.00	
CholQR							
CholQR+							
ShCholQR++	-17.30	-17.30			-17.30	-17.30	



	BCGSI+T	BMGS-T	BMGS-SVL
MGS	1.66	-0.00	-0.00
MGS-SVL	-7.92	-7.92	-7.92

3.4. `monomial`

The `monomial` matrix also reveals a few interesting properties. In Figure 5 we now see that all Cholesky-based methods can run, but they all lose quite a bit of orthogonality. Also, `CGS` in particular struggles across the board. We again see that `BCGSI+` $\circ$ `MGS-SVL` (and also `BCGSI+` $\circ$ `MGS-CWY`) does not perform as well as other variants of `BCGSI+`; however, the T-fix will not provide any benefit here, because `BCGSI+` $\circ$ `MGS-SVL` is already as stable as `BMGS-SVL` $\circ$ `MGS-SVL`.

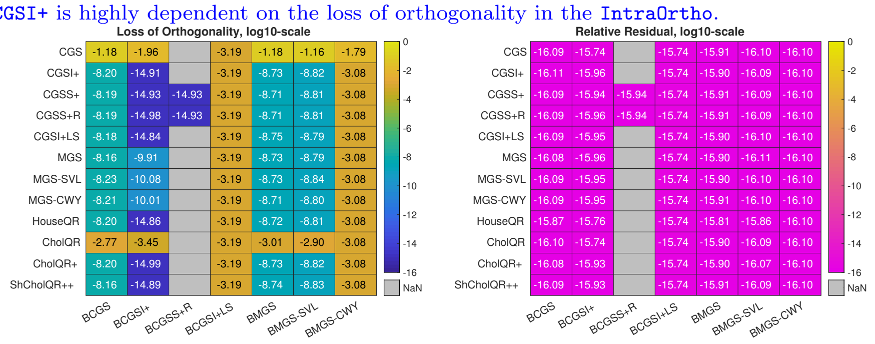


Figure 5: Measurements for the `monomial` matrix. For this problem, we see that behavior of a skeleton can vary based on the `IntraOrtho` used. In particular, the loss of orthogonality in `BCGSI+` is highly dependent on the loss of orthogonality in the `IntraOrtho`.

Loss of Orthogonality, log10-scale

	BCGS	BCGSI+	BCGSS+R	BCGSI+LS	BMGS	BMGS-SVL	BMGS-CWY
CGS	-1.18	-1.96		-3.19	-1.18	-1.16	-1.79
CGSI+	-8.20	-14.91		-3.19	-8.73	-8.82	-3.08
CGSS+	-8.19	-14.93	-14.93	-3.19	-8.71	-8.81	-3.08
CGSS+R	-8.19	-14.98	-14.93	-3.19	-8.71	-8.81	-3.08
CGSI+LS	-8.18	-14.84		-3.19	-8.75	-8.79	-3.08
MGS	-8.16	-9.91		-3.19	-8.73	-8.79	-3.08
MGS-SVL	-8.23	-10.08		-3.19	-8.73	-8.84	-3.08
MGS-CWY	-8.21	-10.01		-3.19	-8.71	-8.80	-3.08
HouseQR	-8.20	-14.86		-3.19	-8.72	-8.81	-3.08
CholQR	-2.77	-3.45		-3.19	-3.01	-2.90	-3.08
CholQR+	-8.20	-14.99		-3.19	-8.73	-8.82	-3.08
ShCholQR++	-8.16	-14.89		-3.19	-8.74	-8.83	-3.08

Relative Residual, log10-scale

	BCGS	BCGSI+	BCGSS+R	BCGSI+LS	BMGS	BMGS-SVL	BMGS-CWY
CGS	-16.09	-15.74		-15.74	-15.91	-16.10	-16.10
CGSI+	-16.11	-15.96		-15.74	-15.90	-16.09	-16.10
CGSS+	-16.09	-15.94	-15.94	-15.74	-15.91	-16.09	-16.10
CGSS+R	-16.09	-15.96	-15.94	-15.74	-15.91	-16.09	-16.10
CGSI+LS	-16.09	-15.95		-15.74	-15.90	-16.10	-16.10
MGS	-16.08	-15.96		-15.74	-15.90	-16.11	-16.10
MGS-SVL	-16.09	-15.95		-15.74	-15.90	-16.10	-16.10
MGS-CWY	-16.09	-15.95		-15.74	-15.91	-16.10	-16.10
HouseQR	-15.87	-15.76		-15.74	-15.81	-15.86	-16.10
CholQR	-16.10	-15.74		-15.74	-15.90	-16.09	-16.10
CholQR+	-16.08	-15.93		-15.74	-15.90	-16.07	-16.10
ShCholQR++	-16.09	-15.93		-15.74	-15.91	-16.09	-16.10

3.5. `s-step` *and* `newton`

The `s-step` and `newton` matrices simulate behavior one would encounter when implementing s-step Krylov subspace methods; see, e.g., [5, 50, 24]. The `s-step` matrix is poorly conditioned, as expected, because a monomial is used to build the basis; for these particular parameters and test matrix it is even numerically singular. The `newton` matrix simulates what would happen when Newton polynomials are used instead, which leads to much better conditioning. It is interesting that only `BCGSS+rpl` is capable of factoring the `s-step` matrix; not even `BCGSI+` $\circ$ `HouseQR` is capable. One the other hand, both one-sync methods (`BCGSI+LS` and `BMGS-CWY`) factor `newton` satisfactorily, almost to full precision.

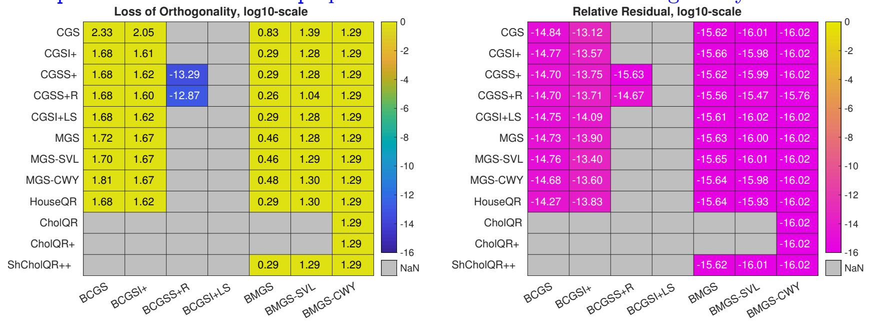
3.6. `stewart`

Stewart [36] purposefully designed his algorithms to be robust for pathologically bad matrices, and the results in Figure 8 demonstrate the failure of most algorithms for such matrices. Because of the identically zero column, nearly all methods encounter a `NaN` at some point after division of 0 by 0, which then spoils the rest of the algorithm. The only methods robust enough for such situations is Stewart's own algorithm, `BCGSS+rpl` $\circ$ `CGSS+rpl`, its variant without replacement, and `BCGSI+` with Stewart's `IntraOrtho`s and `HouseQR`.

3.7. `stewart_extreme`

This matrix in our collection demonstrates the utility of aggressive replacement. Figure 9 collects the results for `stewart_extreme`.

Figure 6: Measurements for the **s-step** matrix. For this numerically singular matrix, only BCGSS+rpl with CGSS+ or CGSS+rpl provides a reasonable loss of orthogonality.



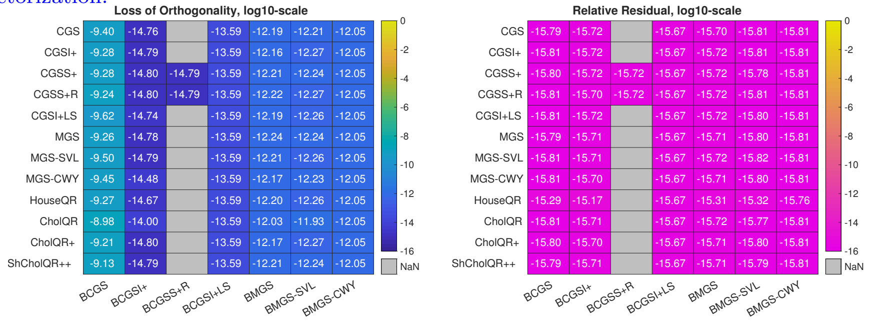
Loss of Orthogonality, log10-scale

	BCGS	BCGSI+	BCGSS+R	BCGSI+LS	BMGS	BMGS-SVL	BMGS-CWY
CGS	2.33	2.05			0.83	1.39	1.29
CGSI+	1.68	1.61			0.29	1.28	1.29
CGSS+	1.68	1.62	-13.29		0.29	1.28	1.29
CGSS+R	1.68	1.60	-12.87		0.26	1.04	1.29
CGSI+LS	1.68	1.62			0.29	1.28	1.29
MGS	1.72	1.67			0.46	1.28	1.29
MGS-SVL	1.70	1.67			0.46	1.29	1.29
MGS-CWY	1.81	1.67			0.48	1.30	1.29
HouseQR	1.68	1.62			0.29	1.30	1.29
CholQR							1.29
CholQR+							1.29
ShCholQR++					0.29	1.29	1.29

Relative Residual, log10-scale

	BCGS	BCGSI+	BCGSS+R	BCGSI+LS	BMGS	BMGS-SVL	BMGS-CWY
CGS	-14.84	-13.12			-15.62	-16.01	-16.02
CGSI+	-14.77	-13.57			-15.66	-15.98	-16.02
CGSS+	-14.70	-13.75	-15.63		-15.62	-15.99	-16.02
CGSS+R	-14.70	-13.71	-14.67		-15.56	-15.47	-15.76
CGSI+LS	-14.75	-14.09			-15.61	-16.02	-16.02
MGS	-14.73	-13.90			-15.63	-16.00	-16.02
MGS-SVL	-14.76	-13.40			-15.65	-16.01	-16.02
MGS-CWY	-14.68	-13.60			-15.64	-15.98	-16.02
HouseQR	-14.27	-13.83			-15.64	-15.93	-16.02
CholQR							-16.02
CholQR+							-16.02
ShCholQR++					-15.62	-16.01	-16.02

Figure 7: Measurements for the **newton** matrix. Because the use of Newton polynomials results in a more well-conditioned matrix, all algorithms are able to produce a satisfactory factorization.



