INSTITUTE OF MATHEMATICS

THE CZECH ACADEMY OF SCIENCES

An overview of block Gram-Schmidt methods and their stability properties

Erin Carson
Kathryn Lund
Miroslav Rozložník
Stephen Thomas

Preprint No. 50-2020

PRAHA 2020

AN OVERVIEW OF BLOCK GRAM-SCHMIDT METHODS AND THEIR STABILITY PROPERTIES

ERIN CARSON*, KATHRYN LUND†,†, MIROSLAV ROZLOŽNÍK‡, AND STEPHEN THOMAS§

Abstract. Block Gram-Schmidt algorithms comprise essential kernels in many scientific computing applications, but for many commonly used variants, a rigorous treatment of their stability properties remains open. This survey provides a comprehensive categorization of block Gram-Schmidt algorithms, especially those used in Krylov subspace methods to build orthonormal bases one block vector at a time. All known stability results are assembled, and new results are summarized or conjectured for important communication-reducing variants. A diverse array of numerical illustrations are presented, along with the MATLAB code for reproducing the results in a publicly available at repository https://github.com/katlund/BlockStab. A number of open problems are discussed, and an appendix containing all algorithms type-set in a uniform fashion is provided.

Key words. Gram-Schmidt, block Arnoldi, block Krylov subspace methods, stability, loss of orthogonality

AMS subject classifications. 15-02, 15A23, 65-02, 65F05, 65F10, 65F25

1. Introduction. Block Gram-Schmidt (BGS) algorithms are kernels in a variety of important applications in scientific computing. 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 [22], Stewart [52], and O'Leary [39]. Block KSMs 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 [2].

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 [3, 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 [24]; 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 $R \in \mathbb{R}^{s\times s}$[1]. Popular choices include classical Gram-Schmidt (with

*Faculty of Mathematics and Physics, Charles University, Prague, Czech Republic, {carson, lundka}@karlin.mff.cuni.cz. Supported by Charles University PRIMUS project no. PRIMUS/19/SCI/11.

†Also supported by Charles University Research program no. UNCE/SCI/023.

‡Institute of Mathematics, Czech Academy of Sciences, Prague, Czech Republic. 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.

§National Renewable Energy Laboratory, Boulder, Colorado, USA 80401

[1]Although we formulate all results here in terms of $\mathbb{R}$, they should transfer with little effort to $\mathbb{C}$.

reorthogonalization), modified Gram-Schmidt, Householder-based QR, and more recently Cholesky-based QR. Backward stability analyses for all these methods are well established and understood, and they are foundational for the analysis of BGS methods; see, e.g., [7, 8, 28, 40, 59, 18], and especially the survey by Leon, Björck, and Gander [36].

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. By *stability*, we refer specifically to the *loss of orthogonality* of computed basis vectors, which is important for down-stream applications, such as the backward stability analysis of (block) GMRES.

Recent work by Barlow [4], as well as important results by Barlow and Smoktunowicz [5], Vanderstraeten [56], and Jalby and Philippe [31], appear to be the only rigorous stability treatments of BGS. Some of the most influential papers on block Krylov subspace methods (KSMs) devote little or no attention to the stability analysis of the underlying block Arnoldi, and implicitly, BGS, routine. Consider, for example, the following highly cited books and papers, most of which implicitly or explicitly use an algorithm derived from the block modified Gram-Schmidt skeleton, referred to here as `BMGS` and given as Algorithm A.6 in the appendix:

- 1980: O'Leary [39] 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 [46] 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 [49] 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 [45] 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 [37] uses Ruhe's variant of block Arnoldi (see, e.g., [45, 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 [2] 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 [25] 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`.

The intention of this list is not to chide our fellow researchers but merely to highlight

a hidden danger: no one has proven stability bounds for BMGS with HouseQR, and yet the numerical linear algebra community (and beyond) has been using this algorithm ubiquitously for decades. Even worse, it *is* known that BMGS with, for example, MGS, can suffer from steep orthogonality loss (see [31], our Section 3, as well as Section 4.5), and likely block Conjugate Gradients with MGS as well.

Our goal with the present work is twofold: to provide a survey of existing variants of BGS algorithms by classifying them according to a skeleton-muscle framework first proposed by Hoemmen [29]; and to collect, fill in, and note the challenges in proving stability results for commonly used skeleton-muscle combinations. In particular, we highlight how the stability of a BGS skeleton depends intrinsically on that of the intra-orthogonalization muscle. We also formulate results with an eye towards block Arnoldi/GMRES and communication-avoiding variants (like s-step and enlarged), which can themselves be expressed as block KSMs with special operators and starting vectors. 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 [40]. 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 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, block generalizations of the low-synchronization algorithms by Świrydowicz, et al. [55] and block versions of CGS-P by Smoktunowicz, et al. [50]. 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 IntraOrthos 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. Appendices A and B contain pseudocode for algorithms, and Appendix C contains MATLAB scripts for reproducing the plots in Sections 3 and 4.

2. Block Gram-Schmidt variants and a skeleton-muscle analogy. Given a matrix $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{m\times n}$, $m \gg n$, partitioned into a set of p *block vectors*, each of size $m \times s$ ($n = ps$),

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

we treat block Gram-Schmidt (BGS) algorithms that return an "economic" QR factorization

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

where $\boldsymbol{\mathcal{Q}} = [\boldsymbol{Q}_1 \,|\, \boldsymbol{Q}_2 \,|\, \cdots \,|\, \boldsymbol{Q}_p] \in \mathbb{R}^{m\times n}$ 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 $s \times s$.[2] We focus on scenarios where $\boldsymbol{\mathcal{X}}$ is a set of block vectors that may not all

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

be available at once, as is the case in block Arnoldi or block Krylov subspace methods (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 [13]. 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 this paper, 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. [6]. 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 [29]. 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.

The analogy is best understood via the literal meanings of skeleton and muscle in mammals. Without muscles,[3] 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. Pseudocode for each skeleton can be found in Appendix A, and unless otherwise noted, column versions of block algorithms can be obtained by setting $s = 1$ and replacing `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 `BGS` $\circ$ `IntraOrtho`, where `BGS` is a general skeleton, `IntraOrtho` is a general muscle, 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" be 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 2.1.

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

TABLE 2.1
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

2.1. Block classical Gram-Schmidt skeletons.

2.1.1. `BCGS`. Block Classical Gram-Schmidt (`BCGS`) is a straightforward generalization of `CGS` by replacing vectors with block vectors and norms with `IntraOrthos`; see Algorithm A.1. `BCGS` is especially attractive in massively parallel settings because it passes $O(p)$ fewer messages asymptotically than block Modified Gram-Schmidt (`BMGS`). See [29, Table 2.4] and also discussions in [3, 12, 13] for more details regarding the communication properties of `BCGS`.

By itself, `BCGS` is not often considered a viable skeleton, especially not prior to its use in communication-avoiding Krylov subspace methods [12]. 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 polynomial in terms of $\kappa(\boldsymbol{\mathcal{X}})$; see Section 4.1. To overcome this issue for `CGS`, Smoktunowicz et al. introduced a variant of `CGS`, whereby the diagonal entries of the R-factor are computed via the Pythagorean theorem. The resulting algorithm (`CGS-P`) has $\mathcal{O}(\varepsilon)\kappa^2(\boldsymbol{\mathcal{X}})$, as long as $\mathcal{O}(\varepsilon)\kappa^2(\boldsymbol{\mathcal{X}}) < 1$, which can be a steep limitation for many practical applications.

A similar correction is possible for `BCGS`, but a block Pythagorean theorem is needed instead.

THEOREM 2.1. *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*

$$\boldsymbol{X}^T\boldsymbol{X} = \boldsymbol{Y}^T\boldsymbol{Y} + \boldsymbol{Z}^T\boldsymbol{Z}. \tag{2.1}$$

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

$$R^T R = S^T S + T^T T. \tag{2.2}$$

Proof. It directly follows that

$$\begin{aligned} R^T R = \boldsymbol{X}^T \boldsymbol{X} &= (\boldsymbol{Y} + \boldsymbol{Z})^T (\boldsymbol{Y} + \boldsymbol{Z}) \\ &= \boldsymbol{Y}^T \boldsymbol{Y} + \underbrace{\boldsymbol{Z}^T \boldsymbol{Y} + \boldsymbol{Y}^T \boldsymbol{Z}}_{=0, \text{ because } \boldsymbol{Y} \perp \boldsymbol{Z}} + \boldsymbol{Z}^T \boldsymbol{Z} \\ &= S^T S + T^T T. \end{aligned}$$

∎

Stability analysis and implementation variants for `BCGS-P` (i.e., `BCGS` with a block Pythagorean correction) are the subject of ongoing work; see also Section 4.1. We note, however, 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. See Algorithms A.2 and A.3, as well as [61, Figure 2].

2.1.2. `BCGS` with reorthogonalization. It is well known that reorthogonalization can stabilize `CGS`; see, e.g., [1]. We note that there are essentially two reorthogonalization approaches: simply running the algorithm twice (`CGS+`) and performing the inner loop twice (`CGSI+`, where the `I` stands for "inner loop"). While the two approaches differ in floating-point arithmetic, they exhibit the same stability and communication behavior. In particular, they achieve $\mathcal{O}(\varepsilon)$ loss of orthogonality, thanks to the "twice is enough" principle [43], and double the communication costs of `CGS`. `CGSI+` is preferred when one wants to selectively orthogonalize, 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., [11, 30, 36, 51, 56].

Both approaches can be easily generalized to block algorithms. `BCGSI+` (Algorithm A.4) has received more attention in recent years and is analyzed in detail by Barlow and Smoktunowicz in 2013 [5]. 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` ∘ `HouseQR`.

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 [53] developed what we call `CGS` with Selective Reorthogonalization and Replacement (`CGSS+rpl`, Algorithm B.1), which replaces low-quality vectors with random ones of the same magnitude. He also developed a block variant (`BCGSS+rpl`, Algorithm A.11), 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 [29] 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 function `cgs_step_sror`, which is also reproduced in the Appendix (Figure B).[4] The latter is in fact the core of the algorithm and performs 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 additionally replaces identically zero vectors with random ones of small norm, an essential feature that is not explicitly mentioned in Stewart's paper [53] 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 $\mathcal{Q}$ 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., [29, Algorithm 13]). We do not explore this further here, however, because there is little practical benefit from the communication point-of-view.

2.1.4. One-sync `BCGSI+`. Recently, a number of low-synchronization variants of Gram-Schmidt have been proposed [55]. 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 matrix-vector products, inner products, and norms, because large vectors are often stored and operated on in a distributed fashion.

`CGSI+` (if implemented like Algorithm A.4 with $s = 1$) has up to four synchronization points per column, while Algorithm 3 from [55]– which we denote here as `CGSI+LS`, where "LS" stands for "low-synchronization"– has just one per column. See Algorithm B.2. Note that it is formulated a bit differently from the presentation in [55]; in particular, lines 21-23 introduce a single final synchronization point necessary for orthogonalizing the last column.

Note that `CGSI+LS` differs from Cholesky-based QR [18, 59], such as `CholQR`, which has a single synchronization point for the entire algorithm. Reorthogonalized `CholQR` (`CholQR+`) and shifted and reorthogonalized `CholQR` (`ShCholQR++`) require two and three total synchronization points, respectively. See Algorithms B.7-B.9 to compare details.

The low-synchronization algorithms from [55] 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.

[4] 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 [53], we have therefore incorporated the original implementations into our package, to facilitate and preserve accessibility.

Ruhe [44] 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 [55]):

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

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 B.2, 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 B.2.

A block generalization of Algorithm B.2 is rather straightforward; see Algorithm A.5. 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 B.2 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 A.5. 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 is essentially the same as [61, Figure 3], which is known to be unstable. Algorithm A.5 corrects this instability but still exhibits some problems for certain classes of matrices; see Sections 3 and 4.4.

2.2. Block modified Gram-Schmidt skeletons.

2.2.1. BMGS. Much like BCGS, Block Modified Gram-Schmidt (BMGS) is a straightforward replacement of the column vectors of MGS with block vectors; see Algorithm A.6. An initial stability study can be found in [31], and the algorithm was perhaps first considered in a high-performance setting by Vital [57].