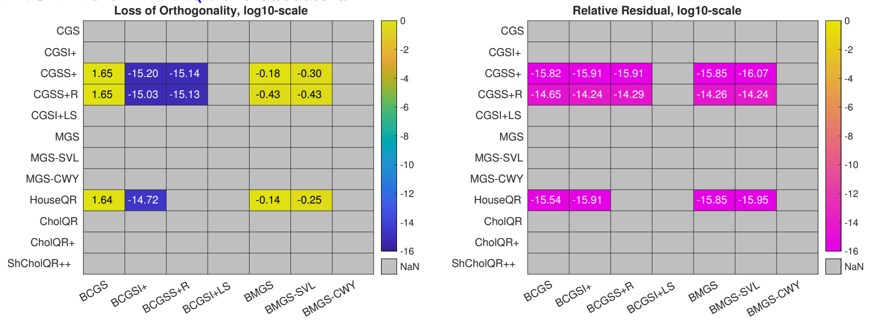
Loss of Orthogonality, log10-scale

	BCGS	BCGSI+	BCGSS+R	BCGSI+LS	BMGS	BMGS-SVL	BMGS-CWY
CGS	-9.40	-14.76		-13.59	-12.19	-12.21	-12.05
CGSI+	-9.28	-14.79		-13.59	-12.16	-12.27	-12.05
CGSS+	-9.28	-14.80	-14.79	-13.59	-12.21	-12.24	-12.05
CGSS+R	-9.24	-14.80	-14.79	-13.59	-12.22	-12.27	-12.05
CGSI+LS	-9.62	-14.74		-13.59	-12.19	-12.26	-12.05
MGS	-9.26	-14.78		-13.59	-12.24	-12.24	-12.05
MGS-SVL	-9.50	-14.79		-13.59	-12.21	-12.26	-12.05
MGS-CWY	-9.45	-14.48		-13.59	-12.17	-12.23	-12.05
HouseQR	-9.27	-14.67		-13.59	-12.20	-12.26	-12.05
CholQR	-8.98	-14.00		-13.59	-12.03	-11.93	-12.05
CholQR+	-9.21	-14.80		-13.59	-12.17	-12.27	-12.05
ShCholQR++	-9.13	-14.79		-13.59	-12.21	-12.24	-12.05

Relative Residual, log10-scale

	BCGS	BCGSI+	BCGSS+R	BCGSI+LS	BMGS	BMGS-SVL	BMGS-CWY
CGS	-15.79	-15.72		-15.67	-15.70	-15.81	-15.81
CGSI+	-15.81	-15.72		-15.67	-15.72	-15.81	-15.81
CGSS+	-15.80	-15.72	-15.72	-15.67	-15.72	-15.78	-15.81
CGSS+R	-15.81	-15.70	-15.72	-15.67	-15.72	-15.81	-15.81
CGSI+LS	-15.81	-15.72		-15.67	-15.72	-15.80	-15.81
MGS	-15.79	-15.71		-15.67	-15.71	-15.80	-15.81
MGS-SVL	-15.81	-15.71		-15.67	-15.72	-15.82	-15.81
MGS-CWY	-15.81	-15.70		-15.67	-15.71	-15.80	-15.81
HouseQR	-15.29	-15.17		-15.67	-15.31	-15.32	-15.76
CholQR	-15.81	-15.71		-15.67	-15.72	-15.77	-15.81
CholQR+	-15.80	-15.70		-15.67	-15.71	-15.80	-15.81
ShCholQR++	-15.79	-15.71		-15.67	-15.71	-15.79	-15.81

Figure 8: Measurements for the **stewart** matrix. Because this matrix has an identically zero column, most algorithms fail. Only Stewart's BCGSS+rpl ∘ CGSS+rpl and BCGSI+ with Stewart's IntraOrthos or HouseQR are successful.



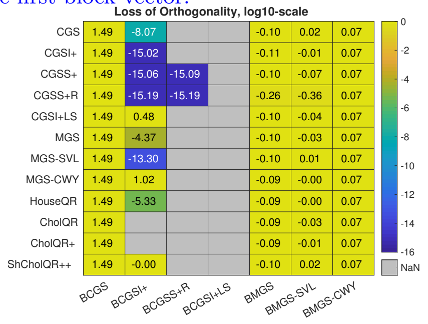
Loss of Orthogonality, log10-scale

	BCGS	BCGSI+	BCGSS+R	BCGSI+LS	BMGS	BMGS-SVL	BMGS-CWY
CGS							
CGSI+							
CGSS+	1.65	-15.20	-15.14		-0.18	-0.30	
CGSS+R	1.65	-15.03	-15.13		-0.43	-0.43	
CGSI+LS							
MGS							
MGS-SVL							
MGS-CWY							
HouseQR	1.64	-14.72			-0.14	-0.25	
CholQR							
CholQR+							
ShCholQR++							

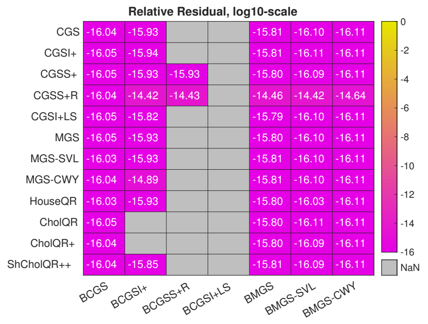
Relative Residual, log10-scale

	BCGS	BCGSI+	BCGSS+R	BCGSI+LS	BMGS	BMGS-SVL	BMGS-CWY
CGS							
CGSI+							
CGSS+	-15.82	-15.91	-15.91		-15.85	-16.07	
CGSS+R	-14.65	-14.24	-14.29		-14.26	-14.24	
CGSI+LS							
MGS							
MGS-SVL							
MGS-CWY							
HouseQR	-15.54	-15.91			-15.85	-15.95	
CholQR							
CholQR+							
ShCholQR++							

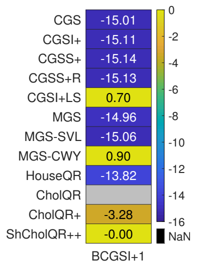
Figure 9: Measurements for the `stewart_extreme` matrix. Notice that the loss of orthogonality in `BCGSI+` is highly dependent on the `IntraOrtho` used. The bottom heatplot demonstrates that the behavior of `BCGSI+` with particular `IntraOrtho`s can be improved by reorthogonalizing the first block vector.



	BCGS	BCGSI+	BCGSS+R	BCGSI+LS	BMGS	BMGS-SVL	BMGS-CWY
CGS	1.49	-8.07	▨	▨	-0.10	0.02	0.07
CGSI+	1.49	-15.02	▨	▨	-0.11	-0.01	0.07
CGSS+	1.49	-15.06	-15.09	▨	-0.10	-0.07	0.07
CGSS+R	1.49	-15.19	-15.19	▨	-0.26	-0.36	0.07
CGSI+LS	1.49	0.48	▨	▨	-0.10	-0.04	0.07
MGS	1.49	-4.37	▨	▨	-0.10	-0.03	0.07
MGS-SVL	1.49	-13.30	▨	▨	-0.10	0.01	0.07
MGS-CWY	1.49	1.02	▨	▨	-0.09	-0.00	0.07
HouseQR	1.49	-5.33	▨	▨	-0.09	-0.00	0.07
CholQR	1.49	▨	▨	▨	-0.09	-0.03	0.07
CholQR+	1.49	▨	▨	▨	-0.09	-0.01	0.07
ShCholQR++	1.49	-0.00	▨	▨	-0.10	0.02	0.07



	BCGS	BCGSI+	BCGSS+R	BCGSI+LS	BMGS	BMGS-SVL	BMGS-CWY
CGS	-16.04	-15.93	▨	▨	-15.81	-16.10	-16.11
CGSI+	-16.05	-15.94	▨	▨	-15.81	-16.11	-16.11
CGSS+	-16.05	-15.93	-15.93	▨	-15.80	-16.09	-16.11
CGSS+R	-16.04	-14.42	-14.43	▨	-14.46	-14.42	-14.64
CGSI+LS	-16.05	-15.82	▨	▨	-15.79	-16.10	-16.11
MGS	-16.05	-15.93	▨	▨	-15.80	-16.10	-16.11
MGS-SVL	-16.03	-15.93	▨	▨	-15.81	-16.10	-16.11
MGS-CWY	-16.04	-14.89	▨	▨	-15.81	-16.10	-16.11
HouseQR	-16.03	-15.93	▨	▨	-15.80	-16.03	-16.11
CholQR	-16.05	▨	▨	▨	-15.80	-16.11	-16.11
CholQR+	-16.04	▨	▨	▨	-15.80	-16.09	-16.11
ShCholQR++	-16.04	-15.85	▨	▨	-15.81	-16.09	-16.11



	BCGSI+1
CGS	-15.01
CGSI+	-15.11
CGSS+	-15.14
CGSS+R	-15.13
CGSI+LS	0.70
MGS	-14.96
MGS-SVL	-15.06
MGS-CWY	0.90
HouseQR	-13.82
CholQR	▨
CholQR+	-3.28
ShCholQR++	-0.00

For both `BCGSI+` and `BCGSS+rpl` we can see how aggressive replacement can make a significant difference. The only algorithms capable of achieving $\mathcal{O}(\varepsilon)$ loss of orthogonality in this case are `BCGSS+rpl` with `CGSS+` and `CGSS+rpl`, and `BCGSI+` with `CGSS+`, `CGSS+rpl`, and `CGSI+`. It is unclear why for this particular case `BCGSI+` $\circ$ `CGSI+` exhibits better orthogonality than `BCGSI+` $\circ$ `HouseQR`.

Looking closely at Algorithm 4, we see that `BCGSI+` does not reorthogonalize the first block vector in line 2, likely because this is not done for the column-wise algorithm, where it is unnecessary. By simply rerunning `IntraOrtho` for the first block vector (denoted "BCGSI+1" in Figure 9), we can reduce the loss of orthogonality to $\mathcal{O}(\varepsilon)$ for some muscles. Notably both the one-sync methods– `CGSI+LS` and `MGS-CWY`– still struggle, and `CholQR` still cannot run to completion.

This example highlights a problem that remains unaddressed in [13] for `ShCholQR++`: when $\boldsymbol{X}$ contains a column very close to the zero vector. In this situation, the bounds for the shift σ may be too stringent to allow $\boldsymbol{X}^T\boldsymbol{X} + \sigma I_s$ to be numerically positive definite. In the `stewart_extreme` example, setting $\sigma = \|\boldsymbol{X}\|_2^2$ allows `BCGSI+` $\circ$ `ShCholQR++` to run to completion with $\mathcal{O}\left(10^{-12}\right)$ loss of orthogonality and $\mathcal{O}\left(10^{-17}\right)$ residual. It seems that the upper bound $\sigma \leq \frac{1}{100}\|\boldsymbol{X}\|_2^2$ is more likely an artifact of the analysis and that a larger bound may in fact be tolerable. To be fully robust, `ShCholQR++` should be written to allow for an automatically tuned bound based on whether the shifted Gramian is found to be positive definite enough for `chol` to run. This remains a topic of future work.

We note that this trick of reorthogonalizing the first block vector has even better results for other test problems we tried. For both `monomial` and `laeuchli`, if we reorthogonalize the first block vector then `BCGSI+` with any of the tested muscles provides $\mathcal{O}(\varepsilon)$ loss of orthogonality. We do not display these tests here, but see a related discussion in Section 4.2.

Overall, the experiments in this section give indication that different skeletons have very different requirements on the muscle. For `BCGS`, the choice of muscle makes little difference. This is because `BCGS` is itself unstable as a skeleton, so there is no benefit to using, say, `HouseQR` instead of `CGS` (as long as the constraints on condition number are satisfied to make the muscle viable). In constrast, the reorthogonalization in `BCGSI+` stablizes the skeleton, and thus the quality of the `IntraOrtho` will have a greater affect on the resulting stability. This is also true for `BMGS` and variants like `BMGS-SVL`. We elaborate on these details in the following section.

4. Skeleton stability in terms of muscle stability: details

We now consider the stability of each skeleton in more detail. We utilize a common plot format, wherein stability quantities like loss of orthogonality or relative residual are plotted against varying condition numbers. We refer to such plots as *κ-plots*, where κ refers to the fact that we study stability with respect to changes in condition number. The simplest version of these plots takes a series of matrices $\boldsymbol{\mathcal{X}}_t = \boldsymbol{\mathcal{U}}\Sigma_t\mathcal{V}^T \in \mathbb{R}^{m\times ps}$, where $\boldsymbol{\mathcal{U}} \in \mathbb{R}^{m\times ps}$ is orthonormal, $\Sigma_t \in \mathbb{R}^{ps\times ps}$ is a diagonal matrix whose entries are drawn from the logarithmic interval $10^{[-t,0]}$, and $\mathcal{V} \in \mathbb{R}^{ps\times ps}$ is unitary. We also consider **`glued`** κ-plots, where the matrices are instead built as the "glued" matrices from [28]; see Appendix C for how they are generated. The `monomial` matrices from Section 3 also lend themselves to κ-plots, because varying condition numbers can be induced by varying block sizes. All κ-plots are run in MATLAB 9.8.0.1451342 (R2020a) Update 5. We provide the command-line calls for each figure in Appendix C.

4.1. BCGS

Using similar techniques as for `CGS`, we conjecture that the loss of orthogonality in `BCGS` $\circ$ `IntraOrtho` can be bounded as

$$\left\| I - \bar{\boldsymbol{\mathcal{Q}}}^T \bar{\boldsymbol{\mathcal{Q}}} \right\|_2 \leq \mathcal{O}(\varepsilon)\, \kappa^{n-1}(\boldsymbol{\mathcal{X}})$$

as long as $\mathcal{O}(\varepsilon)\,\kappa(\boldsymbol{\mathcal{X}}) < 1$. A full proof that the `BCGS` P-variants have $\mathcal{O}(\varepsilon)\,\kappa^2(\boldsymbol{\mathcal{X}})$ loss of orthogonality and $\mathcal{O}(\varepsilon)$ relative Cholesky residual as long as $\mathcal{O}(\varepsilon)\,\kappa(\boldsymbol{\mathcal{X}}) < 1/2$ is given in [29]. These bounds hold as long as `IntraOrtho`$(\boldsymbol{X})$ satisfies

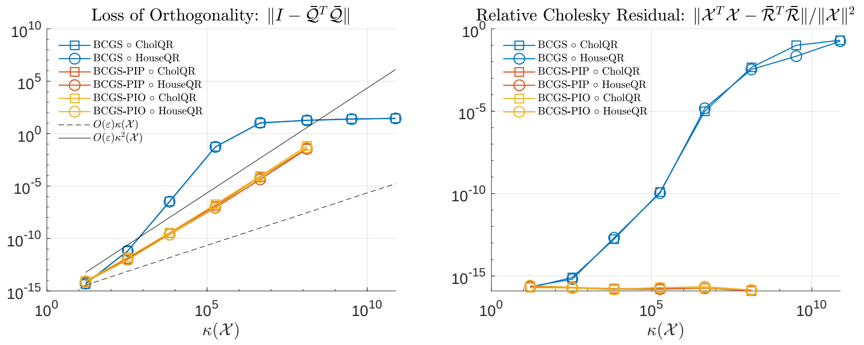
$$\bar{R}^T\bar{R} = \boldsymbol{X}^T\boldsymbol{X} + E, \quad \|E\| \leq \mathcal{O}(\varepsilon)\,\|\boldsymbol{X}\|^2, \tag{10}$$

and

$$\bar{\boldsymbol{Q}}\bar{R} = \boldsymbol{X} + \boldsymbol{D}, \quad \|\boldsymbol{D}\| \leq \mathcal{O}(\varepsilon)\left(\|\boldsymbol{X}\| + \left\|\bar{\boldsymbol{Q}}\right\| \left\|\bar{R}\right\|\right). \tag{11}$$

We note that these constraints on the IntraOrtho are relatively relaxed, and are satisfied by almost any reasonable choice of IntraOrtho, from CholQR to HouseQR. Indeed, in Figure 10, we see that both BCGS-PIP and BCGS-PIO (as well as BCGS) behave similarly for both CholQR and HouseQR. We also note that it is clear that BCGS (without the Pythagorean fix) does not satisfy the $\mathcal{O}(\varepsilon)\kappa^2\boldsymbol{\mathcal{X}}$ bound on loss of orthogonality; as the relative Cholesky residual departs from $\mathcal{O}(\varepsilon)$, the loss of orthogonality follows suit.

Figure 10: glued κ-plots for P-variants of BCGS. The loss of orthogonality for BCGS exceeds $O(\varepsilon)\kappa^2(\boldsymbol{\mathcal{X}})$, but is bounded by $O(\varepsilon)\kappa^2(\boldsymbol{\mathcal{X}})$ for the P-variants, which have an $O(\varepsilon)$ relative Cholesky residual. Notice that the skeletons behave similarly regardless of whether CholQR or HouseQR are used as the IntraOrtho.



4.2. BCGSI+

Barlow and Smoktunowicz prove an $\mathcal{O}(\varepsilon)$ loss of orthogonality for BCGSI+ and an unconditionally stable IntraOrtho, as long as $\mathcal{O}(\varepsilon)\kappa(\boldsymbol{\mathcal{X}}) < 1$ [16]. The key to their approach is to first obtain bounds for the following subproblem: given a near left-orthogonal matrix $\boldsymbol{\mathcal{U}} \in \mathbb{R}^{m\times t}$ and a matrix $\boldsymbol{B} \in \mathbb{R}^{m\times s}$, find an $\boldsymbol{S} \in \mathbb{R}^{t\times s}$, $R \in \mathbb{R}^{s\times s}$ upper triangular, and $\boldsymbol{Q}$ left orthogonal such that

$$\boldsymbol{B} = \boldsymbol{\mathcal{U}}\boldsymbol{S}_{\boldsymbol{B}} + \boldsymbol{Q}_{\boldsymbol{B}}R_{\boldsymbol{B}} \text{ and } \boldsymbol{\mathcal{U}}^T\boldsymbol{Q}_{\boldsymbol{B}} \approx 0. \tag{12}$$

This subproblem is present at every step of BCGSI+, as long as the previous step has produced a near-left orthogonal matrix $\boldsymbol{\mathcal{U}}$. With the unconditionally stable IntraOrtho guaranteeing that $\boldsymbol{Q}_1$ (see line 2 in Algorithm 4) is near left-orthogonal, induction over bounds on the subproblem leads to the desired result.

One potential drawback to Barlow and Smoktunowicz's approach is the *a posteriori* assumption that $\mathcal{O}(\varepsilon)\|\boldsymbol{B}\|\left\|\bar{R}_{\boldsymbol{B}}^{-1}\right\| < 1$, where $\bar{R}_{\boldsymbol{B}}$ is the computed version of $R_{\boldsymbol{B}}$ from the subproblem (12); see [16, Equation (3.27)]. In addition to simplifying the proof, the authors argue that such an assumption is useful in practice, because the computed quantities can be measured on the fly.

Figure 11 displays standard κ-plots for `BCGSI+` in comparison to `BCGS` for muscles `CGS`, `MGS`, and `HouseQR`. It is tempting to conclude that the skeletons' loss of orthogonality is independent of the muscle, but recalling the results for `laeuchli` and `monomial` matrices in Sections 3.3 and 3.4 (both of which satisfy $\mathcal{O}(\varepsilon)\,\kappa(\boldsymbol{\mathcal{X}}) < 1$; cf. Table 5), it is clear that we may need an unconditionally stable `IntraOrtho` for some tough matrices. Figure 12 shows κ-plots generated using `laeuchli` matrices. Indeed, here we see highlighted the different ways that `BCGS` and `BCGSI+` are affected by the `IntraOrtho`. Whereas `BCGS` behaves the same regardless of what `IntraOrtho` is used, `BCGSI+` is much more sensitive.

Figure 11: standard κ-plots for `BCGSI+`, in comparison to `BCGS`. The reorthogonalization in `BCGSI+` results in $O(\varepsilon)$ loss of orthogonality. Although the behavior of the skeleton seems independent of the muscle here, this is not true in general.

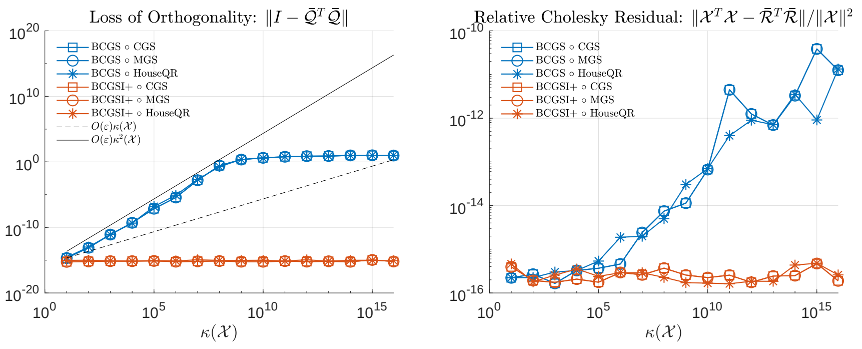
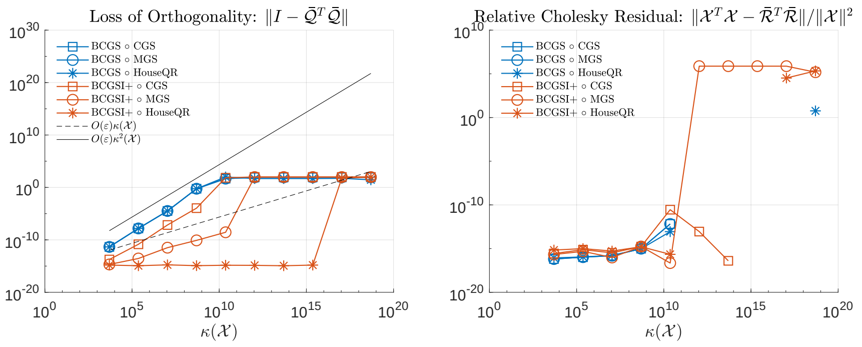


Figure 12: Laeuchli κ-plots for `BCGSI+`, in comparison to `BCGS`. These plots demonstrate that `BCGSI+` must have an unconditionally stable `IntraOrtho` in order to guarantee $O(\varepsilon)$ loss of orthogonality.



4.3. BCGSS+rpl

Although Stewart's development of `BCGSS+rpl` [36] is largely driven by stability considerations, a rigorous proof is lacking. We conjecture that for any $\boldsymbol{\mathcal{X}}$, even those with rank deficiency, we have $\mathcal{O}(\varepsilon)$ loss of orthogonality, up to a small constant accounting for the replacement tolerance δ.

This conjecture is based largely on numerical results from both Stewart's paper [36] and our own studies in Section 3, wherein a clear trade-off between loss of orthogonality and relative residual seems to be driven largely by δ. It is also clear that the orthogonalization fault step, designed precisely to avoid situations leading to instability, plays an important role. A successful stability analysis for `BCGSS+rpl` of course also depends on one for `CGSS+rpl`.

4.4. BCGSI+LS

Rigorous stability analysis for the low-sync algorithms from [25], especially their block versions, remains open. We can, however, formulate some conjectures.

In Figure 13, we see that indicate that `BCGSI+LS` remains relatively stable, especially its relative Cholesky residual. The loss of orthogonality for `BCGSI+LS` begins to deviate gradually from $\mathcal{O}(\varepsilon)$ in Figure 13. However, looking at the more challenging `laeuchli` matrix problems in Figure 14, we see that the loss of orthogonality can be much more significant, exceeding $\mathcal{O}(\varepsilon)\,\kappa(\boldsymbol{\mathcal{X}})$. Based on these results, we conjecture that `BCGSI+LS` satisfies the bound $\mathcal{O}(\varepsilon)\,\kappa^2(\boldsymbol{\mathcal{X}})$ as long as $\mathcal{O}(\varepsilon)\,\kappa^2(\boldsymbol{\mathcal{X}}) < 1$ (due to the use of `chol` within the algorithm). This conjecture is stated in Table 4; a formal proof remains future work. The norm lagging and delayed reorthogonalization seems to have a much more significant effect in `BCGSI+LS` versus `CGSI+LS`.