A quick comparison between Algorithms A.1 and A.6 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.

2.2.2. Low-sync BMGS variants. Although increases in floating-point or communication operations 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 A.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 A.6 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:

$$\texttt{CGS}: \quad (I - \boldsymbol{Q}_{1:k}\boldsymbol{Q}_{1:k}^T)\boldsymbol{x}_{k+1} \tag{2.3}$$

$$\texttt{MGS}: \quad (I - \boldsymbol{q}_k\boldsymbol{q}_k^T)\cdots(I - \boldsymbol{q}_1\boldsymbol{q}_1^T)\boldsymbol{x}_{k+1} \tag{2.4}$$

It is well known that the application of the single projector (2.3) is unstable, but breaking it up like (2.4) 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.

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) [34], in order to bring the number of synchronization points per vector down to one:

- MGS-SVL [4, Func. 3.1]: Barlow calls this variant "MGS2." We prefer the suffix "SVL" for "Schreiber and Van Loan," on whose work [48] the development of this algorithm is largely based. See Algorithm B.3.
- MGS-LTS [55, Alg. 4]: This variant is not assigned a name in [55]; 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 B.4
- MGS-CWY [55, 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 B.5.
- MGS-ICWY [55, Alg. 5]: The suffix here stands for "Inverse Compact WY," referring to an alternative formulation of the MGS projector (2.4). It turns out that MGS-ICWY is the one-sync version of MGS-LTS. See Algorithm B.6.

The differences between the four algorithms are highlighted in violet in the Appendix.

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.

$$\texttt{MGS-SVL}, \texttt{MGS-CWY}: \quad (I - \boldsymbol{Q}_{1:k}T_{1:k,1:k}^T\boldsymbol{Q}_{1:k}^T)\boldsymbol{x}_{k+1} \tag{2.5}$$

$$\texttt{MGS-LTS}, \texttt{MGS-ICWY}: \quad (I - \boldsymbol{Q}_{1:k}T_{1:k,1:k}^{-T}\boldsymbol{Q}_{1:k}^T)\boldsymbol{x}_{k+1} \tag{2.6}$$

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 [58] and the technical report by Sun [54]. 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 B.5, 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 [4], equation (2.17) on page 1262.

Block generalizations. Block generalizations of MGS-SVL (Algorithm A.7) and MGS-LTS (Algorithm A.8) are quite straightforward. In [4], BMGS-SVL appears as MGS3 and BMGS_H with IntraOrthos MGS-SVL (Algorithm B.3) and HouseQR (i.e., Householder-based QR, as in, e.g., [23]), 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 IntraOrthos 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 [61, Figure 3]. Instead of the implied square root in line 7 of Algorithms B.5 and B.6, 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 A.9 and A.10 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 B.5 and B.6 can be replaced by an IntraOrtho or another Cholesky factorization; we opt for an IntraOrtho in line 16 of Algorithms A.9 and A.10.

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

2.2.3. 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. 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 $\boldsymbol{\mathcal{X}}$ is fixed a priori. However, a stability analysis of DBMGS would greatly inform that of adaptive s-step algorithms [10].

3. Skeleton stability in terms of muscle stability: overview. We primarily focus on loss of orthogonality as a measure of stability. In more detail, 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, 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

$$\left\| I_n - \bar{\boldsymbol{\mathcal{Q}}}^T \bar{\boldsymbol{\mathcal{Q}}} \right\|, \tag{3.1}$$

for some norm $\|\cdot\|$. We take $\|\cdot\|$ to be the Euclidean norm, unless otherwise noted.

Regarding ε as the machine epsilon for standard (double) precision, we seek bounds on (3.1) in terms of $\mathcal{O}(\varepsilon)$ and $\kappa(\boldsymbol{\mathcal{X}})$, which we define as the 2-norm condition number of $\boldsymbol{\mathcal{X}}$ or ratio between its largest and smallest singular values. An algorithm is *unconditionally stable* if (3.1) is bounded by $\mathcal{O}(\varepsilon)$. In contrast, an algorithm is *conditionally stable* if the bound on (3.1) is in terms of $\kappa(\boldsymbol{\mathcal{X}})$ or only holds with some condition on $\kappa(\boldsymbol{\mathcal{X}})$.

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

$$\frac{\left\| \bar{\boldsymbol{\mathcal{Q}}}\bar{\mathcal{R}} - \boldsymbol{\mathcal{X}} \right\|}{\|\boldsymbol{\mathcal{X}}\|}. \tag{3.2}$$

We take for granted that all algorithms considered here (with the exception of, perhaps, CGSS+rpl and BCGSS+rpl and 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*,

$$\frac{\left\| \boldsymbol{\mathcal{X}}^T\boldsymbol{\mathcal{X}} - \bar{\mathcal{R}}^T\bar{\mathcal{R}} \right\|}{\|\boldsymbol{\mathcal{X}}\|^2}, \tag{3.3}$$

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

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

(Q.1) Which skeletons are stable when composed with an unconditionally stable muscle?

(Q.2) Can we determine minimal conditions on the muscle such that a particular skeleton is stable?

To answer either question, it will be helpful to keep in mind the stability properties of different muscles, summarized in Table 3.1. Additionally, we summarize known and conjectured results for block algorithms in Table 3.2. For the specific assumptions on the muscles that lead to these stability bounds, see the exposition in Section 4.

TABLE 3.1

Upper bounds on deviation from orthogonality of the $\boldsymbol{Q}$ factor for various `IntraOrthos`*, 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`	$\left\|I - \bar{\boldsymbol{Q}}^T\bar{\boldsymbol{Q}}\right\|_2$ upper bound	Assumption on $\kappa(\boldsymbol{X})$	Reference(s)
`CGS`	$\mathcal{O}(\varepsilon)\,\kappa^{n-1}(\boldsymbol{X})$	$\mathcal{O}(\varepsilon)\,\kappa(\boldsymbol{X}) < 1$	[33]
`CGS-P`	$\mathcal{O}(\varepsilon)\,\kappa^2(\boldsymbol{X})$	$\mathcal{O}(\varepsilon)\,\kappa^2(\boldsymbol{X}) < 1$	[50]
`CholQR`	$\mathcal{O}(\varepsilon)\,\kappa^2(\boldsymbol{X})$	$\mathcal{O}(\varepsilon)\,\kappa^2(\boldsymbol{X}) < 1$	[59]
`MGS`	$\mathcal{O}(\varepsilon)\,\kappa(\boldsymbol{X})$	$\mathcal{O}(\varepsilon)\,\kappa(\boldsymbol{X}) < 1$	[7]
`MGS-SVL`	$\mathcal{O}(\varepsilon)\,\kappa(\boldsymbol{X})$	$\mathcal{O}(\varepsilon)\,\kappa(\boldsymbol{X}) < 1$	[4]
`MGS-LTS`	$\mathcal{O}(\varepsilon)\,\kappa(\boldsymbol{X})^\dagger$	$\mathcal{O}(\varepsilon)\,\kappa(\boldsymbol{X}) < 1^\dagger$	conjecture
`MGS-CWY`	$\mathcal{O}(\varepsilon)\,\kappa(\boldsymbol{X})^\dagger$	$\mathcal{O}(\varepsilon)\,\kappa(\boldsymbol{X}) < 1^\dagger$	conjecture
`MGS-ICWY`	$\mathcal{O}(\varepsilon)\,\kappa(\boldsymbol{X})^\dagger$	$\mathcal{O}(\varepsilon)\,\kappa(\boldsymbol{X}) < 1^\dagger$	conjecture
`CholQR+`	$\mathcal{O}(\varepsilon)$	$\mathcal{O}(\varepsilon)\,\kappa^2(\boldsymbol{X}) < 1$	[59]
`CGS+`	$\mathcal{O}(\varepsilon)$	$\mathcal{O}(\varepsilon)\,\kappa(\boldsymbol{X}) < 1$	conjecture
`CGSI+`	$\mathcal{O}(\varepsilon)$	$\mathcal{O}(\varepsilon)\,\kappa(\boldsymbol{X}) < 1$	[1, 5, 21]
`CGSS+`	$\mathcal{O}(\varepsilon)$	$\mathcal{O}(\varepsilon)\,\kappa(\boldsymbol{X}) < 1$	[11, 30]
`CGSI+LS`	$\mathcal{O}(\varepsilon)^\dagger$	$\mathcal{O}(\varepsilon)\,\kappa(\boldsymbol{X}) < 1^\dagger$	conjecture
`MGS+`	$\mathcal{O}(\varepsilon)$	$\mathcal{O}(\varepsilon)\,\kappa(\boldsymbol{X}) < 1$	[31, 20]
`MGSI+`	$\mathcal{O}(\varepsilon)$	$\mathcal{O}(\varepsilon)\,\kappa(\boldsymbol{X}) < 1$	[30, 19]
`ShCholQR++`	$\mathcal{O}(\varepsilon)$	$\mathcal{O}(\varepsilon)\,\kappa(\boldsymbol{X}) < 1$	[18]
`CGSS+rpl`	$\mathcal{O}(\varepsilon)^\dagger$	none†	conjecture
`HouseQR`	$\mathcal{O}(\varepsilon)$	none	[28, Sec. 19.3]
`GivensQR`	$\mathcal{O}(\varepsilon)$	none	[28, Sec. 19.6]
`TSQR`	$\mathcal{O}(\varepsilon)$	none	[38]

TABLE 3.2

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

`BGS`	$\left\|I - \bar{\boldsymbol{\mathcal{Q}}}^T\bar{\boldsymbol{\mathcal{Q}}}\right\|_2$ upper bound	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`	$\mathcal{O}(\varepsilon)\,\kappa^2(\boldsymbol{\mathcal{X}})^\dagger$	$\mathcal{O}(\varepsilon)\,\kappa^2(\boldsymbol{\mathcal{X}}) < 1$	conjecture
`BMGS`	$\mathcal{O}(\varepsilon)\,\kappa(\boldsymbol{\mathcal{X}})$	$\mathcal{O}(\varepsilon)\,\kappa(\boldsymbol{\mathcal{X}}) < 1$	[31]; here
`BMGS-SVL`	$\mathcal{O}(\varepsilon)\,\kappa(\boldsymbol{\mathcal{X}})$	$\mathcal{O}(\varepsilon)\,\kappa(\boldsymbol{\mathcal{X}}) < 1$	[4]
`BMGS-LTS`	$\mathcal{O}(\varepsilon)\,\kappa(\boldsymbol{\mathcal{X}})^\dagger$	$\mathcal{O}(\varepsilon)\,\kappa(\boldsymbol{\mathcal{X}}) < 1^\dagger$	conjecture
`BMGS-CWY`	$\mathcal{O}(\varepsilon)\,\kappa(\boldsymbol{\mathcal{X}})^\dagger$	$\mathcal{O}(\varepsilon)\,\kappa(\boldsymbol{\mathcal{X}}) < 1^\dagger$	conjecture
`BMGS-ICWY`	$\mathcal{O}(\varepsilon)\,\kappa(\boldsymbol{\mathcal{X}})^\dagger$	$\mathcal{O}(\varepsilon)\,\kappa(\boldsymbol{\mathcal{X}}) < 1^\dagger$	conjecture
`BCGSI+`	$\mathcal{O}(\varepsilon)$	$\mathcal{O}(\varepsilon)\,\kappa(\boldsymbol{\mathcal{X}}) < 1$	[5]
`BCGSS+rpl`	$\mathcal{O}(\varepsilon)^\dagger$	none†	conjecture

To demonstrate stability properties for multiple skeleton-muscle combinations at once, we build heat-maps for the following test matrices $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{m\times ps}$, where $m = 10000$ (number of rows), $p = 50$ (number of block vectors), and $s = 10$ (columns per block vector). Additional matrix properties are given in Table 3.3. 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.[5] Our code is publicly available at https://github.com/katlund/BlockStab or by direct request from the authors.