Figure 13: standard κ-plots for low-sync `BCGSI+LS`, in comparison to those of `BCGS` and `BCGSI+`. Here the loss of orthogonality in `BCGSI+LS` gradually deviates from $O(\varepsilon)$.

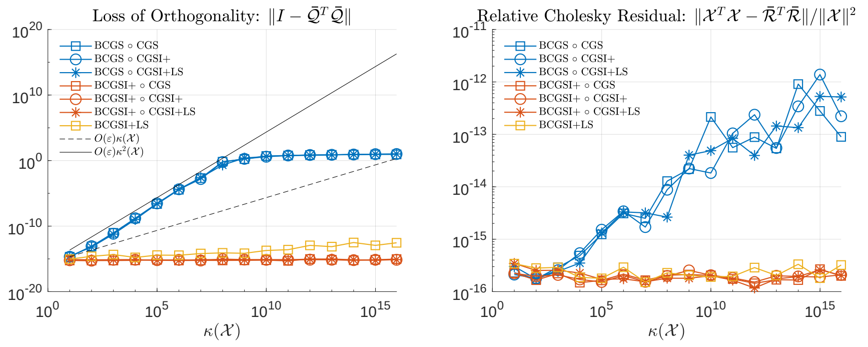
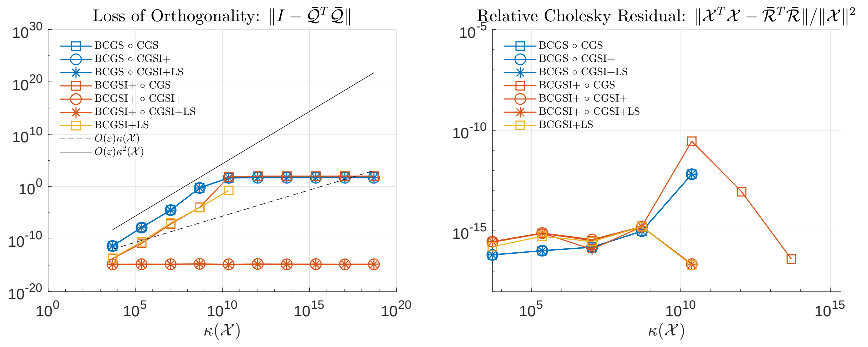


Figure 14: `laeuchli` κ-plots for low-sync `BCGSI+LS`, in comparison to those of `BCGS` and `BCGSI+`. These experiments demonstrate that, in contrast to `BCGSI+`, the one-sync variant `BCGSI+LS` can suffer steep loss of orthogonality, exceeding $O(\varepsilon)\kappa(\boldsymbol{\mathcal{X}})$.



4.5. BMGS

What appears to be the earliest study on the stability of a block Gram-Schmidt method was conducted by Jalby and Philippe in 1991 [18]. They focus on `BMGS` ∘ `MGS` and `BMGS` ∘ `MGS+`, and note how the underlying `CGS`-like nature of `BMGS` makes it prone to instability. To see this, observe that lines 4-8 of Algorithm 8 can be written more succinctly (in exact arithmetic) as

$$\boldsymbol{W} = (I - \boldsymbol{Q}_k\boldsymbol{Q}_k^T)\cdots(I - \boldsymbol{Q}_1\boldsymbol{Q}_1^T)\boldsymbol{X}_{k+1}. \tag{13}$$

Because each $\boldsymbol{Q}_j$, $j = 1, \ldots, k$, here is a tall-and-skinny matrix, the projector $I - \boldsymbol{Q}_j\boldsymbol{Q}_j^T$ is equivalent to a step of `CGS`. The authors demonstrate this intuition rigorously with `MGS` as the `IntraOrtho` to arrive at the following bound for the loss of orthogonality:

$$\left\|I - \bar{\boldsymbol{\mathcal{Q}}}^T\bar{\boldsymbol{\mathcal{Q}}}\right\|_2 \leq \mathcal{O}(\varepsilon)\, C\, \|\boldsymbol{\mathcal{X}}\|_{\mathrm{F}} \left\|\mathcal{R}^{-1}\right\|_2 \leq \mathcal{O}(\varepsilon)\,\kappa^2(\boldsymbol{\mathcal{X}}), \tag{14}$$

wherein the second inequality follows because $C \leq O(\kappa(\boldsymbol{\mathcal{X}}))$ and we have $\|\boldsymbol{\mathcal{X}}\|_{\mathrm{F}}\left\|\mathcal{R}^{-1}\right\|_2 \leq O(\kappa(\boldsymbol{\mathcal{X}}))$. The authors further show that the constant C becomes $O(1)$ if `MGS` is replaced by `MGS+`, thus giving the MGS-like bound

$$\left\|I - \bar{\boldsymbol{\mathcal{Q}}}^T\bar{\boldsymbol{\mathcal{Q}}}\right\|_2 \leq \mathcal{O}(\varepsilon)\,\kappa(\boldsymbol{\mathcal{X}}). \tag{15}$$

Barlow [15] conjectures that `BMGS∘HouseQR` is as stable as `MGS` but does not prove so explicitly. A careful look at Jalby and Philippe's proof for their Theorem 4.1 reveals that there is not much to prove; in fact, the very same observation they used to derive (15) holds for any `IntraOrtho` that is unconditionally stable.

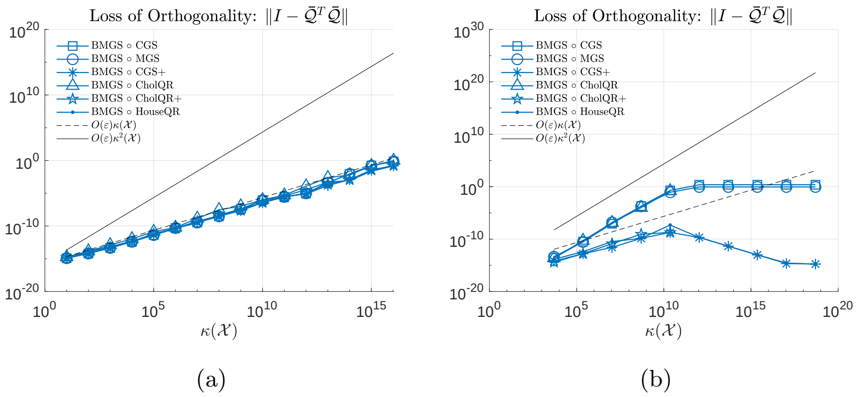
We sketch the proof of this here. At the top of page 1062 of [18], the Jalby and Philippe directly apply `MGS` bounds from [8] and define the constant $C = \max_{1\leq k\leq p-1}\|\boldsymbol{W}_k\|_{\mathrm{F}}\left\|R_{\boldsymbol{W}_k}^{-1}\right\|_2$, where $\boldsymbol{W}_k$ corresponds to the block vector computed up to line 8 of Algorithm 8, and $R_{\boldsymbol{W}_k}$ its upper triangular factor from an exact QR decomposition. The intermediate constants $\|\boldsymbol{W}_k\|_F \left\|R_{\boldsymbol{W}_k}^{-1}\right\|_2$ are absent for any unconditionally stable `IntraOrtho`. Tracking C all the way through the proof of [18, Theorem 4.1] (by which (14) holds) verifies that, without changing any other lines in their text, we would obtain (15) for `BMGS` $\circ$ `IntraOrtho`, assuming `IntraOrtho` is unconditionally stable.

We remark that, to our knowledge, the present work is the first to state bounds for `BMGS` with an unconditionally stable `IntraOrtho` (such as `HouseQR` or `TSQR`), despite the ubiquity of `BMGS` $\circ$ `HouseQR` in practice. More precise bounds for particular `IntraOrtho`s remain open, but can be easily derived on a case-by-case basis from the transparent analysis in [18].

Because `BMGS` $\circ$ `HouseQR` behaves essentially like `MGS`, we conjecture that results analogous to those by Paige, Rozložník, and Strakoš [11] hold for block GMRES, which is usually implemented with `BMGS`$\circ$`HouseQR`. To be more specific, the loss of orthogonality in `BMGS` $\circ$ `HouseQR` is tolerable when the end application is the solution of a linear system via the GMRES method.

Figure 15 demonstrates the `BMGS` behavior for a variety of muscles and their reorthogonalized variants again for both standard κ-plots (left) and `laeuchli` κ-plots (right). For the standard κ-plots, all variants behave the same, giving better orthogonality than indicated by (14). For the `laeuchli` κ-plots, we do observe behavior like (14) here for the `IntraOrtho`s that do not have $\mathcal{O}(\varepsilon)$ loss of orthogonality (`CGS`, `MGS`, and `CholQR`). Indeed, this example demonstrates that an unconditionally stable muscle is really necessary to guarantee $\mathcal{O}(\varepsilon)\,\kappa(\boldsymbol{\mathcal{X}})$ loss of orthogonality.

Figure 15: κ-plots for `BMGS`. (a) standard κ-plot. (b) `laeuchli` κ-plot. Whereas `BMGS` exhibits similar loss of orthogonality regardless of the muscle used for the standard κ-plot, the `laeuchli` κ-plot shows that an unconditionally stable muscle is necessary in order to guarantee $O(\varepsilon)\kappa(\boldsymbol{\mathcal{X}})$ loss of orthogonality in `BMGS`.



4.6. Low-sync BMGS *variants*

To date, we are only aware of Barlow's stability analysis of MGS-SVL and BMGS-SVL for low-synchronization BMGS variants [15]. Indeed, it appears that Barlow was even unaware that his algorithms could take on so many other forms, particularly the work by Świrydowicz et al. [25].

Barlow's analysis relies heavily on the Sheffield observation and Schreiber-van-Loan reformulation– what is sometimes also referred to as the "augmented" problem– for the stability analysis of BMGS-SVL; see, in particular, [15, Section 4]. The following quantities play key roles:

$$\begin{aligned}\Delta_{\mathcal{T}\mathcal{S}} &:= \bar{\mathcal{T}}\mathcal{S} - I,\\ \Delta_{\bar{\boldsymbol{\mathcal{Q}}}\bar{\mathcal{R}}} &:= \bar{\boldsymbol{\mathcal{Q}}}\bar{\mathcal{R}} - \boldsymbol{\mathcal{X}}, \text{ and}\\ \Gamma_{\mathcal{T}\mathcal{R}} &:= (I - \bar{\mathcal{T}})\bar{\mathcal{R}}.\end{aligned}$$

The matrix $\mathcal{S} \in \mathbb{R}^{n\times n}$ is an exact quantity that does not explicitly arise in the computation. Although it has several equivalent formulations in exact arithmetic (see especially page 1261 in [15]), it is apparently defined as the upper triangular part of the exact multiplication between $\bar{\boldsymbol{\mathcal{Q}}}^T$ and $\bar{\boldsymbol{\mathcal{Q}}}$, i.e.,

$$\mathcal{S} := \mathtt{triu}(\bar{\boldsymbol{\mathcal{Q}}}^T\bar{\boldsymbol{\mathcal{Q}}}).$$

In exact arithmetic, $\mathcal{S} = \mathcal{T}^{-1}$, hence why the measure $\Delta_{\mathcal{T}\mathcal{S}}$ bears significance.

For BMGS-SVL $\circ$ IntraOrtho, Barlow shows that

$$\begin{aligned}\|\Delta_{\mathcal{T}\mathcal{S}}\|_F &\leq \mathcal{O}(\varepsilon), & (16)\\ \|\Delta_{\bar{\boldsymbol{\mathcal{Q}}}\bar{\mathcal{R}}}\|_F &\leq \mathcal{O}(\varepsilon)\|\boldsymbol{\mathcal{X}}\|_F, \text{ and} & (17)\\ \|\Gamma_{\mathcal{T}\mathcal{R}}\|_F &\leq \mathcal{O}(\varepsilon)\|\boldsymbol{\mathcal{X}}\|_F, & (18)\end{aligned}$$

as long as $[\bar{\boldsymbol{Q}}, \bar{R}, \bar{T}] = \mathtt{IntraOrtho}(\boldsymbol{X})$ satisfies the triad

$$\begin{aligned}\Delta_{TS} &:= \bar{T}S - I_s, & \|\Delta_{TS}\|_F &\leq \mathcal{O}(\varepsilon); & (19)\\ \Delta_{\bar{\boldsymbol{Q}}\bar{R}} &:= \bar{\boldsymbol{Q}}\bar{R} - \boldsymbol{X}, & \left\|\Delta_{\bar{\boldsymbol{Q}}\bar{R}}\right\|_F &\leq \mathcal{O}(\varepsilon)\|\boldsymbol{X}\|_F; \text{ and} & (20)\\ \Gamma_{TR} &:= (I_s - \bar{T})\bar{R}, & \|\Gamma_{TR}\|_F &\leq \mathcal{O}(\varepsilon)\|\boldsymbol{X}\|_F & (21)\end{aligned}$$

where $S := \mathtt{triu}(\bar{\boldsymbol{Q}}^T\bar{\boldsymbol{Q}})$ and $S = T^{-1}$ in exact arithmetic.

Equation (17) already gives the usual residual bound, and Barlow additionally shows that it holds at every step of BMGS. To arrive at bounds for loss of orthogonality, he further shows that (16)-(18) imply

$$\left\|I - \bar{\boldsymbol{\mathcal{Q}}}^T\bar{\boldsymbol{\mathcal{Q}}}\right\|_F \leq \mathcal{O}(\varepsilon)\|\boldsymbol{\mathcal{X}}\|_F\left\|\mathcal{R}^{-1}\right\|,$$

Recalling (from, e.g., [8]) that $\|\boldsymbol{\mathcal{X}}\|_F\left\|\mathcal{R}^{-1}\right\| \leq O(\kappa(\boldsymbol{\mathcal{X}}))$, this latter condition can be satisfied only if $\boldsymbol{\mathcal{X}}$ is sufficiently far from a rank-deficient matrix, i.e., if $\mathcal{O}(\varepsilon)\kappa(\boldsymbol{\mathcal{X}}) < 1$. We then have that

$$\left\|I - \bar{\boldsymbol{\mathcal{Q}}}^T\bar{\boldsymbol{\mathcal{Q}}}\right\|_F \leq \mathcal{O}(\varepsilon)\kappa(\boldsymbol{\mathcal{X}}),$$

and thanks to norm equivalence, we obtain the bounds reported in Table 4.

At first glance through Appendix A, it may seem that the `MGS` low-sync variants are the only muscles designed to produce a T matrix. Upon further investigation of how Barlow uses the quantities Δ_{TS}, $\Delta_{\bar{\boldsymbol{Q}}\bar{R}}$, and Γ_{TR}, it becomes clear that actually all other `IntraOrtho`s implicitly produce $\bar{T} \equiv I$. This is important for composing `BMGS-SVL` and `BMGS-LTS` with other muscles, because they explicitly update the block diagonal entries of $\mathcal{T}$ with outputs from the `IntraOrtho`.

It then holds for these muscles that

$$\|\Delta_{TS}\|_{\mathrm{F}} \equiv \left\|\mathtt{triu}(I - \bar{\boldsymbol{Q}}^T\bar{\boldsymbol{Q}})\right\|_{\mathrm{F}} \leq \left\|I - \bar{\boldsymbol{Q}}^T\bar{\boldsymbol{Q}}\right\|_{\mathrm{F}},$$

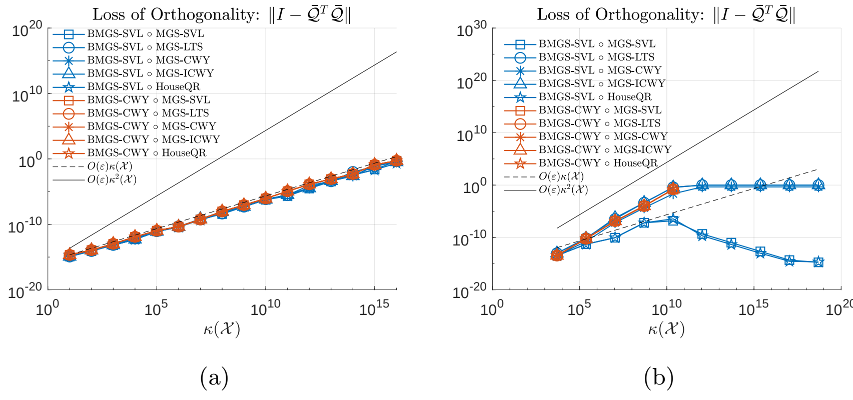
and

$$\|\Gamma_{TR}\|_{\mathrm{F}} = 0.$$

Is is thus clear that (19)-(21) are satisfied by any `IntraOrtho` with $\mathcal{O}(\varepsilon)$ loss of orthogonality and relative residual (such as `CGSI+`), and thus (16)-(18) hold. Note that although `MGS-SVL` does not have $\mathcal{O}(\varepsilon)$ loss of orthogonality, it still satisfies (19)-(21) as long as $\bar{R}$ is assumed to be nonsingular; this is proved directly by Barlow (see [15, Section 4.2]). We conjecture that Barlow's framework could be easily repurposed to prove rigorous stability bounds for `MGS-LTS` and `BMGS-LTS`. A full stability analysis for `MGS-LTS`, `MGS-CWY`, and `MGS-ICWY` and their block variants is the subject of ongoing work.

In Figure 16, we focus on just the `BMGS-SVL` skeleton and its one-sync version, `BMGS-CWY`, paired with all four "T"-variants as muscles as well as `HouseQR`. Standard κ-plots are on the left and `laeuchli` κ-plots are on the right. For the standard κ-plots, the behavior of all variants is nearly identical, achieving $\mathcal{O}(\varepsilon)$ loss of orthogonality. The more challenging `laeuchli` matrices are more discerning, allowing us to draw two main conclusions. First, the T-variant skeletons and muscles are not "mix-and-match"; `BMGS-SVL` works with `MGS-SVL`, but in combination with other T-variant muscles, the loss of orthogonality exceeds the $\mathcal{O}(\varepsilon)\,\kappa(\boldsymbol{\mathcal{X}})$ bound. Second, as with `BCGSI+LS` (cf. Section 4.4), normalization lagging in `BMGS-CWY` can lead to complete loss of orthogonality. Here, `BMGS-CWY` loses orthogonality completely after $\mathcal{O}(\varepsilon)\,\kappa^2(\boldsymbol{\mathcal{X}})$ exceeds 1, even when combined with an unconditionally stable muscle like `HouseQR`.

Figure 16: κ-plots for low-sync variants of BMGS. (a) standard κ-plot. (b) laeuchli κ-plot. For the standard κ-plot, all algorithms exhibit similar loss of orthogonality, even when mixing and matching low-sync skeleton and muscle variants. However, the laeuchli κ-plot shows both that (1) the one-sync skeleton, BMGS-CWY, can suffer greater loss of orthogonality than BMGS-SVL and (2) BMGS-SVL is only guaranteed to have $O(\varepsilon)\kappa(\boldsymbol{\mathcal{X}})$ loss of orthogonality when paired with its specialized MGS-SVL muscle or an unconditionally stable muscle like HouseQR.



5. Mixed-precision approaches

Due to the trend of commercially available mixed-precision hardware and its potential to provide computational time and energy savings, there is a growing interest in the use of mixed precision within numerical linear algebra routines. We comment briefly on existing efforts towards mixed-precision orthogonalization schemes.

In [51], a mixed-precision Cholesky QR factorization (mCholQR) is studied, in which some intermediate computations are performed at double the working precision. The authors prove that in such a mixed-precision setting, the loss of orthogonality depends only linearly (rather than quadratically) on the condition number of the input matrix under the assumption (as in CholQR) that $\mathcal{O}(\varepsilon)\,\kappa^2(\boldsymbol{X}) < 1$ (cf. Table 3).

Building on this work, in [52], the authors propose using the mCholQR of [51] as the IntraOrtho within BGS. The paper does not contain a formal error analysis, but various numerical experiments are performed to examine the loss of orthogonality and residual error when mCholQR is combined with BMGS and BCGS. The experiments suggest that in some cases the numerical stability of the BMGS ∘ mCholQR variants can be maintained to the same level of MGS if reorthogonalization is used.

We note also that in [53], the finite-precision behavior of HouseQR and its blocked variants is analyzed in various mixed-precision settings. The potential performance benefits of mixed precision make the study of the stability properties of mixed-precision BGS variants of great interest going forward.

6. Software Frameworks

In the following, we highlight which variants of BGS are implemented in well-known software packages.

To our knowledge, the Trilinos Library, in particular the Belos package, provides the most opportunity for the user to select among various orthogonalization schemes to be used within Krylov subspace methods. Belos currently contains implementations of `TSQR`, iterated CGS and MGS, and `CGSS+`. For block Krylov subspace methods, Belos implements various BGS methods. For example, in block GMRES, the user may choose between block versions of iterated CGS and MGS, and `CGSS+`, the default being block iterated CGS. A recent presentation [54] on the Trilinos Library presents experiments with communication-avoiding (s-step) GMRES using $\texttt{CGS} \circ \texttt{CholQR}$ as well as a "low-sync CGS" with `CholQR+` as the `IntraOrtho`. We also note that s-step GMRES with low-sync block Gram-Schmidt orthogonalization algorithms including `BCGS-PIP`, `BMGS-ICWY`, and `BCGSI+LS` is described in the recent paper by Yamazaki et al. [30]. The authors demonstrate significant parallel scaling improvements over the previous Trilinos implementations of these KSM algorithms and lower time-to-solution for the iterative solver. The implementation of communication-avoiding Krylov subspace methods and associated block orthogonalization schemes, in particular low-synch variants, is a goal of the ongoing Exascale Computing Project PEEKS effort.

Block variants of GMRES, CG, break-down free CG and recycling GCRO Krylov solvers have all been implemented recently in PETSc as described in the paper by Jolivet et al. [55]. The authors also describe blocked eigensolvers employed within the PETSc software framework that are based on SLEPc. These employ the Locally Optimal Block Preconditioned Conjugate Gradient (LOBPCG) algorithm due to Knyazev [56] and the Contour Integral Spectrum Slicing method (CISS) of Sakurai and Suguria [57].

For prototype algorithms and testing routines for stability conjectures, we direct the reader to the MATLAB package developed in conjunction with this survey, hosted at `https://github.com/katlund/BlockStab`.

7. Conclusions and outlook

The take-home lesson of stability studies is always the same, but often forgotten: be careful, because the finite precision universe can behave in exceptionally undesirable ways, especially if you have only analyzed your method in exact arithmetic. At the same time, when an algorithm works well "most of the time," there are probably good reasons, and we should be realistic about the statistical likelihood that something catastrophic may occur, especially if the algorithm is providing massive benefit in other ways.