- `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.
- `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`: $\boldsymbol{\mathcal{X}} = \boldsymbol{\mathcal{U}}\Sigma\boldsymbol{\mathcal{V}}^T$, where $\boldsymbol{\mathcal{U}}$ and $\boldsymbol{\mathcal{V}}$ are random unitary matrices and the diagonal of Σ is geometric sequence from 1 to 10^{-20}. The first and 25th columns of $\boldsymbol{\mathcal{X}}$ are the same, and 35th column is identically zero.
- `stewart_extreme`: $\boldsymbol{\mathcal{X}} = \boldsymbol{\mathcal{U}}\Sigma\boldsymbol{\mathcal{V}}^T$, where $\boldsymbol{\mathcal{U}}$ and $\boldsymbol{\mathcal{V}}$ are random unitary matrices and the first half of the diagonal of Σ is geometric sequence from 1 to 10^{-10}, while the second half is identically zero.

Note that while `stewart` and `stewart_extreme` are taken from [53], matrices like these have been used to study stability properties in Gram-Schmidt algorithms perhaps as early as Hoffman [30]. 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. To demonstrate the effect of replacement, `CGSS+` has a replacement tolerance $\delta = 0$, while `CGSS+rpl` has $\delta = 100$, which Stewart [53] notes is relatively

[5] http://cluster.karlin.mff.cuni.cz/.

TABLE 3.3

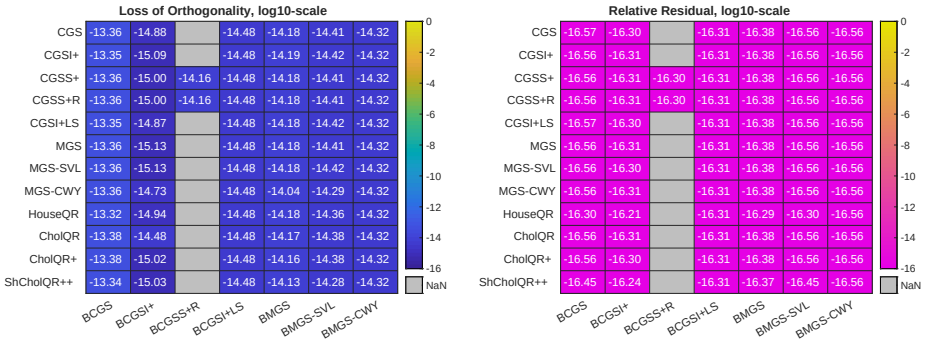
Matrix properties. Note that singular values are computed via `svd` *after* $\boldsymbol{\mathcal{X}}$ *is explicitly formed. When exact values are known, which is the case with the smallest singular value of* `rank_def`*, and all the singular values of* `stewart` *and* `stewart_extreme`*, those values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

"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 3.1 and 3.2, there is little difference overall among the algorithms for either matrix. The only noticeable differences arise for `rand_uniform` (Figure 3.1) 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 3.2; for all methods they perform slightly worse.

FIG. 3.1. *Measurements for* `rand_uniform`



3.2. `rank_def`**.** In Figure 3.3, 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`, but muscles `CGS`, `CGSS+`, `CGSS+rpl`, `MGS`, and `MGS-SVL` are the best.

FIG. 3.2. *Measurements for* `rand normal`.

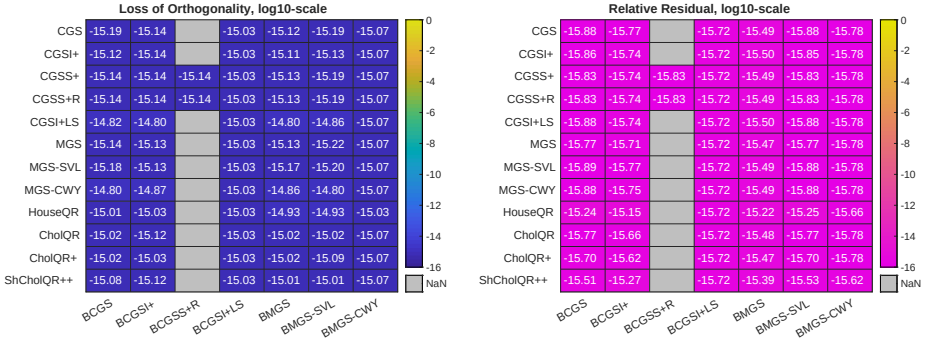
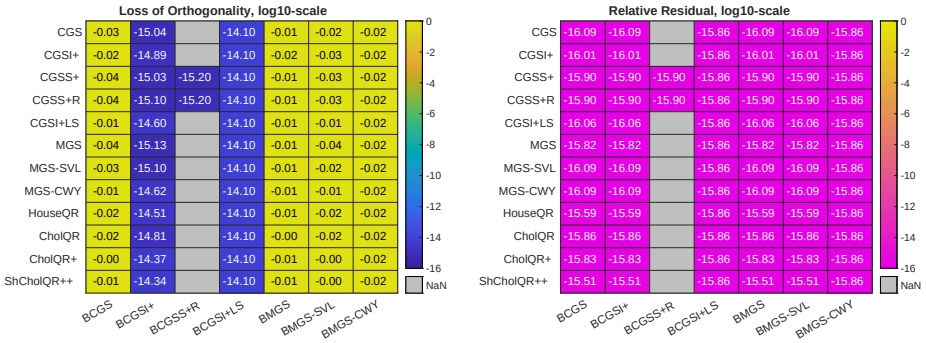


FIG. 3.3. *Measurements for* `rank def`.

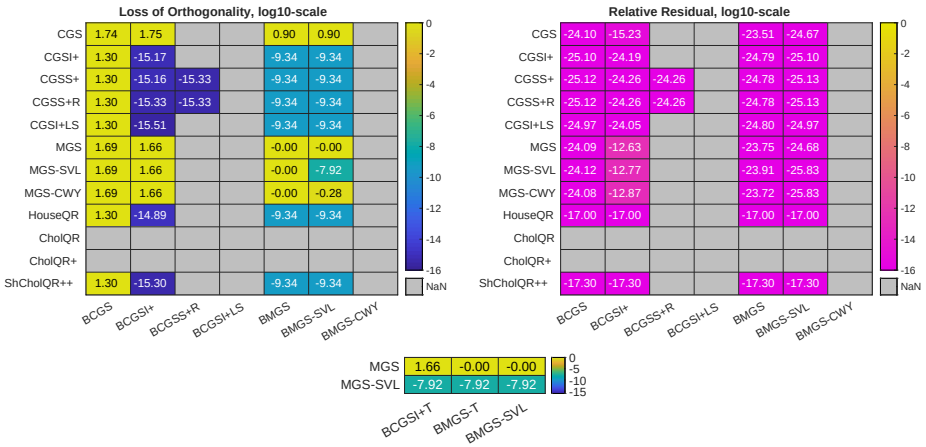


3.3. `laeuchli`**.** The Läuchli matrix reveals many interesting nuances. We first note in Figure 3.4 that for all algorithms that run to completion, except `BCGSI+` with `CGS`, `MGS`, or `MGS-SVL`, the relative residual exceeds double precision. This is due to the nature of the Läuchli matrix itself: 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 there is an easy fix: 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 3.4 demonstrates the outcome, with a "T" suffix denoting the altered algorithms. With the T-fix, both `BCGSI+` $\circ$ `MGS-SVL` and `BMGS` $\circ$ `MGS-SVL` perform as well as `BMGS-SVL` $\circ$ `MGS-SVL`; in fact, `BCGSI+` $\circ$ `MGS-SVL` matches the others in residual too. (Note that `BCGS` with the T-fix would reproduce `BMGS-SVL`.)

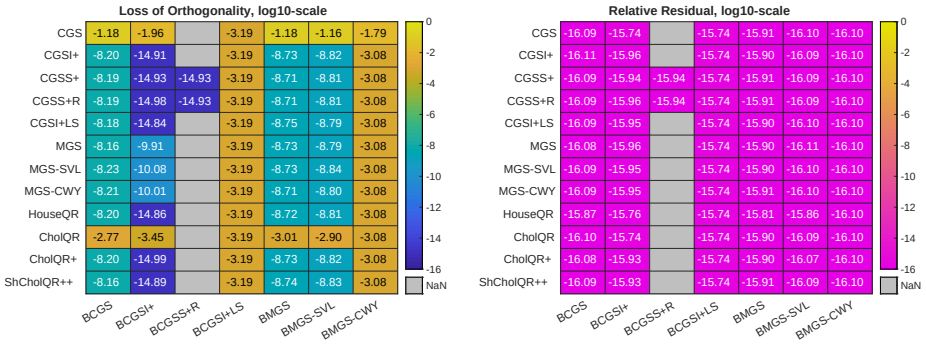
3.4. `monomial`**.** The `monomial` matrix also reveals a few interesting properties. In Figure 3.5 we now see that all Cholesky-based methods can run, but they all lose quite of bit orthogonality. `CGS` struggles especially across the board. We again see that `BCGSI+` $\circ$ `MGS-SVL` does not perform as well as other variants of `BCGSI+`; however, the T-fix will not work here, because `BCGSI+` $\circ$ `MGS-SVL` is already as stable as `BMGS-SVL` $\circ$ `MGS-SVL`.

FIG. 3.4. *Measurements for* `laeuchli`.



Looking closely at Algorithm A.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, we reduce the loss of orthogonality to nearly the same level *regardless of the muscle*. Indeed, the same also holds for `BCGSI+` on `laeuchli`, if we reorthogonalize the first block vector. We do not report these tests here, but see a related discussion in Section 4.2.

FIG. 3.5. *Measurements for* `monomial`.



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., [3, 9, 29]. The `s-step` matrix is poorly conditioned, as expected, because a monomial is used to build the basis. The `newton` matrix simulates what would happen when Newton polynomials are used instead, which lead 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.

As noted in Section 2.1.3, `BCGSS+rpl` is not suitable for high-performance computing due to the high communication costs of repeatedly checking norms. These results strongly reinforce that the right direction for improving s-step methods lies

not in a more stable QR factorization but rather in a more stable polynomial.

FIG. 3.6. *Measurements for s-step*

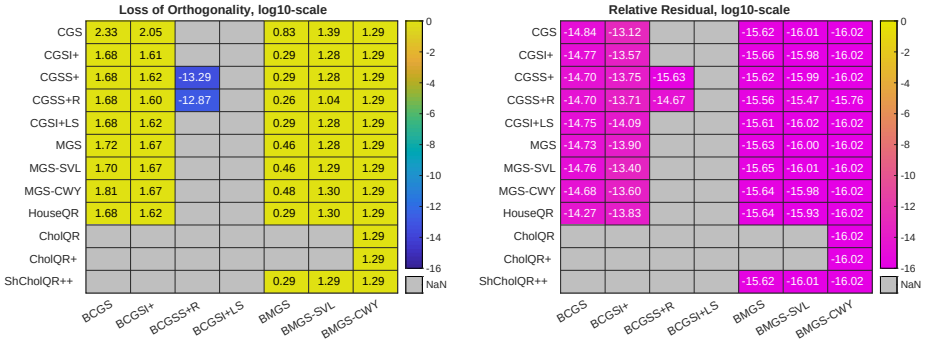
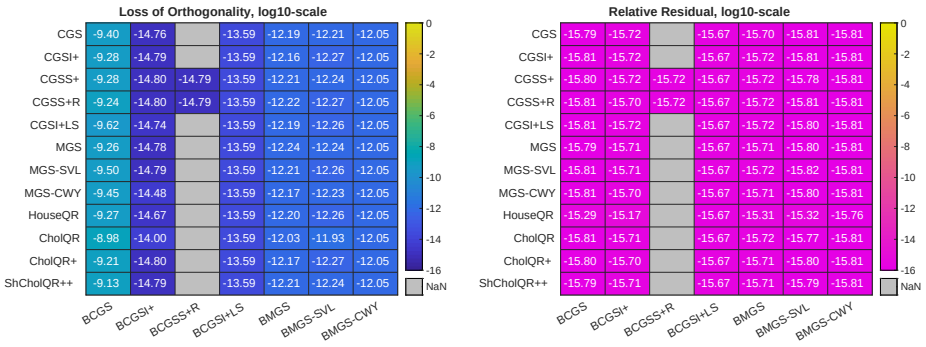


FIG. 3.7. *Measurements for newton*



3.6. `stewart`. Stewart [53] purposefully designed his algorithms to be robust for pathologically bad matrices, and the results in Figure 3.8 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` ∘ `CGSS+rpl`, its variant without replacement, and `BCGSI+` with Stewart's `IntraOrthos` and `HouseQR`.

3.7. `stewart_extreme`. This matrix in our collection demonstrates the utility of aggressive replacement. Figure 3.9 collects the results for `stewart_extreme`.

For both `BCGSI+` and `BCGSS+rpl` we can see how aggressive replacement can make a significant difference: for `BCGSI+`, the loss of orthogonality is reduced by nearly 10 orders of magnitude, while the residual increases by only 2 orders; for `BCGSS+rpl`, we decrease loss of orthogonality by two orders, but gain only one order in residual.

Reorthogonalizing the first block vector of `BCGSI+` helps for some muscles in this scenario. 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 [18] 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+`∘`ShCholQR++`

FIG. 3.8. *Measurements for stewart*

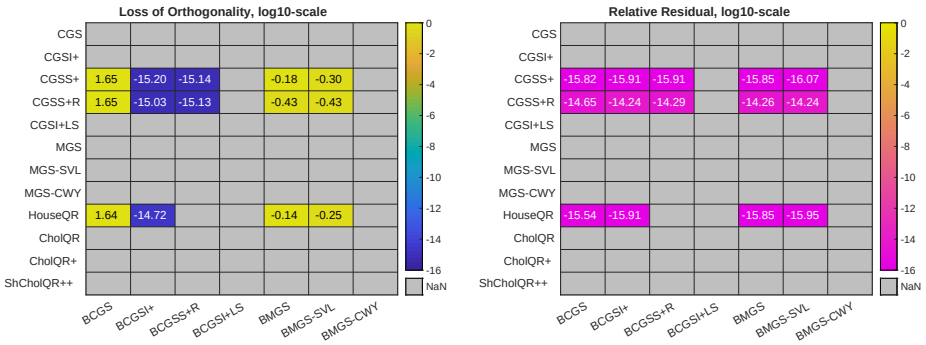
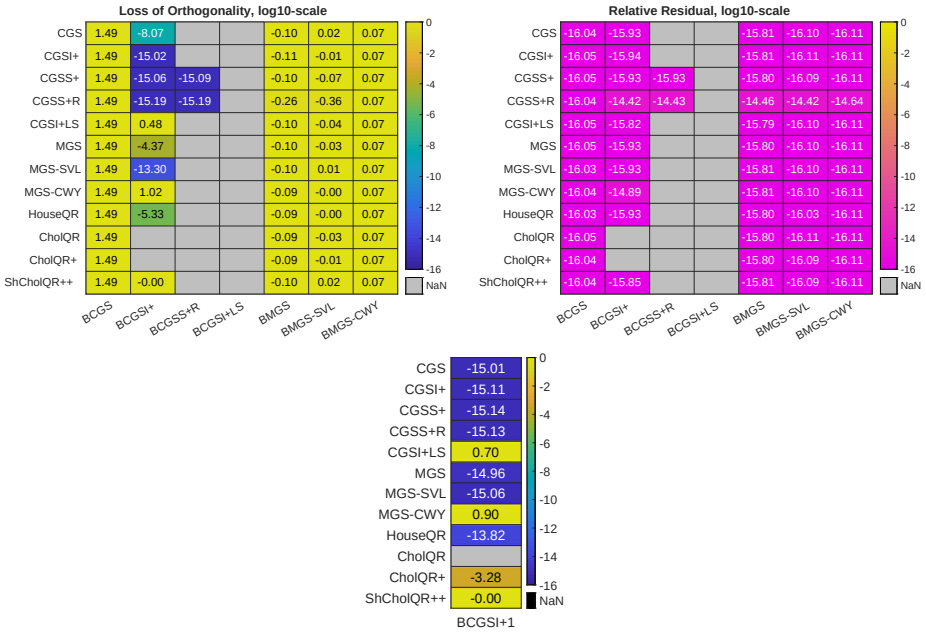


FIG. 3.9. *Measurements for stewart extreme*



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.

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 [50]; 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 2019a on an Lenovo Thinkpad X1 Carbon, Generation 6 running 64-bit Windows 10 with an Intel Core i7-8550U CPU and 16GB of RAM. We provide the command-line calls for each figure in Appendix C.

4.1. `BCGS`**.** Using similar techniques as for `CGS`, it is not hard to show that the loss of orthogonality in `BCGS` $\circ$ `IntraOrtho` can be bounded as

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

Figure 4.1 shows the loss of orthogonality and relative Cholesky residual for `CGS` and `CGS-P`. It's clear that as the relative Cholesky residual departs from $\mathcal{O}(\varepsilon)$, the loss of orthogonality follows suit. We observe similar behavior for `BCGS` and the Pythagorean variants in 4.2. We only show the results for when `IntraOrtho` = `HouseQR`, but they hold similarly for other muscles.

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 is the subject of ongoing work. Necessary requirements are that $[\bar{\boldsymbol{Q}}, \bar{R}] =$ `IntraOrtho`$(\boldsymbol{X})$ approximates the Cholesky decomposition of $\boldsymbol{X}^T\boldsymbol{X}$ when $\mathcal{O}(\varepsilon)\,\kappa^2(\boldsymbol{X}) < 1$, and that the `IntraOrtho` has $\mathcal{O}(\varepsilon)$ relative residual:

$$\bar{R}^T\bar{R} = \boldsymbol{X}^T\boldsymbol{X} + E, \quad \|E\| \leq \mathcal{O}(\varepsilon)\,\|\boldsymbol{X}\|^2, \tag{4.1}$$

and

$$\bar{\boldsymbol{Q}}\bar{R} = \boldsymbol{X} + \boldsymbol{D}, \quad \|\boldsymbol{D}\| \leq \mathcal{O}(\varepsilon)\,\|\boldsymbol{X}\|. \tag{4.2}$$

Two results follow immediately from these assumptions. The first is a classic Weyl-type bound (see, e.g., [43, Theorem 10.3.1]) on the norm of $\bar{R}^{-1}$:

$$\left\|\bar{R}^{-1}\right\| = \frac{1}{\sigma_{\min}(\bar{R})} \leq \frac{1}{\sqrt{\sigma^2_{\min}(\boldsymbol{X}_1)(1 - \mathcal{O}(\varepsilon)\,\kappa^2(\boldsymbol{X}_1))}}. \tag{4.3}$$

The second is a second-order loss of orthogonality bound:

$$\begin{aligned}
& \bar{R}^T\bar{\boldsymbol{Q}}^T\bar{\boldsymbol{Q}}\bar{R} = \boldsymbol{X}^T\boldsymbol{X} + \underbrace{\boldsymbol{X}^T\boldsymbol{D} + \boldsymbol{D}^T\boldsymbol{X} + \boldsymbol{D}^T\boldsymbol{D}}_{:=F}, \text{ by (4.2)} \\
\Leftrightarrow \quad & \bar{R}^T(I - \bar{\boldsymbol{Q}}^T\bar{\boldsymbol{Q}})\bar{R} = \bar{R}^T\bar{R} - \boldsymbol{X}^T\boldsymbol{X} - F = E - F, \text{ by (4.2)} \\
\Leftrightarrow \quad & I - \bar{\boldsymbol{Q}}^T\bar{\boldsymbol{Q}} = \bar{R}^{-T}(E - F)\bar{R}^{-1}
\end{aligned}$$

Applying the error bounds in (4.1) and (4.2), along with (4.3) gives

$$\begin{aligned}
\left\|I - \bar{\boldsymbol{Q}}^T\bar{\boldsymbol{Q}}\right\| &\leq \left\|\bar{R}^{-1}\right\|^2 (\|E\| + \|F\|) \\
&\leq \frac{\mathcal{O}(\varepsilon)\,\|\boldsymbol{X}\|^2}{\sigma^2_{\min}(\boldsymbol{X})(1 - \mathcal{O}(\varepsilon)\,\kappa^2(\boldsymbol{X}))} \\
&\leq \mathcal{O}(\varepsilon)\,\kappa^2(\boldsymbol{X})\frac{1}{1 - \mathcal{O}(\varepsilon)\,\kappa^2(\boldsymbol{X})}.
\end{aligned} \tag{4.4}$$

Therefore in general, (4.1) and (4.2) are enough to ensure an $\mathcal{O}(\varepsilon)\,\kappa^2(\boldsymbol{X})$ loss of orthogonality. All algorithms we consider satisfy (4.2).

FIG. 4.1. *glued κ-plots for P-variants of CGS.*

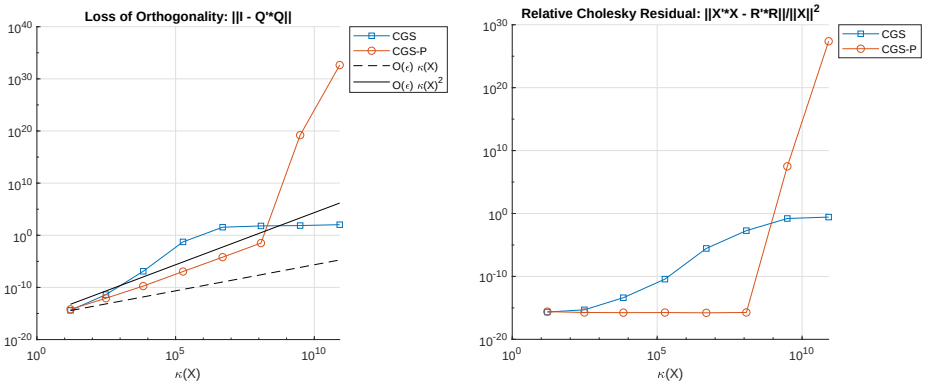
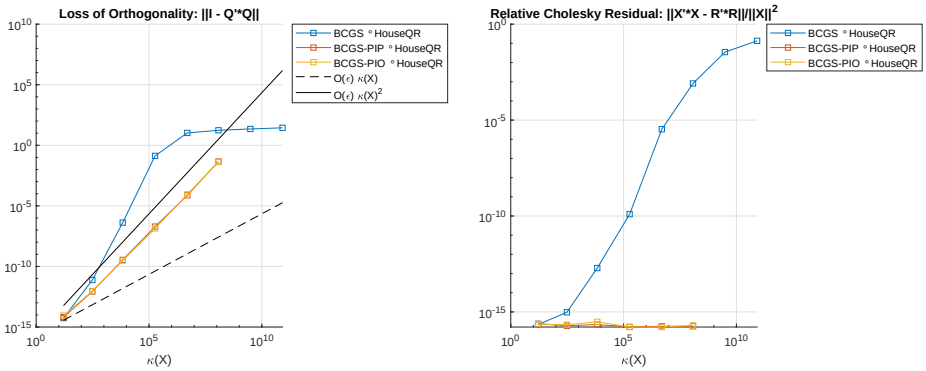


FIG. 4.2. *glued κ-plots for P-variants of BCGS.*



4.2. `BCGSI+`**.** Barlow and Smoktunowicz prove an $\mathcal{O}(\varepsilon)$ loss of orthogonality for `BCGSI+` and an unconditionally stable `IntraOrtho`, as long as $\mathcal{O}(\varepsilon)\,\kappa(\boldsymbol{\mathcal{X}}) < 1$ [5]. 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

$$\boldsymbol{B} = \boldsymbol{\mathcal{U}}\boldsymbol{S}_{\boldsymbol{B}} + \boldsymbol{Q}_{\boldsymbol{B}} R_{\boldsymbol{B}} \text{ and } \boldsymbol{\mathcal{U}}^T \boldsymbol{Q}_{\boldsymbol{B}} \approx 0. \tag{4.5}$$

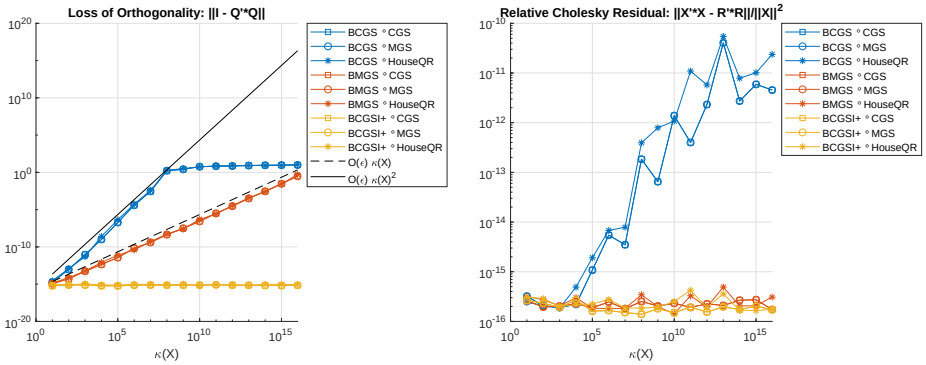
This subproblem is present at every step of `BCGSI+`, as long as the previous step has produced a near-left orthogonal matrix $\boldsymbol{\mathcal{U}}$. With the unconditionally stable `IntraOrtho` guaranteeing that $\boldsymbol{Q}_1$ (see line 2 in Algorithm A.4) is near left-orthogonal, induction 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 $\mathcal{O}(\varepsilon)\,\|\boldsymbol{B}\|\,\left\|\bar{R}_B^{-1}\right\| < 1$, where $\bar{R}_{\boldsymbol{B}}$ is the computed version of $R_{\boldsymbol{B}}$ from the subproblem (4.5); see [5, 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 4.3 displays standard κ-plots for `BCGSI+` in comparison the `BCGS` and `BMGS` skeletons 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 3.3), it is clear that we may need an unconditionally stable `IntraOrtho` for some tough matrices.