Sometimes, however, stability may be a stringent requirement for a software pipeline. This is especially true if eigenvalue estimates are needed anywhere [2]. Lately, Krylov subspace methods, and particularly block Krylov subspace methods, form an important component in matrix function approximations, which often use eigenvalue estimations to accelerate convergence [58, 59, 60, 61, 62]. Other applications where reliable eigenvalue estimates are needed include Krylov subspace recycling and spectral deflation [63, 64].

It is also worth exploring whether the low-sync BGS variants – `BCGSI+LS`, `BMGS-SVL`, `BMGS-LTS`, `BMGS-CWY`, `BMGS-ICWY`– coupled with muscles such as `TSQR` or `ShCholQR++`, provide similar communication benefits as `BCGS` $\circ$ `TSQR` but with better stability properties. This is especially relevant for s-step KSMs, which rely on the low communication of `BCGS` but can also suffer from its instability.

Although our focus has been on the stability properties of block orthogonalization routines, an overarching goal is to be able to say something about the backward stability of block (and s-step) variants of GMRES methods. Paige, Rozložník, and Strakoš have proven that (non-block) GMRES with MGS orthogonalization is backward stable, despite loss of orthogonality of the Arnoldi basis vectors due to the finite precision MGS computation [11]. It remains an open problem, even in the non-blocked case, to determine the level to which orthogonality can be lost in the finite precision orthogonalization routine while still obtaining a backward stable GMRES solution. Further, theoretical treatment of the backward stability of block Arnoldi/GMRES methods is entirely lacking from the literature. We hope that the present work provides a path forward in this direction, and a detailed application of the BGS results compiled here to block GMRES and methods like s-step GMRES remains as future work. The natural next step is an examination of the least-squares problem for block upper Hessenberg matrices. Work by Gutknecht and Schmelzer [65] may prove relevant, as well as the low-rank update formulation devised in [61] for matrix functions.

Appendix A. Pseudocode for muscles

Algorithm 13 $[\boldsymbol{Q}, R] = \texttt{CGS}(\boldsymbol{X})$

```
1: $\boldsymbol{Q} = \boldsymbol{X}$
2: for $k = 1, \ldots, n$ do
3:     $R_{1:k-1,k} = \boldsymbol{Q}_{:,1:k-1}^T \boldsymbol{X}_{:,k}$
4:     $\boldsymbol{Q}_{:,k} = X_{:,k} - \boldsymbol{Q}_{:,1:k-1} R_{1:k-1,k}$
5:     $R_{k,k} = \|\boldsymbol{Q}_{:,k}\|$
6:     $\boldsymbol{Q}_{:,k} = \boldsymbol{Q}_{:,k} / R_{k,k}$
7: end for
8: return $\boldsymbol{Q}$, $R$
```

Algorithm 14 $[\boldsymbol{Q}, R] = \texttt{MGS}(\boldsymbol{X})$

```
1: $\boldsymbol{Q} = \boldsymbol{X}$
2: for $k = 1, \ldots, n$ do
3:     $R_{k,k} = \|\boldsymbol{Q}_{:,k}\|$
4:     $\boldsymbol{Q}_{:,k} = \boldsymbol{Q}_{:,k}/R_{k,k}$
5:     $R_{k,k+1:n} = Q_{:,k}^T Q_{:,k+1:n}$
6:     $Q_{:,k+1:n} = Q_{:,k+1:n} - Q_{:,k} R_{k,k+1:n}$
7: end for
8: return $\boldsymbol{Q}$, $R$
```

Algorithm 15 $[\boldsymbol{Q}, \boldsymbol{R}] = \texttt{CGSS+rpl}(\boldsymbol{X}, \delta)$

```
1: Allocate memory for $\boldsymbol{Q}$ and $R$
2: $[\boldsymbol{q}_1, \sim, r_{11}] = \texttt{cgs\_step\_sror}(\mathbf{0}, \boldsymbol{x}_1, \delta)$
3: for $k = 1, \ldots, s-1$ do
4:     $[\boldsymbol{q}_{k+1}, R_{1:k,k+1}, r_{k+1,k+1}] = \texttt{cgs\_step\_sror}(\boldsymbol{Q}_{1:k}, \boldsymbol{x}_{k+1}, \delta)$
5: end for
6: return $\boldsymbol{Q} = [\boldsymbol{q}_1, \ldots, \boldsymbol{q}_s]$, $R = (r_{jk})$
```

Algorithm 16 $[\boldsymbol{Q}, R, T] = \texttt{MGS-SVL}(\boldsymbol{X})$

```
1: Allocate memory for $\boldsymbol{Q}$ and $R$
2: $T = I_s$
3: $r_{11} = \|\boldsymbol{x}_1\|$; $\boldsymbol{q}_1 = \boldsymbol{x}_1/r_{11}$
4: for $k = 1, \ldots, s-1$ do
5:     $\boldsymbol{w} = \boldsymbol{x}_{k+1}$
6:     $R_{1:k,k+1} = T_{1:k,1:k}^T (\boldsymbol{Q}_{1:k}^T \boldsymbol{w})$
7:     $\boldsymbol{w} = \boldsymbol{w} - \boldsymbol{Q}_{1:k} R_{1:k,k+1}$
8:     $r_{k+1,k+1} = \|\boldsymbol{w}\|$
9:     $\boldsymbol{q}_{k+1} = \boldsymbol{w}/r_{k+1,k+1}$
10:    $T_{1:k,k+1} = -T_{1:k,1:k} (\boldsymbol{Q}_{1:k}^T \boldsymbol{q}_{k+1})$
11: end for
12: return $\boldsymbol{Q} = [\boldsymbol{q}_1, \ldots, \boldsymbol{q}_s]$, $R = (r_{jk})$, $T = (t_{jk})$
```

Algorithm 17 $[\boldsymbol{Q}, R, T] = \texttt{MGS-LTS}(\boldsymbol{X})$

```
1:  Allocate memory for $\boldsymbol{Q}$ and $R$
2:  $T = I_s$
3:  $r_{11} = \|\boldsymbol{x}_1\|$; $\boldsymbol{q}_1 = \boldsymbol{x}_1/r_{11}$
4:  for $k = 1, \ldots, s-1$ do
5:      $\boldsymbol{w} = \boldsymbol{x}_{k+1}$
6:      $R_{1:k,k+1} = T^{-T}_{1:k,1:k}(\boldsymbol{Q}^T_{1:k}\boldsymbol{w})$
7:      $\boldsymbol{w} = \boldsymbol{w} - \boldsymbol{Q}_{1:k}R_{1:k,k+1}$
8:      $r_{k+1,k+1} = \|\boldsymbol{w}\|$
9:      $\boldsymbol{q}_{k+1} = \boldsymbol{w}/r_{k+1,k+1}$
10:     $T_{1:k,k+1} = \boldsymbol{Q}^T_{1:k}\boldsymbol{q}_{k+1}$
11: end for
12: return $\boldsymbol{Q} = [\boldsymbol{q}_1, \ldots, \boldsymbol{q}_s]$, $R = (r_{jk})$, $T = (t_{jk})$
```

Algorithm 18 $[\boldsymbol{Q}, R, T] = \texttt{MGS-CWY}(\boldsymbol{X})$

```
1:  Allocate memory for $\boldsymbol{Q}$ and $R$
2:  $T = I_s$
3:  $\boldsymbol{u} = \boldsymbol{x}_1$
4:  for $k = 1, \ldots, s-1$ do
5:      $\boldsymbol{w} = \boldsymbol{x}_{k+1}$
6:      if $k = 1$ then
7:          $\begin{bmatrix} r^2_{k,k} & \rho \end{bmatrix} = \boldsymbol{u}^T[\boldsymbol{u}\ \boldsymbol{w}]$
8:      else if $k > 1$ then
9:          $\begin{bmatrix} \boldsymbol{t} & \boldsymbol{r} \\ r^2_{k,k} & \rho \end{bmatrix} = [\boldsymbol{Q}_{1:k-1}\ \boldsymbol{u}]^T[\boldsymbol{u}\ \boldsymbol{w}]$
10:         $T_{1:k-1,k} = -T_{1:k-1,1:k-1}(\boldsymbol{t}/r_{k,k})$
11:     end if
12:     $R_{1:k,k+1} = T^T_{1:k,1:k}\begin{bmatrix} \boldsymbol{r} \\ \rho/r_{k,k} \end{bmatrix}$
13:     $\boldsymbol{q}_k = \boldsymbol{u}/r_{k,k}$
14:     $\boldsymbol{u} = \boldsymbol{w} - \boldsymbol{Q}_{:,1:k}R_{1:k,k+1}$
15: end for
16: $r_{s,s} = \|\boldsymbol{u}\|$
17: $\boldsymbol{q}_s = \boldsymbol{u}/r_{s,s}$
18: return $\boldsymbol{Q} = [\boldsymbol{q}_1, \ldots, \boldsymbol{q}_s]$, $R = (r_{jk})$, $T = (t_{jk})$
```

Algorithm 19 $[\boldsymbol{Q}, R, T] = \texttt{MGS-ICWY}(\boldsymbol{X})$

```
 1: Allocate memory for $\boldsymbol{Q}$ and $R$
 2: $T = I_s$
 3: $\boldsymbol{u} = \boldsymbol{x}_1$
 4: for $k = 1, \dots, s-1$ do
 5:     $\boldsymbol{w} = \boldsymbol{x}_{k+1}$
 6:     if $k = 1$ then
 7:         $\begin{bmatrix} r_{k,k}^2 & \rho \end{bmatrix} = \boldsymbol{u}^T [\boldsymbol{u}\ \boldsymbol{w}]$
 8:     else if $k > 1$ then
 9:         $\begin{bmatrix} \boldsymbol{t} & \boldsymbol{r} \\ r_{k,k}^2 & \rho \end{bmatrix} = [\boldsymbol{Q}_{1:k-1}\ \boldsymbol{u}]^T [\boldsymbol{u}\ \boldsymbol{w}]$
10:         $T_{1:k-1,k} = \boldsymbol{t}/r_{k,k}$
11:     end if
12:     $R_{1:k,k+1} = T_{1:k,1:k}^{-T} \begin{bmatrix} \boldsymbol{r} \\ \rho/r_{k,k} \end{bmatrix}$
13:     $\boldsymbol{q}_s = \boldsymbol{u}/r_{s,s}$
14:     $\boldsymbol{u} = \boldsymbol{w} - \boldsymbol{Q}_{:,1:k} R_{1:k,k+1}$
15: end for
16: $r_{s,s} = \|\boldsymbol{u}\|$
17: $\boldsymbol{q}_s = \boldsymbol{u}/r_{s,s}$
18: return $\boldsymbol{Q} = [\boldsymbol{q}_1, \dots, \boldsymbol{q}_s]$, $R = (r_{jk})$, $T = (t_{jk})$
```

Algorithm 20 $[\boldsymbol{Q}, R] = \texttt{CholQR}(\boldsymbol{X})$

```
1: $X = \boldsymbol{X}^T \boldsymbol{X}$
2: if $X$ is positive definite then
3:     $R = \texttt{chol}(X)$
4:     $\boldsymbol{Q} = \boldsymbol{X} R^{-1}$
5: else
6:     return $\boldsymbol{Q} = \texttt{NaN}$, $R = \texttt{NaN}$
7: end if
8: return $\boldsymbol{Q}$, $R$
```

Algorithm 21 $[\boldsymbol{Q}, R] = \texttt{CholQR+}(\boldsymbol{X})$

```
1: $[\boldsymbol{Q}, R^{(1)}] = \texttt{CholQR}(\boldsymbol{X})$
2: $[\boldsymbol{Q}, R^{(2)}] = \texttt{CholQR}(\boldsymbol{Q})$
3: $R = R^{(2)} R^{(1)}$
4: return $\boldsymbol{Q}$, $R$
```