FIG. 4.3. *standard κ-plots for BCGSI+. in comparison to those of BCGS and BMGS.*



4.3. `BCGSS+rpl`. Although Stewart's development of `BCGSS+rpl` [53] is largely driven by stability considerations, a rigorous proof is lacking. We conjecture that for any $\boldsymbol{\mathcal{X}}$, even those with rank deficiency, 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 [53] 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. One-sync `BCGSI+`. Rigorous stability analysis for the low-sync algorithms from [55], especially their block versions, remains open. We can, however, formulate some conjectures.

Figures 4.4 and 4.5 indicate that `BCGSI+LS` remains relatively stable, especially its relative Cholesky residual. The loss of orthogonality begins to deviate gradually from $\mathcal{O}(\varepsilon)$ in Figure 4.3 and somewhat more drastically for the more challenging `monomial` matrices in Figure 4.5. It is not possible to correct this by just reorthogonalizing the first block vector (cf. discussion for `BCGSI+` in Section 3.4), and we suspect the deviation from $\mathcal{O}(\varepsilon)$ is due to the normalization lagging.

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 [31]. They focus on `BMGS` $\circ$ `MGS` and `BMGS` $\circ$ `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 A.6 can be written more succinctly (in exact arithmetic) as

$$\boldsymbol{W} = (I - \boldsymbol{Q}_k \boldsymbol{Q}_k^T) \cdots (I - \boldsymbol{Q}_1 \boldsymbol{Q}_1^T) \boldsymbol{X}_{k+1}. \tag{4.6}$$

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:

$$\left\| I - \bar{\boldsymbol{Q}}^T \bar{\boldsymbol{Q}} \right\|_2 \leq \mathcal{O}(\varepsilon)\, C\, \|\boldsymbol{\mathcal{X}}\|_{\mathrm{F}} \left\| \mathcal{R}^{-1} \right\|_2 \leq \mathcal{O}(\varepsilon)\, \kappa^2(\boldsymbol{\mathcal{X}}), \tag{4.7}$$

FIG. 4.4. *standard κ-plots for low-sync* BCGSI+*, in comparison to those of* BCGS *and* BCGSI+

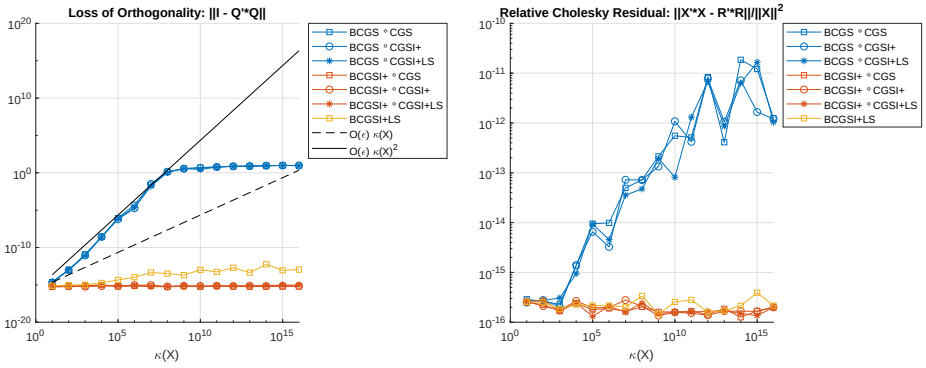
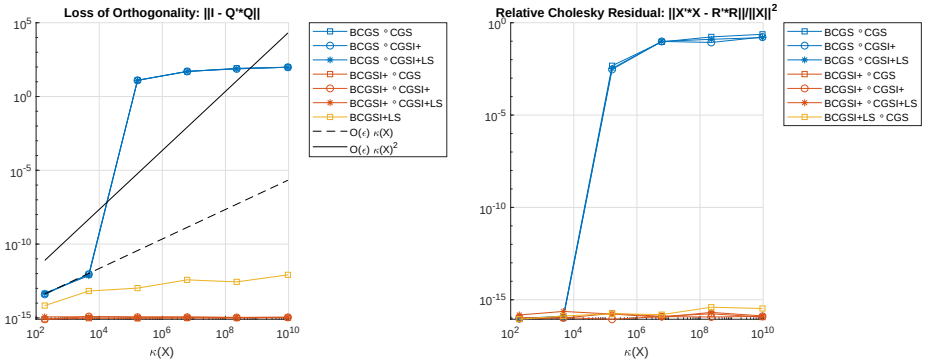


FIG. 4.5. *monomial κ-plots for low-sync* BCGSI+*, in comparison to those of* BCGS *and* BCGSI+



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

$$\left\| I - \bar{\mathcal{Q}}^T \bar{\mathcal{Q}} \right\|_2 \leq \mathcal{O}(\varepsilon)\, \kappa(\boldsymbol{\mathcal{X}}). \tag{4.8}$$

Barlow [4] 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 (4.8) holds for any `IntraOrtho` that is unconditionally stable.

At the top of page 1062 of [31], the authors directly apply `MGS` bounds from [7] 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 A.6, and $R_{\boldsymbol{W}_k}$ its upper triangular factor from an exact QR decomposition. The intermediate constants $\|\boldsymbol{W}_k\|_{\mathrm{F}} \left\|R_{\boldsymbol{W}_k}^{-1}\right\|_2$ are absent for any unconditionally stable `IntraOrtho`. Tracking C all the way through the proof of [31, Theorem 4.1] (from which we have concluded (4.7)) verifies that, without changing any other lines in their text, we would obtain (4.8) for `BMGS` ∘ `IntraOrtho`, assuming `IntraOrtho` is unconditionally stable.

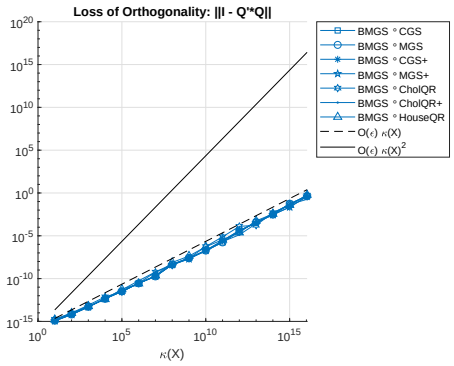
We remark that the present work is the first to state bounds for `BMGS`∘`IntraOrtho`, despite the ubiquity of `BMGS`∘`HouseQR` in practice. More precise bounds for particular `IntraOrtho` remain open, but can be easily derived on a case-by-case basis from the

transparent analysis in [31]. We do not think the framework developed by Barlow [4] to be suitable for studying BMGS.

Because BMGS ∘ HouseQR behaves essentially like MGS, we conjecture that results analogous to those by Paige, Rozložník, and Strakoš [40] hold for block GMRES, which is usually implemented with BMGS ∘ HouseQR. To be more specific, it is likely that the loss of orthogonality in BMGS ∘ HouseQR is tolerable when the end application is the solution of a linear system via the GMRES method.

Figure 4.6 demonstrates the $\mathcal{O}(\varepsilon)\,\kappa(\boldsymbol{\mathcal{X}})$ loss of orthogonality for a variety of muscles and their reorthogonalized variants. Although we do not observe behavior like (4.7) here, the laeuchli and monomial studies in Sections 3.3 and 3.4 affirm that an unconditionally stable muscle is really necessary for $\mathcal{O}(\varepsilon)\,\kappa(\boldsymbol{\mathcal{X}})$ loss of orthogonality.

FIG. 4.6. *κ-plots for BMGS*



4.6. BMGS T-variants. To date, we are only aware of Barlow's stability analysis on MGS-SVL and BMGS-SVL for low-synchronization BMGS variants [4]. 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. [55].

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, [4, Section 4]. The following quantities play key roles:

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

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 [4]), 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.,

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

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

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

$$\|\Delta_{\mathcal{TS}}\|_{\mathrm{F}} \leq \mathcal{O}(\varepsilon), \tag{4.9}$$

$$\|\Delta_{\bar{\boldsymbol{Q}}\bar{R}}\|_{\mathrm{F}} \leq \mathcal{O}(\varepsilon)\,\|\boldsymbol{\mathcal{X}}\|_{\mathrm{F}}, \text{ and} \tag{4.10}$$

$$\|\Gamma_{\mathcal{TR}}\|_{\mathrm{F}} \leq \mathcal{O}(\varepsilon)\,\|\boldsymbol{\mathcal{X}}\|_{\mathrm{F}}, \tag{4.11}$$

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

$$\Delta_{TS} := \bar{T}S - I_s, \quad \|\Delta_{TS}\|_{\mathrm{F}} \leq \mathcal{O}(\varepsilon); \tag{4.12}$$

$$\Delta_{\bar{\boldsymbol{Q}}\bar{R}} := \bar{\boldsymbol{Q}}\bar{R} - \boldsymbol{X}, \quad \left\|\Delta_{\bar{\boldsymbol{Q}}\bar{R}}\right\|_{\mathrm{F}} \leq \mathcal{O}(\varepsilon)\,\|\boldsymbol{X}\|_{\mathrm{F}}; \text{ and} \tag{4.13}$$

$$\Gamma_{TR} := (I_s - \bar{T})\bar{R}, \quad \|\Gamma_{TR}\|_{\mathrm{F}} \leq \mathcal{O}(\varepsilon)\,\|\boldsymbol{X}\|_{\mathrm{F}} \tag{4.14}$$

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

At first glance through Appendix B, 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 IntraOrthos 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

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

and

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

Table 3.1 clarifies that all IntraOrthos except CGS, MGS, CholQR, and CholQR+ (because of the restriction on $\kappa(\boldsymbol{X})$) satisfy the triad (4.12)-(4.14). Barlow demonstrates directly that MGS-SVL meets (4.12)-(4.14) (see [4, Section 4.2]), as long as $\bar{R}$ is assumed to be nonsingular. Numerical experiments indicate that MGS-LTS satisfies (4.12)-(4.14) under similar assumptions, and 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.

Equation 4.10 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 (4.9) - (4.11) imply

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

Recalling (from, e.g., [7]) that $\|\boldsymbol{\mathcal{X}}\|_{\mathrm{F}}\left\|\mathcal{R}^{-1}\right\| \leq O(\kappa(\boldsymbol{\mathcal{X}}))$, 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

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

and thanks to norm equivalence, we obtain the bounds reported in Table 3.2.

Figure 4.7 displays standard κ-plots for all the MGS variants in comparison to MGS itself; there are hardly any perceptible differences in behavior. In Figure 4.8, we focus on just the BMGS-SVL and BMGS-LTS pair of skeletons in comparison to BMGS and note that the behavior is nearly identical for the pair BMGS-LTS and BMGS-ICWY. Only for the monomial κ-plots in Figure 4.8(b) do we begin to observe some deviation from $\mathcal{O}(\varepsilon)\,\kappa(\boldsymbol{\mathcal{X}})$. It is likely that, as with BCGSI+LS (cf. Section 4.4), normalization lagging in BMGS-CWY begins to ruin the quality of orthogonality.

FIG 4.7 *standard κ-plots for low-sync variants of* MGS

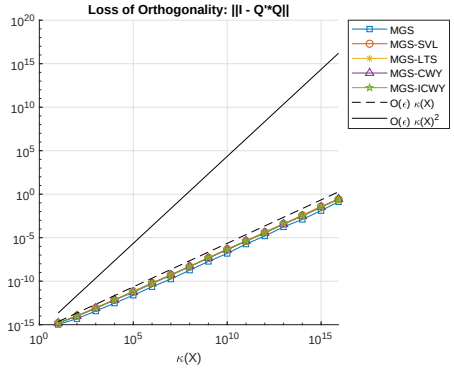
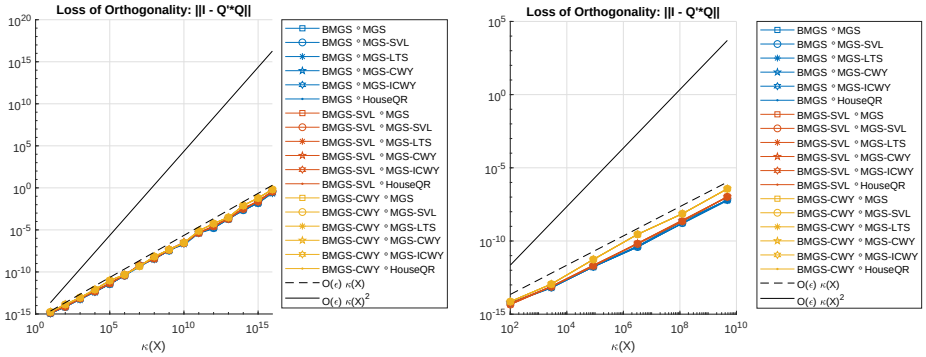


FIG 4.8 *κ-plots for low-sync variants of* BMGS. *(a) standard κ-plot. (b)* monomial *κ-plot.*



(a) (b)

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 [62], 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 3.1).

Building on this work, in [63], the authors propose using the `mCholQR` of [62] 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` $\circ$ `mCholQR` variants can be maintained to the same level of `MGS` if reorthogonalization is used.

We note also that in [64], 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 [60] on the Trilinos Library presents experiments with communication-avoiding (s-step) GMRES using `CGS` $\circ$ `CholQR` as well as a "low-synch CGS" with CholQR2 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. [61]. 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. [32]. 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 [35] and the Contour Integral Spectrum Slicing method (CISS) of Sakurai and Suguria [47].

For prototype algorithms and testing routines for stability conjectures, we direct the reader to the 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 is a stringent requirement for a software pipeline. This is especially true if eigenvalue estimates are needed anywhere [52]. Lately, Krylov subspace methods (KSMs), and particularly block KSMs, form an important component in matrix function approximations, which often use eigenvalue estimations to accelerate convergence [14, 16, 15, 17, 27]. Other applications where reliable eigenvalue estimates are needed include Krylov subspace recycling and spectral deflation [41, 42].

This work constitutes a crucial first step towards a fuller understanding of the stability properties of block KSMs. The natural next step is an examination of the least-squares problem for block upper Hessenberg matrices. Work by Gutknecht and Schmelzer [26] may prove relevant, as well as the low-rank update formulation devised in [17] for matrix functions.

It is also worth exploring whether the low-sync BGS variants– `BCGSI+LS`, `BMGS-SVL`,

BMGS-LTS, BMGS-CWY, BMGS-ICWY– coupled with, e.g., TSQR or ShCholQR++, provide the same communication benefits as BCGS ∘ 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. We conjecture that block and s-step GMRES are backward stable if the block orthogonalization scheme is as stable as MGS and as long as some constraint on the condition number of the individual blocks is satisfied; we leave further investigation as future work.

Appendix A. Pseudocode for skeletons. For the following algorithms, recall that $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{m\times n}$, $m \gg n$, is partitioned into a set of p block vectors, each of size $m \times s$.

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

and that muscles operate on a single block vector $\boldsymbol{X} \in \mathbb{R}^{m\times s}$. Entries of block matrices $\mathcal{R}$ and $\mathcal{T}$ are denoted by their non-calligraphic counterparts R_{jk} and T_{jk}, respectively.

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,

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

Algorithms A.2 and A.3 are both variants of BCGS-P. "IP" stands for "inner product," while "IO" stands for "intra-orthogonalization," each referring to the different representations of the block Pythagorean theorem (Theorem 2.1).

Algorithm A.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** $k = 1, \ldots, p-1$ **do**
4: $\quad \mathcal{R}_{1:k,k+1} = \boldsymbol{\mathcal{Q}}_{1:k}^T \boldsymbol{X}_{k+1}$
5: $\quad \boldsymbol{W} = \boldsymbol{X}_{k+1} - \boldsymbol{\mathcal{Q}}_{1:k}\mathcal{R}_{1:k,k+1}$
6: $\quad [\boldsymbol{Q}_{k+1}, R_{k+1,k+1}] = \texttt{IntraOrtho}(\boldsymbol{W})$
7: **end for**
8: **return** $\boldsymbol{\mathcal{Q}} = [\boldsymbol{Q}_1, \ldots, \boldsymbol{Q}_p]$, $\mathcal{R} = (R_{jk})$

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

Allocate memory for $\boldsymbol{\mathcal{Q}}$ and $\mathcal{R}$
$[\boldsymbol{Q}_1, R_{11}] = \texttt{IntraOrtho}(\boldsymbol{X}_1)$
for $k = 1, \ldots, p-1$ **do**
  $\begin{bmatrix} \mathcal{R}_{1:k,k+1} \\ \Omega \end{bmatrix} = [\boldsymbol{\mathcal{Q}}_{1:k} \ \boldsymbol{X}_{k+1}]^T \boldsymbol{X}_{k+1}$
  $R_{k+1,k+1} = \texttt{chol}(\Omega - \mathcal{R}_{1:k,k+1}^T \mathcal{R}_{1:k,k+1})$
  $\boldsymbol{W} = \boldsymbol{X}_{k+1} - \boldsymbol{\mathcal{Q}}_{1:k} \mathcal{R}_{1:k,k+1}$
  $\boldsymbol{Q}_{k+1} = \boldsymbol{W} R_{k+1,k+1}^{-1}$
end for
return $\boldsymbol{\mathcal{Q}} = [\boldsymbol{Q}_1, \ldots, \boldsymbol{Q}_p]$, $\mathcal{R} = (R_{jk})$

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

Allocate memory for $\boldsymbol{\mathcal{Q}}$ and $\mathcal{R}$
$[\boldsymbol{Q}_1, R_{11}] = \texttt{IntraOrtho}(\boldsymbol{X}_1)$
for $k = 1, \ldots, p-1$ **do**
  $\mathcal{R}_{1:k,k+1} = \boldsymbol{\mathcal{Q}}_{1:k}^T \boldsymbol{X}_{k+1}$
  $\left[\sim, \begin{bmatrix} S_{k+1} & \\ & T_{k+1} \end{bmatrix}\right] = \texttt{IntraOrtho}\left(\begin{bmatrix} \boldsymbol{X}_{k+1} & \mathcal{R}_{1:k,k+1} \end{bmatrix}\right)$
  $R_{k+1,k+1} = \texttt{chol}(S_{k+1}^T S_{k+1} - T_{k+1}^T T_{k+1})$
  $\boldsymbol{W} = \boldsymbol{X}_{k+1} - \boldsymbol{\mathcal{Q}}_{1:k} \mathcal{R}_{1:k,k+1}$
  $\boldsymbol{Q}_{k+1} = \boldsymbol{W} R_{k+1,k+1}^{-1}$
end for
return $\boldsymbol{\mathcal{Q}} = [\boldsymbol{Q}_1, \ldots, \boldsymbol{Q}_p]$, $\mathcal{R} = (R_{jk})$

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

Allocate memory for $\boldsymbol{\mathcal{Q}}$ and $\mathcal{R}$
$[\boldsymbol{Q}_1, R_{11}] = \texttt{IntraOrtho}(\boldsymbol{X}_1)$
for $k = 1, \ldots, p-1$ **do**
  $\mathcal{R}_{1:k,k+1}^{(1)} = \boldsymbol{\mathcal{Q}}_{1:k}^T \boldsymbol{X}_{k+1}$ % first BCGS step
  $\boldsymbol{W} = \boldsymbol{X}_{k+1} - \boldsymbol{\mathcal{Q}}_{1:k} \mathcal{R}_{1:k,k+1}^{(1)}$
  $[\widehat{\boldsymbol{Q}}, R_{k+1,k+1}^{(1)}] = \texttt{IntraOrtho}(\boldsymbol{W})$
  $\mathcal{R}_{1:k,k+1}^{(2)} = \boldsymbol{\mathcal{Q}}_{1:k}^T \widehat{\boldsymbol{Q}}$ % second BCGS step
  $\boldsymbol{W} = \widehat{\boldsymbol{Q}} - \boldsymbol{\mathcal{Q}}_{1:k} \mathcal{R}_{1:k,k+1}^{(2)}$
  $[\boldsymbol{Q}_{k+1}, R_{k+1,k+1}^{(2)}] = \texttt{IntraOrtho}(\boldsymbol{W})$
  $\mathcal{R}_{1:k,k+1} = \mathcal{R}_{1:k,k+1}^{(1)} + \mathcal{R}_{1:k,k+1}^{(2)} R_{k+1,k+1}^{(1)}$ % combine both steps
  $R_{k+1,k+1} = R_{k+1,k+1}^{(2)} R_{k+1,k+1}^{(1)}$
end for
return $\boldsymbol{\mathcal{Q}} = [\boldsymbol{Q}_1, \ldots, \boldsymbol{Q}_p]$, $\mathcal{R} = (R_{jk})$

Algorithm A.5 $[\boldsymbol{\mathcal{Q}}, \mathcal{R}] = \texttt{BCGSI+LS}(\boldsymbol{\mathcal{X}})$

Allocate memory for $\boldsymbol{\mathcal{Q}}$ and $\mathcal{R}$
$\boldsymbol{U} = \boldsymbol{X}_1$
for $k = 2, \ldots p$ **do**
  if $k = 2$ **then**
    $[R_{k-1,k-1}^T R_{k-1,k-1} \ P] = \boldsymbol{U}^T [\boldsymbol{U} \ \boldsymbol{X}_k]$
  else if $k > 2$ **then**
    $\begin{bmatrix} \boldsymbol{W} & \boldsymbol{Z} \\ \Omega & Z \end{bmatrix} = [\boldsymbol{\mathcal{Q}}_{1:k-2} \ \boldsymbol{U}]^T [\boldsymbol{U} \ \boldsymbol{X}_k]$
    $[R_{k-1,k-1}^T R_{k-1,k-1} \ P] = [\Omega \ Z] - \boldsymbol{W}^T [\boldsymbol{W} \ \boldsymbol{Z}]$
  end if
  $R_{k-1,k} = R_{k-1,k-1}^{-T} P$
  if $k = 2$ **then**
    $\boldsymbol{Q}_{k-1} = \boldsymbol{U} R_{k-1,k-1}^{-1}$
  else if $k > 2$ **then**
    $\mathcal{R}_{1:k-2,k-1} = \mathcal{R}_{1:k-2,k-1} + \boldsymbol{W}$
    $\mathcal{R}_{1:k-2,k} = \boldsymbol{Z}$
    $\boldsymbol{Q}_{k-1} = (\boldsymbol{U} - \boldsymbol{\mathcal{Q}}_{1:k-2} \boldsymbol{W}) R_{k-1,k-1}^{-1}$
  end if
  $\boldsymbol{U} = \boldsymbol{X}_k - \boldsymbol{\mathcal{Q}}_{1:k-1} \mathcal{R}_{1:k-1,k}$
end for
$\begin{bmatrix} \boldsymbol{W} \\ \Omega \end{bmatrix} = [\boldsymbol{\mathcal{Q}}_{1:s-1} \ \boldsymbol{U}]^T \boldsymbol{U}$
$R_{s,s}^T R_{s,s} = \Omega - \boldsymbol{W}^T \boldsymbol{W}$
$\mathcal{R}_{1:s-1,s} = \mathcal{R}_{1:s-1,s} + \boldsymbol{W}$
$\boldsymbol{Q}_s = (\boldsymbol{U} - \boldsymbol{\mathcal{Q}}_{1:s-1} \boldsymbol{W}) R_{s,s}^{-1}$
return $\boldsymbol{\mathcal{Q}} = [\boldsymbol{Q}_1, \ldots, \boldsymbol{Q}_s]$, $\mathcal{R} = (R_{jk})$