Algorithm 22 $[\boldsymbol{Q}, R] = \texttt{ShCholQR++}(\boldsymbol{X})$

```
1: Define shift $\sigma = 11(ms + s(s+1))\varepsilon \|\boldsymbol{X}\|_2^2$
2: $R^{(1)} = \texttt{chol}(\boldsymbol{X}^T \boldsymbol{X} + \sigma I_s)$
3: $\boldsymbol{Q} = \boldsymbol{X}(R^{(1)})^{-1}$
4: $[\boldsymbol{Q}, R^{(2)}] = \texttt{CholQR+}(\boldsymbol{Q})$
5: $R = R^{(2)} R^{(1)}$
```

Appendix B. Matlab code for routines in BCGSS+rpl and CGSS+rpl

```
function [y, r, rho, northog] = cgs_step_sror(Q, x, nu, rpltol)
% [y, r, rho, northog] = CGS_STEP_SROR(Q, x, nu, rpltol) orthogonalizes x
% against the columns of Q using the the Classical Gram-Schmidt method.
% Reorthogonalization is performed as necessary to ensure orthogonality.
% Specifically, given an orthonormal matrix Q and a vector x, the function
% returns a vectors y and r, and a scalar rho satisfying
%
% (*)   x = Q*r + rho*y
%
% where y is a normalized vector orthogonal to the column space of Q.
% The matrix Q may have zero columns.
%
% The optional argument nu is the norm of the original value of x, to be
% used when x has been subject to previous orthogonalizaton steps.  If
% nu is absent or is less than or equal to norm(x), is is set to norm(x).
%
% The optional argument rlptol controls when the current vector y is
% replaced by a random vector.  Its default value is 1.  If it is set
% to a value greater than one, the relation (*) will be compromised
% somewhat, but the number of orthogonalizations may be decreased.
%
% The optional output argument northog, if present, contains the number
% of orthogonalizations.
%
% Originally written by G. W. Stewart, 2008. Modified by Kathryn Lund,
% 2020.

%%
% Initialize
[n, nq] = size(Q);
r = zeros(nq, 1);
nux = norm(x);
if nargout >= 4
northog = 0;
end

% If Q has no columns, return normalized x if x~=0, otherwise return a
% normalized random vector.
if nq == 0
if nux == 0
y = rand(n,1)-0.5;
y = y/norm(y);
rho = 0;
else
y = x/nux;
rho = nux;
end
return
```

```
end

%  If required, intitalize the optional arguments nu and rpltol.
if nargin < 3
nu = nux;
else
if nu < nux
nu = nux;
end
end

if nargin < 4
rpltol = 1;
end

% If norm(x)==0, set it to a nonzero vector and proceed, noting the fact in
% the flag zeronorm.
if nux ~= 0
zeronorm = false;
y = x/nux;
nu = nu/nux;
else
zeronorm = true;
y = rand(n,1) - 0.5;
y = y/norm(y);
nu = 1;
end

% Main orthogonalization loop.
nu1 = nu;
while true
if nargout == 4
northog = northog + 1;
end
s = Q'*y;
r = r + s;
y = y - Q*s;
nu2 = norm(y);

% Return if y is orthogonal.
if nu2 > 0.5*nu1
break
end

% Continue orthogonalizing if nu2 is not too small.
if nu2 > rpltol*nu*eps
nu1 = nu2;
else % Replace y by a random vector.
nu = nu*eps;
nu1 = nu;
```

```
y = rand(n,1) - 0.5;
y = nu*y/norm(y);
end
end

% Calculate rho and normalize y.
if ~zeronorm
rho = norm(y);
y = y/rho;
rho = rho*nux;
r = r*nux;
else
y = y/norm(y);
r = zeros(nq, 1);
rho = 0;
end


function [Y, R12, R22] = bcgs_step_sror(QQ, X, rpltol)
% [Y, R12, R22] = BCGS_STEP_SROR(QQ, X, rpltol) returns an orthonormal
% matrix Y whose columns lie in the orthogonal complement of the column
% space of QQ, a matrix R12, and an upper triangular matrix R22 satisfying
%
% (*)   X = QQ*R12 + Y*R22.
%
% The optional argument rpltol has a default value of 1.  Increasing it,
% say to 100, may improve performance, but will degrade the accuracy of the
% relation (*).
%
% Originally written by G. W. Stewart, 2008.  Modified by Kathryn Lund,
% 2020.

%%
%  Initialization.
reorth = false;
if nargin < 3
rpltol = 1;
else
if rpltol < 1
rpltol = 1;
end
end

s = size(X,2);
sk = size(QQ,2);

nu = zeros(1,s);
for k = 1:s
nu(k) = norm(X(:,k));
end
```

```
%  Beginning of the first orthogonalization step.  Project Y
%  onto the orthogonal complement of Q.
R12 = QQ'*X;
Y = X - QQ*R12;
R22 = zeros(s);

%  Orthogonalize the columns of Y.
for k = 1:s
[yk, r, rho] = cgs_step_sror(Y(:,1:k-1), Y(:,k), nu(k), rpltol);
Y(:,k) = yk;
R22(1:k-1,k) = r;
R22(k,k) = rho;
if rho <= 0.5*nu(k)
reorth = true;
end
end

%  Return if Q has zero columns.
if sk == 0 || ~reorth
 return
end

% Beginning of the reorthogonalization.  Project Y onto the orthogonal
% complement of Q.
S12 = QQ'*Y;
Y = Y - QQ*S12;

% Orthogonalize the columns of Y.
S22 = zeros(s);
for k = 1:s
[yk, r, sig] = cgs_step_sror(Y(:,1:k-1), Y(:,k), 1, rpltol);

if sig < 0.5
% The result, yk, is not satisfactorily orthogonal.  Perform an
% unblocked orthogonalization against Q and the previous columns of Y.
[yk, r, sig] = cgs_step_sror([QQ, Y(:,1:k-1)], Y(:,k), rpltol);
S12(:,k) = S12(:,k) + r(1:sk);
r = r(sk+1:sk+k-1);
end
Y(:,k) = yk;
S22(1:k-1,k) = r;
S22(k,k) = sig;
end

R12 = R12 + S12*R22;
R22 = S22*R22;
end
```

Appendix C. Glued matrices and command-line calls

Glued matrices are generated by the following MATLAB code, modified from [28]:

```
function A = CreateGluedMatrix(m, p, s, r, t)
% Example 2 matrix from [Smoktunowicz et al., 2006]. Generates a glued
% matrix of size m x ps, with parameters r and t specifying the powers of
% the largest condition number of the first stage and second stages of the
% matrix, respectively:
%
% Stage 1: A = U * Sigma * V'
%
% Stage 2: A = A * kron(I, Sigma_block) * kron(I, V_block)

%%
n = p * s;
U = orth(randn(m,n));
V = orth(randn(n,m));
Sigma = diag(logspace(0, r, n));
A = U * Sigma * V';

Sigma_block = diag(logspace(0, t, s));
V_block = orth(randn(s,s));

ind = 1:s;
for i = 1:p
A(:,ind) = A(:,ind) * Sigma_block * V_block';
ind = ind + s;
end
end
```

All the heat-maps in Section 3 can be reproduced with the code hosted at `https://github.com/katlund/BlockStab` and the following commands:

```
XXdim = [10000 50 10];
mat = {'rand_uniform', 'rand_normal', 'rank_def', 'laeuchli',...
    'monomial', 'stewart', 'stewart_extreme', 'hilbert',...
    's-step', 'newton'};
skel = {'BCGS', 'BCGS_IRO', 'BCGS_SROR', 'BCGS_IRO_LS',...
    'BMGS', 'BMGS_SVL', 'BMGS_CWY'};
musc = {'CGS', 'CGS_IRO', 'CGS_SRO', 'CGS_SROR',...
    'CGS_IRO_LS', 'MGS', 'MGS_SVL', 'MGS_CWY', 'HouseQR',...
    'CholQR', 'CholQR_RO', 'Sh_CholQR_RORO'};
rpltol = 100;
verbose = 1;
MakeHeatmap(XXdim, mat, skel, musc, rpltol, verbose)
```

All the κ-plots in Section 4 can also be easily reproduced; see Table C.6.

Table C.6: Command-line calls for κ-plots

Figure	Code
10	`GluedBlockKappaPlot([1000 50 4], 1:8, ...`
	`{'BCGS', 'BCGS_PIP', 'BCGS_PIO'}, ...`
	`{'CholQR', 'HouseQR'})`
11	`BlockKappaPlot([100 20 2], -(1:16), ...`
	`    {'BCGS', 'BCGS_IRO'}, ...`
	`    {'CGS', 'MGS', 'HouseQR'})`
12	`LaeuchliBlockKappaPlot([1000 100 5], ...`
	`    logspace(-1,-16,10), ...`
	`    {'BCGS', 'BCGS_IRO'}, ...`
	`    {'CGS', 'MGS', 'HouseQR'})`
13	`BlockKappaPlot([100 20 2], -(1:16), ...`
	`    {'BCGS', 'BCGS_IRO', 'BCGS_IRO_LS'}, ...`
	`    {'CGS', 'CGS_IRO', 'CGS_IRO_LS'})`
14	`LaeuchliBlockKappaPlot([1000 120 2], ...`
	` logspace(-1,-16,10), ...`
	`    {'BCGS', 'BCGS_IRO', 'BCGS_IRO_LS'}, ...`
	`    {'CGS', 'CGS_IRO', 'CGS_IRO_LS'})`
15	`BlockKappaPlot([100 20 2], -(1:16), 'BMGS', ...`
	`    {'CGS', 'MGS', 'CGS_RO', ...`
	`    'CholQR', 'Cholqr_RO', 'HouseQR'})`
	`LaeuchliBlockKappaPlot([1000 120 2], ...`
	`  logspace(-1,-16,10), 'BMGS', ...`
	`    {'CGS', 'MGS', 'CGS_RO', ...`
	`    'CholQR', 'Cholqr_RO', 'HouseQR'})`
16	`BlockKappaPlot([100 20 2], -(1:16), ...`
	`    {'BMGS_SVL', 'BMGS_CWY'}, ...`
	`    {'MGS_SVL', 'MGS_LTS', ...`
	`    'MGS_CWY', 'MGS_ICWY', 'HouseQR'})`
	`LaeuchliBlockKappaPlot([1000 120 2], ...`
	`    logspace(-1,-16,10), ...`
	`    {'BMGS_SVL', 'BMGS_CWY'}, ...`
	`    {'MGS_SVL', 'MGS_LTS', ...`
	`    'MGS_CWY', 'MGS_ICWY', 'HouseQR'})`

References

[1] G. Golub, R. Underwood, The block Lanczos method for computing eigenvalues, in: J. R. Rice (Ed.), Mathematical Software, Academic Press, 1977, pp. 361–377.

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

[3] D. P. O'Leary, The block conjugate gradient algorithm and related methods, Lin. Alg. Appl. 29 (1980) (1980) 293–322.

[4] A. H. Baker, J. M. Dennis, E. R. Jessup, On improving linear solver performance: a block variant of GMRES, SIAM J. Sci. Comput. 27 (5) (2006) 1608–1626.

[5] G. Ballard, E. C. Carson, J. W. Demmel, M. Hoemmen, N. Knight, O. Schwartz, Communication lower bounds and optimal algorithms for numerical linear algebra, Acta Numer. 23 (2014) (2014) 1–155.