Algorithm A.6 $[\boldsymbol{\mathcal{Q}}, \mathcal{R}] = \texttt{BMGS}(\boldsymbol{\mathcal{X}})$

Allocate memory for $\boldsymbol{\mathcal{Q}}$ and $\mathcal{R}$
$[\boldsymbol{Q}_1, R_{11}] = \texttt{IntraOrtho}(\boldsymbol{X}_1)$
for $k = 1, \ldots, p-1$ **do**
  $\boldsymbol{W} = \boldsymbol{X}_{k+1}$
  for $j = 1, \ldots, k$ **do**
    $R_{j,k+1} = \boldsymbol{Q}_j^T \boldsymbol{W}$
    $\boldsymbol{W} = \boldsymbol{W} - \boldsymbol{Q}_j R_{j,k+1}$
  end for
  $[\boldsymbol{Q}_{k+1}, R_{k+1,k+1}] = \texttt{IntraOrtho}(\boldsymbol{W})$
end for
return $\boldsymbol{\mathcal{Q}} = [\boldsymbol{Q}_1, \ldots, \boldsymbol{Q}_p]$, $\mathcal{R} = (R_{jk})$

Algorithm A.7 $[\boldsymbol{\mathcal{Q}}, \mathcal{R}, \mathcal{T}] = \texttt{BMGS-SVL}(\boldsymbol{\mathcal{X}})$

Allocate memory for $\boldsymbol{\mathcal{Q}}$, $\mathcal{R}$, and $\mathcal{T}$
$[\boldsymbol{Q}_1, R_{11}, T_{11}] = \texttt{IntraOrtho}(\boldsymbol{X}_1)$
for $k = 1, \ldots, p-1$ **do**
  $\mathcal{R}_{1:k,k+1} = \mathcal{T}_{1:k,1:k}^T(\boldsymbol{\mathcal{Q}}_{1:k}^T \boldsymbol{X}_{k+1})$
  $\boldsymbol{W} = \boldsymbol{X}_{k+1} - \boldsymbol{\mathcal{Q}}_{1:k}\mathcal{R}_{1:k,k+1}$
  $[\boldsymbol{Q}_{k+1}, R_{k+1,k+1}, T_{k+1,k+1}] = \texttt{IntraOrtho}(\boldsymbol{W})$
  $\mathcal{T}_{1:k,k+1} = -\mathcal{T}_{1:k,1:k}(\boldsymbol{\mathcal{Q}}_{1:k}^T \boldsymbol{Q}_{k+1}) T_{k+1,k+1}$
end for
return $\boldsymbol{\mathcal{Q}} = [\boldsymbol{Q}_1, \ldots, \boldsymbol{Q}_p]$, $\mathcal{R} = (R_{jk})$, $\mathcal{T} = (T_{jk})$

Algorithm A.8 $[\boldsymbol{\mathcal{Q}}, \mathcal{R}, \mathcal{T}] = \texttt{BMGS-LTS}(\boldsymbol{X})$

Allocate memory for $\boldsymbol{\mathcal{Q}}$, $\mathcal{R}$, and $\mathcal{T}$
$[\boldsymbol{Q}_1, R_{11}, T_{11}] = \texttt{IntraOrtho}(\boldsymbol{X}_1)$
for $k = 1, \ldots, p-1$ **do**
  $\mathcal{R}_{1:k,k+1} = \mathcal{T}_{1:k,1:k}^{-T}(\boldsymbol{\mathcal{Q}}_{1:k}^T \boldsymbol{X}_{k+1})$
  $\boldsymbol{W} = \boldsymbol{X}_{k+1} - \boldsymbol{\mathcal{Q}}_{1:k}\mathcal{R}_{1:k,k+1}$
  $[\boldsymbol{Q}_{k+1}, R_{k+1,k+1}, T_{k+1,k+1}] = \texttt{IntraOrtho}(\boldsymbol{W})$
  $\mathcal{T}_{1:k,k+1} = (\boldsymbol{\mathcal{Q}}_{1:k}^T \boldsymbol{Q}_{k+1}) T_{k+1,k+1}$
end for
return $\boldsymbol{\mathcal{Q}} = [\boldsymbol{Q}_1, \ldots, \boldsymbol{Q}_p]$, $\mathcal{R} = (R_{jk})$, $\mathcal{T} = (T_{jk})$

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

Allocate memory for $\boldsymbol{\mathcal{Q}}$ and $\mathcal{R}$
$\mathcal{T} = I_n$
$\boldsymbol{U} = \boldsymbol{X}_1$
for $k = 1, \ldots, p-1$ **do**
  $\boldsymbol{W} = \boldsymbol{X}_{k+1}$
  if $k = 1$ **then**
    $\begin{bmatrix} R_{k,k}^T R_{k,k} & P \end{bmatrix} = \boldsymbol{U}^T[\boldsymbol{U}\ \boldsymbol{W}]$ % recover $R_{k,k}$ with `chol`
  else if $k > 1$ **then**
    $\begin{bmatrix} \boldsymbol{T} & \boldsymbol{R} \\ R_{k,k}^T R_{k,k} & P \end{bmatrix} = [\boldsymbol{\mathcal{Q}}_{1:k-1}\ \boldsymbol{U}]^T[\boldsymbol{U}\ \boldsymbol{W}]$ % recover $R_{k,k}$ with `chol`
    $\mathcal{T}_{1:k-1,k} = -\mathcal{T}_{1:k-1,1:k-1}(\boldsymbol{T} R_{k,k}^{-1})$
  end if
  $\mathcal{R}_{1:k,k+1} = \mathcal{T}_{1:k,1:k}^T \begin{bmatrix} \boldsymbol{R} \\ R_{k,k}^{-T} P \end{bmatrix}$
  $\boldsymbol{Q}_k = \boldsymbol{U} R_{k,k}^{-1}$
  $\boldsymbol{U} = \boldsymbol{W} - \boldsymbol{\mathcal{Q}}_{:,1:k}\mathcal{R}_{1:k,k+1}$
end for
$[\boldsymbol{Q}_p, R_{p,p}] = \texttt{IntraOrtho}(\boldsymbol{U})$
return $\boldsymbol{\mathcal{Q}} = [\boldsymbol{Q}_1, \ldots, \boldsymbol{Q}_p]$, $\mathcal{R} = (R_{jk})$, $\mathcal{T} = (T_{jk})$

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

Allocate memory for $\boldsymbol{\mathcal{Q}}$ and $\mathcal{R}$
$\mathcal{T} = I_n$
$\boldsymbol{U} = \boldsymbol{X}_1$
for $k = 1, \ldots, p-1$ **do**
  $\boldsymbol{W} = \boldsymbol{X}_{k+1}$
  if $k = 1$ **then**
    $\begin{bmatrix} R_{k,k}^T R_{k,k} & P \end{bmatrix} = \boldsymbol{U}^T [\boldsymbol{U}\ \boldsymbol{W}]$ % recover $R_{k,k}$ with `chol`
  else if $k > 1$ **then**
    $\begin{bmatrix} \boldsymbol{T} & \boldsymbol{R} \\ R_{k,k}^T R_{k,k} & P \end{bmatrix} = [\boldsymbol{\mathcal{Q}}_{1:k-1}\ \boldsymbol{U}]^T [\boldsymbol{U}\ \boldsymbol{W}]$ % recover $R_{k,k}$ with `chol`
    $\mathcal{T}_{1:k-1,k} = \boldsymbol{T} R_{k,k}^{-1}$
  end if
  $\mathcal{R}_{1:k,k+1} = \mathcal{T}_{1:k,1:k}^{-T} \begin{bmatrix} \boldsymbol{R} \\ R_{k,k}^{-T} P \end{bmatrix}$
  $\boldsymbol{Q}_k = \boldsymbol{U} R_{k,k}^{-1}$
  $\boldsymbol{U} = \boldsymbol{W} - \boldsymbol{\mathcal{Q}}_{:,1:k} \mathcal{R}_{1:k,k+1}$
end for
$[\boldsymbol{Q}_p, R_{p,p}] = \texttt{IntraOrtho}(\boldsymbol{U})$
return $\boldsymbol{\mathcal{Q}} = [\boldsymbol{Q}_1, \ldots, \boldsymbol{Q}_p]$, $\mathcal{R} = (R_{jk})$, $\mathcal{T} = (T_{jk})$

Algorithm A.11 $[\boldsymbol{\mathcal{Q}}, \mathcal{R}] = \texttt{BCGSS+rpl}(\boldsymbol{\mathcal{X}}, \delta)$

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

Appendix B. Pseudocode for muscles. For the muscles, we reference the column vectors of $\boldsymbol{X} \in \mathbb{R}^{m \times s}$ as

$$\boldsymbol{X} = [\boldsymbol{x}_1 \,|\, \boldsymbol{x}_2 \,|\, \cdots \,|\, \boldsymbol{x}_s],$$

and likewise for $\boldsymbol{Q}$. Entries of R and T are expressed as their lowercase counterparts, r_{jk} and t_{jk}, respectively.

We simplify indexing in a similar manner as in the previous section:

$$\boldsymbol{Q}_{1:j} = [\boldsymbol{x}_1 \,|\, \cdots \,|\, \boldsymbol{x}_s] \in \mathbb{R}^{m \times j},$$

and

$$R_{1:j,k} = \begin{bmatrix} r_{1,k} \\ r_{2,k} \\ \vdots \\ r_{j,k} \end{bmatrix} \in \mathbb{R}^{j \times 1},$$

where $1 : j := \{1, \ldots, j\}$.

Note that by setting $s = 1$ for all the algorithms in Appendix A (except `BCGSS+rpl`), we recover the column-wise version. Therefore, such algorithms as `CGS`, `MGS`, etc. are

omitted. The low-sync variants from Świrydowicz et al. [55] are included in full for clarity. For all Cholesky-based algorithms, we let $R = \texttt{chol}(A)$ denote an algorithm that takes a numerically positive-definite matrix $A \in \mathbb{R}^{s\times s}$ and returns an upper triangular matrix $R \in \mathbb{R}^{s\times s}$.