[6] L. Grigori, S. Moufawad, F. Nataf, Enlarged Krylov subspace conjugate gradient methods for reducing communicaiton, SIAM J. Matrix Anal. Appl. 37 (2) (2016) 744–773.

[7] J. Demmel, L. Grigori, M. Hoemmen, J. Langou, Communication-optimal parallel and sequential QR and LU factorizations (2008). arXiv:0808.2664.

[8] Å. Björck, Solving linear least squares problems by Gram-Schmidt orthogonalization, BIT 7 (1967) 1–21.

[9] Å. Björck, C. C. Paige, Loss and recapture of orthogonality in the modified Gram–Schmidt algorithm, SIAM J. Matrix Anal. Appl. 13 (1) (1992) 176–190.

[10] N. J. Higham, Accuracy and Stability of Numerical Algorithms, 2nd Edition, SIAM, Philadelphia, 2002.

[11] C. C. Paige, M. Rozložník, Z. Strakoš, Modified Gram-Schmidt (MGS), least squares, and backward stability of MGS-GMRES, SIAM J. Matrix Anal. Appl. 28 (1) (2006) 264–284.

[12] Y. Yamamoto, Y. Nakatsukasa, Y. Yanagisawa, T. Fukaya, Roundoff error analysis of the CholeskyQR2 algorithm, Electron. Trans. Numer. Anal. 44 (2015) 306–326.

[13] T. Fukaya, R. Kannan, Y. Nakatsukasa, Y. Yamamoto, Y. Yanagisawa, Shifted Cholesky QR for computing the QR factorization of ill-conditioned matrices, SIAM J. Sci. Comput. 42 (1) (2020) A477–A503.

[14] S. J. Leon, Å. Björck, W. Gander, Gram-Schmidt orthogonalization: 100 years and more, Numer. Lin. Alg. Appl. 20 (3) (2013) 492–532.

[15] J. L. Barlow, Block modified Gram-Schmidt algorithms and their analysis, SIAM J. Matrix Anal. Appl. 40 (4) (2019) 1257–1290.

[16] J. L. Barlow, A. Smoktunowicz, Reorthogonalized block classical Gram-Schmidt, Numer. Math. 123 (3) (2013) 395–423.

[17] D. Vanderstraeten, An accurate parallel block Gram-Schmidt algorithm without reorthogonalization, Numer. Lin. Alg. Appl. 7 (4) (2000) 219–236.

[18] W. Jalby, B. Philippe, Stability analysis and improvement of the block Gram-Schmidt algorithm, SIAM J. Sci. Comput. 12 (5) (1991) 1058–1073.

[19] M. Sadkane, Block-Arnoldi and Davidson methods for unsymmetric large eigenvalue problems, Numer. Math. 64 (1) (1993) 195–211.

[20] V. Simoncini, E. Gallopoulos, Convergence properties of block GMRES and matrix polynomials, Lin. Alg. Appl. 247 (1996) 97–119.

[21] Y. Saad, Iterative methods for sparse linear systems, 2nd Edition, SIAM, Philadelphia, 2003.

[22] R. B. Morgan, Restarted block-GMRES with deflation of eigenvalues, Appl. Numer. Math. 54 (2) (2005) 222–236.

[23] M. H. Gutknecht, Block Krylov space methods for linear systems with multiple right-hand sides: An introduction, in: A. H. Siddiqi, I. S. Duff, O. Christensen (Eds.), Modern Mathematical Models, Methods and Algorithms for Real World Systems, Anamaya Publishers, 2007, pp. 420–447.

[24] M. Hoemmen, Communication-avoiding Krylov subspace methods, Ph.D. thesis, Department of Electrical Engineering and Computer Science, University of California at Berkeley (2010).

[25] K. Świrydowicz, J. Langou, S. Ananthan, U. Yang, S. Thomas, Low synchronization Gram–Schmidt and generalized minimal residual algorithms, Num. Lin. Alg. Appl. 28 (2) (2021) e2343.

[26] J. W. Demmel, L. Grigori, M. Hoemmen, J. Langou, Communication-optimal parallel and sequential QR and LU factorizations, SIAM J. Sci. Comput. 34 (1) (2012) A206–A239.

[27] A. Bienz, W. Gropp, L. Olson, Node-aware improvements to allreduce, in: Proceedings of the 2019 IEEE/ACM Workshop on Exascale MPI (ExaMPI), ACM, 2019.

[28] A. Smoktunowicz, J. L. Barlow, J. Langou, A note on the error analysis of classical Gram-Schmidt, Numer. Math. 105 (2) (2006) 299–313.

[29] E. Carson, K. Lund, M. Rozložník, The stability of block variants of classical Gram-Schmidt, SIAM J. Matrix Anal. Appl. 42 (3) (2021) 1365–1380.

[30] I. Yamazaki, S. Thomas, M. Hoemmen, E. G. Boman, K. Świrydowicz, J. J. Eilliot, Low-synchronization orthogonalization schemes for s-step and pipelined Krylov solvers in Trilinos, in: Proceedings of the SIAM Conference on Parallel Processing 2020, Seattle WA, 2020.

[31] N. N. Abdelmalek, Round off error analysis for Gram-Schmidt method and solution of linear least squares problems, BIT 11 (1971) 345–367.

[32] B. N. Parlett, The Symmetric Eigenvalue Problem, SIAM, Philadelphia, 1998.

[33] J. W. Daniel, W. B. Gragg, L. Kaufman, G. W. Stewart, Reorthogonalization and stable algorithms for updating the Gram-Schmidt QR factorization, Math. Comput. 30 (1976) 772–795.

[34] W. Hoffmann, Iterative algorithms for Gram-Schmidt orthogonalization, Computing 41 (4) (1989) 335–348.

[35] G. W. Stewart, Matrix Algorithms Volume 1: Basic Decompositions, SIAM, Philadelphia, 1998.

[36] G. W. Stewart, Block Gram-Schmidt orthogonalization, SIAM J. Sci. Comput. 31 (1) (2008) 761–775.

[37] A. Ruhe, Numerical aspects of Gram-Schmidt orthogonalization of vectors, Lin. Alg. Appl. 52 (1983) 591–601.

[38] B. Vital, Étude de quelques méthodes de résolution de problèmes linéaires de grande taille sur multiprocesseur, Ph.D. thesis, Université de Rennes (1990).

[39] S. K. Kim, A. T. Chronopoulos, An efficient parallel algorithm for extreme eigenvalues of sparse nonsymmetric matrices, Int. J. Supercomputing Appl. 6 (1) (1992) 98–111.

[40] R. Schreiber, C. Van Loan, A storage-efficient WY representation for products of Householder transformations, SIAM J. Sci. Stat. Comput. 10 (1) (1989) 53–57.

[41] H. F. Walker, Implementation of the GMRES method using Householder transformations, SIAM J. Sci. Stat. Comput. 9 (1) (1988) 152–163.

[42] X. Sun, Aggregations of elementary transformations., Tech. Rep. DUKE-TR-1996-03, Duke University (1996).

[43] G. H. Golub, C. F. Van Loan, Matrix Computations, 4th Edition, Johns Hopkins University Press, Baltimore, 2013.

[44] E. C. Carson, An Adaptive s-step Conjugate Gradient Algorithm with Dynamic Basis Updating, Appl. Math. 65 (2) (2020) 123–151.

[45] A. Kiełbasiński, Analiza numeryczna algorytmu ortogonałizacji Grama-Schmidta, Ser. III Mat. Stosow. II (1974) 15–35.

[46] L. Giraud, J. Langou, M. Rozložník, J. Van Den Eshof, Rounding error analysis of the classical Gram-Schmidt orthogonalization process, Numer. Math. 101 (2005) 87–100.

[47] L. Giraud, J. Langou, When modified Gram-Schmidt generates a well-conditioned set of vectors, IMA J. Numer. Anal. 22 (4) (2002) 521–528.

[48] W. Gander, Algorithms for the QR-decomposition, Tech. Rep. 80-02, April 1980, Seminar für Angewandte Mathematik, Eidgenössische Technische Hochschule (1980).

[49] D. Mori, Y. Yamamoto, S. L. Zhang, Backward error analysis of the AllReduce algorithm for Householder QR decomposition, Jpn. J. Ind. Appl. Math. 29 (1) (2012) 111–130.

[50] E. C. Carson, Communication-avoiding Krylov subspace methods in theory and practice, Ph.D. thesis, Department of Electrical Engineering and Computer Science, University of California, Berkeley (2015).

[51] I. Yamazaki, S. Tomov, J. Dongarra, Mixed-precision Cholesky QR factorization and its case studies on multicore CPU with multiple GPUs, SIAM J. Sci. Comput. 37 (3) (2015) C307–C330.

[52] I. Yamazaki, S. Tomov, J. Kurzak, J. Dongarra, J. L. Barlow, Mixed-precision block Gram Schmidt orthogonalization, in: Proceedings of the 6th Workshop on Latest Advances in Scalable Algorithms for Large-Scale Systems, ScalA '15, ACM, 2015.

[53] L. M. Yang, A. Fox, G. Sanders, Rounding error analysis of mixed precision block Householder QR algorithms (2021). arXiv:1912.06217.

[54] I. Yamazaki, M. Hoemmen, Communication-avoiding & pipelined Krylov solvers in Trilinos, SIAM Conference on Computational Science and Engineering, Spokane, Washington, presentation (2019).

[55] P. Jolivet, J. E. Roman, S. Zampini, KSPHPDDM and PCHPDDM: extending PETSc with advanced Krylov methods and robust multilevel overlapping Schwarz preconditioners, Computers and Math. with Appl. 84 (2021) 277–295.

[56] A. V. Knyazev, Toward the optimal preconditioned eigensolver: locally optimal block preconditioned conjugate gradient method, SIAM J. Sci. Comput. 23 (2) (2001) 517–541.

[57] T. Sakurai, H. Sugiura, A projection method for generalized eigenvalue problems using numerical integration, J. Comput. Appl. Math. 159 (1) (2003) 119–128.

[58] M. Eiermann, O. G. Ernst, S. Güttel, Deflated restarting for matrix functions, SIAM J. Matrix Anal. Appl. 32 (2) (2011) 621–641.

[59] A. Frommer, S. Güttel, M. Schweitzer, Efficient and stable Arnoldi restarts for matrix functions based on quadrature, SIAM J. Matrix Anal. Appl. 35 (2) (2014) 661–683.

[60] A. Frommer, S. Güttel, M. Schweitzer, Convergence of restarted Krylov subspace methods for Stieltjes functions of matrices, SIAM J. Matrix Anal. Appl. 35 (4) (2014) 1602–1624.

[61] A. Frommer, K. Lund, D. B. Szyld, Block Krylov subspace methods for functions of matrices II: Modified block FOM, SIAM J. Matrix Anal. Appl. 41 (2) (2020) 804–837.

[62] S. Güttel, Rational Krylov approximation of matrix functions: Numerical methods and optimal pole selection, GAMM Mitteilungen 36 (1) (2013) 8–31.

[63] M. L. Parks, E. De Sturler, G. Mackey, D. D. Johnson, S. Maiti, Recycling Krylov subspaces for sequences of linear systems, SIAM J. Sci. Comput. 28 (5) (2006) 1651–1674.

[64] M. L. Parks, K. M. Soodhalter, D. B. Szyld, A block recycled GMRES method with investigations into aspects of solver performance (2016). arXiv:1604.01713.

[65] M. H. Gutknecht, T. Schmelzer, Updating the QR decomposition of block tridiagonal and block Hessenberg matrices, Appl. Numer. Math. 58 (6) (2008) 871–883.