Algorithm B.1 $[\boldsymbol{Q}, \boldsymbol{R}] = \texttt{CGSS+rpl}(\boldsymbol{X}, \delta)$

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

```
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
```

Algorithm B.2 $[\boldsymbol{Q}, R] =$ `CGSI+LS`$(\boldsymbol{X})$

Allocate memory for $\boldsymbol{Q}$ and R
$\boldsymbol{u} = \boldsymbol{x}_1$
for $k = 2, \ldots s$ **do**
  if $k = 2$ **then**
    $[r_{k-1,k-1}^2 \ \rho] = \boldsymbol{u}^T[\boldsymbol{u} \ \boldsymbol{x}_k]$
  else if $k > 2$ **then**
    $\begin{bmatrix} \boldsymbol{w} & \boldsymbol{z} \\ \omega & \zeta \end{bmatrix} = [\boldsymbol{Q}_{1:k-2} \ \boldsymbol{u}]^T[\boldsymbol{u} \ \boldsymbol{x}_k]$
    $[r_{k-1,k-1}^2 \ \rho] = [\omega \ \zeta] - \boldsymbol{w}^T[\boldsymbol{w} \ \boldsymbol{z}]$
  end if
  $r_{k-1,k} = \rho / r_{k-1,k-1}$
  if $k = 2$ **then**
    $\boldsymbol{q}_{k-1} = \boldsymbol{u} / r_{k-1,k-1}$
  else if $k > 2$ **then**
    $R_{1:k-2,k-1} = R_{1:k-2,k-1} + \boldsymbol{w}$
    $R_{1:k-2,k} = \boldsymbol{z}$
    $\boldsymbol{q}_{k-1} = (\boldsymbol{u} - \boldsymbol{Q}_{1:k-2}\boldsymbol{w}) / r_{k-1,k-1}$
  end if
  $\boldsymbol{u} = \boldsymbol{x}_k - \boldsymbol{Q}_{1:k-1} R_{1:k-1,k}$
end for
$\begin{bmatrix} \boldsymbol{w} \\ \omega \end{bmatrix} = [\boldsymbol{Q}_{1:s-1} \ \boldsymbol{u}]^T \boldsymbol{u}$
$r_{s,s}^2 = \omega - \boldsymbol{w}^T \boldsymbol{w}$
$R_{1:s-1,s} = R_{1:s-1,s} + \boldsymbol{w}$
$\boldsymbol{q}_s = (\boldsymbol{u} - \boldsymbol{Q}_{1:s-1}\boldsymbol{w}) / r_{s,s}$
return $\boldsymbol{Q} = [\boldsymbol{q}_1, \ldots, \boldsymbol{q}_s]$, $R = (r_{jk})$

Algorithm B.3 $[\boldsymbol{Q}, R, T] = \texttt{MGS-SVL}(\boldsymbol{X})$

Allocate memory for $\boldsymbol{Q}$ and R
$T = I_s$
$r_{11} = \|\boldsymbol{x}_1\|$; $\boldsymbol{q}_1 = \boldsymbol{x}_1 / r_{11}$
for $k = 1, \ldots, s-1$ **do**
  $\boldsymbol{w} = \boldsymbol{x}_{k+1}$
  $R_{1:k,k+1} = T^T_{1:k,1:k}(\boldsymbol{Q}^T_{1:k}\boldsymbol{w})$
  $\boldsymbol{w} = \boldsymbol{w} - \boldsymbol{Q}_{1:k} R_{1:k,k+1}$
  $r_{k+1,k+1} = \|\boldsymbol{w}\|$
  $\boldsymbol{q}_{k+1} = \boldsymbol{w} / r_{k+1,k+1}$
  $T_{1:k,k+1} = -T_{1:k,1:k}(\boldsymbol{Q}^T_{1:k}\boldsymbol{q}_{k+1})$
end for
return $\boldsymbol{Q} = [\boldsymbol{q}_1, \ldots, \boldsymbol{q}_s]$, $R = (r_{jk})$, $T = (t_{jk})$

Algorithm B.4 $[\boldsymbol{Q}, R, T] = \texttt{MGS-LTS}(\boldsymbol{X})$

Allocate memory for $\boldsymbol{Q}$ and R
$T = I_s$
$r_{11} = \|\boldsymbol{x}_1\|$; $\boldsymbol{q}_1 = \boldsymbol{x}_1 / r_{11}$
for $k = 1, \ldots, s-1$ **do**
  $\boldsymbol{w} = \boldsymbol{x}_{k+1}$
  $R_{1:k,k+1} = T^{-T}_{1:k,1:k}(\boldsymbol{Q}^T_{1:k}\boldsymbol{w})$
  $\boldsymbol{w} = \boldsymbol{w} - \boldsymbol{Q}_{1:k} R_{1:k,k+1}$
  $r_{k+1,k+1} = \|\boldsymbol{w}\|$
  $\boldsymbol{q}_{k+1} = \boldsymbol{w} / r_{k+1,k+1}$
  $T_{1:k,k+1} = \boldsymbol{Q}^T_{1:k}\boldsymbol{q}_{k+1}$
end for
return $\boldsymbol{Q} = [\boldsymbol{q}_1, \ldots, \boldsymbol{q}_s]$, $R = (r_{jk})$, $T = (t_{jk})$

Algorithm B.5 $[\boldsymbol{Q}, R, T] = \texttt{MGS-CWY}(\boldsymbol{X})$

Allocate memory for $\boldsymbol{Q}$ and R
$T = I_s$
$\boldsymbol{u} = \boldsymbol{x}_1$
for $k = 1, \ldots, s-1$ **do**
  $\boldsymbol{w} = \boldsymbol{x}_{k+1}$
  if $k = 1$ **then**
    $\begin{bmatrix} r_{k,k}^2 & \rho \end{bmatrix} = \boldsymbol{u}^T [\boldsymbol{u}\ \boldsymbol{w}]$
  else if $k > 1$ **then**
    $\begin{bmatrix} \boldsymbol{t} & \boldsymbol{r} \\ r_{k,k}^2 & \rho \end{bmatrix} = [\boldsymbol{Q}_{1:k-1}\ \boldsymbol{u}]^T [\boldsymbol{u}\ \boldsymbol{w}]$
    $T_{1:k-1,k} = -T_{1:k-1,1:k-1}(\boldsymbol{t}/r_{k,k})$
  end if
  $R_{1:k,k+1} = T_{1:k,1:k}^T \begin{bmatrix} \boldsymbol{r} \\ \rho/r_{k,k} \end{bmatrix}$
  $\boldsymbol{q}_k = \boldsymbol{u}/r_{k,k}$
  $\boldsymbol{u} = \boldsymbol{w} - \boldsymbol{Q}_{:,1:k} R_{1:k,k+1}$
end for
$r_{s,s} = \|\boldsymbol{u}\|$
$\boldsymbol{q}_s = \boldsymbol{u}/r_{s,s}$
return $\boldsymbol{Q} = [\boldsymbol{q}_1, \ldots, \boldsymbol{q}_s]$, $R = (r_{jk})$, $T = (t_{jk})$

Algorithm B.6 $[\boldsymbol{Q}, R, T] = \texttt{MGS-ICWY}(\boldsymbol{X})$

Allocate memory for $\boldsymbol{Q}$ and R
$T = I_s$
$\boldsymbol{u} = \boldsymbol{x}_1$
for $k = 1, \ldots, s-1$ **do**
  $\boldsymbol{w} = \boldsymbol{x}_{k+1}$
  if $k = 1$ **then**
    $\begin{bmatrix} r_{k,k}^2 & \rho \end{bmatrix} = \boldsymbol{u}^T [\boldsymbol{u}\ \boldsymbol{w}]$
  else if $k > 1$ **then**
    $\begin{bmatrix} \boldsymbol{t} & \boldsymbol{r} \\ r_{k,k}^2 & \rho \end{bmatrix} = [\boldsymbol{Q}_{1:k-1}\ \boldsymbol{u}]^T [\boldsymbol{u}\ \boldsymbol{w}]$
    $T_{1:k-1,k} = \boldsymbol{t}/r_{k,k}$
  end if
  $R_{1:k,k+1} = T_{1:k,1:k}^{-T} \begin{bmatrix} \boldsymbol{r} \\ \rho/r_{k,k} \end{bmatrix}$
  $\boldsymbol{q}_s = \boldsymbol{u}/r_{s,s}$
  $\boldsymbol{u} = \boldsymbol{w} - \boldsymbol{Q}_{:,1:k} R_{1:k,k+1}$
end for
$r_{s,s} = \|\boldsymbol{u}\|$
$\boldsymbol{q}_s = \boldsymbol{u}/r_{s,s}$
return $\boldsymbol{Q} = [\boldsymbol{q}_1, \ldots, \boldsymbol{q}_s]$, $R = (r_{jk})$, $T = (t_{jk})$

Algorithm B.7 $[\boldsymbol{Q}, R] = \texttt{CholQR}(\boldsymbol{X})$

$X = \boldsymbol{X}^T \boldsymbol{X}$
if X is positive definite **then**
  $R = \texttt{chol}(X)$
  $\boldsymbol{Q} = \boldsymbol{X} R^{-1}$
else
  return $\boldsymbol{Q} = \texttt{NaN}$, $R = \texttt{NaN}$
end if
return $\boldsymbol{Q}$, R

Algorithm B.8 $[\boldsymbol{Q}, R] = \texttt{CholQR+}(\boldsymbol{X})$

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

Algorithm B.9 $[\boldsymbol{Q}, R] = \texttt{ShCholQR++}(\boldsymbol{X})$

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

Appendix C. Glued matrices and command-line calls. Glued matrices are generated by the following MATLAB code, modified from [50]:

```
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.1.

TABLE C.1
Command-line calls for κ-plots

Figure	Code
4.1	`GluedKappaPlot([1000 200], 1:8, {'CGS', 'CGS_P'})`
4.2	`GluedBlockKappaPlot([1000 50 4], 1:8, ...` `{'BCGS', 'BCGS_PIP', 'BCGS_PIO'}, 'HouseQR')`

4.3	`BlockKappaPlot([100 20 2], -(1:16), ...`
	`{'BCGS', 'BMGS', 'BCGS_IRO'},...`
	`{'CGS', 'MGS', 'HouseQR'})`

4.4	`BlockKappaPlot([100 20 2], -(1:16),...`
	`{'BCGS', 'BCGS_IRO', 'BCGS_IRO_LS'},...`
	`{'CGS', 'CGS_IRO', 'CGS_IRO_LS'})`

4.5	`MonomialBlockKappaPlot([1000 120 2], 2:2:12,...`
	`{'BCGS', 'BCGS_IRO', 'BCGS_IRO_LS'},...`
	`{'CGS', 'CGS_IRO', 'CGS_IRO_LS'})`

4.6	`BlockKappaPlot([100 20 2], -(1:16), 'BMGS',...`
	`{'CGS', 'MGS', 'CGS_RO', 'MGS_RO',...`
	`'CholQR', 'Cholqr_RO', 'HouseQR'})`

4.7	`KappaPlot([1000 20], [],...` `{'MGS', 'MGS_SVL', 'MGS_LTS', 'MGS_CWY', 'MGS_ICWY'})`

4.8	`BlockKappaPlot([100 20 2], -(1:16),...`
	`{'BMGS', 'BMGS_SVL', 'BMGS_CWY'},...`
	`{'MGS', 'MGS_SVL', 'MGS_LTS', ...`
	`'MGS_CWY', 'MGS_ICWY', 'HouseQR'})`
	`MonomialBlockKappaPlot([1000 120 2], 2:2:12,...`
	`{'BMGS', 'BMGS_SVL', 'BMGS_CWY'},...`
	`{'MGS', 'MGS_SVL', 'MGS_LTS', ...`
	`'MGS_CWY', 'MGS_ICWY', 'HouseQR'})`

REFERENCES

[1] N. N. ABDELMALEK, *Round off error analysis for Gram-Schmidt method and solution of linear least squares problems*, BIT, 11 (1971), pp. 345–368, https://doi.org/10.1007/BF01939404.

[2] A. H. BAKER, J. M. DENNIS, AND E. R. JESSUP, *On improving linear solver performance: a block variant of GMRES*, SIAM J. Sci. Comput., 27 (2006), pp. 1608–1626, https://doi.org/10.1137/040608088.

[3] G. BALLARD, E. C. CARSON, J. W. DEMMEL, M. HOEMMEN, N. KNIGHT, AND O. SCHWARTZ, *Communication lower bounds and optimal algorithms for numerical linear algebra*, Acta Numer., 23 (2014), pp. 1–155, https://doi.org/10.1017/S0962492914000038.

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

[5] J. L. BARLOW AND A. SMOKTUNOWICZ, *Reorthogonalized block classical Gram-Schmidt*, Numer. Math., 123 (2013), pp. 395–423, https://doi.org/10.1007/s00211-012-0496-2.

[6] A. BIENZ, W. GROPP, AND L. OLSON, *Node-aware improvements to allreduce*, in Proceedings of the 2019 IEEE/ACM Workshop on Exascale MPI (ExaMPI), Association for Computing Machinery, 2019.

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

[8] Å. BJÖRCK AND C. C. PAIGE, *Loss and Recapture of Orthogonality in the Modified Gram–Schmidt Algorithm*, SIAM J. Matrix Anal. Appl., 13 (1992), pp. 176–190, https://doi.org/10.1137/0613015.

[9] E. C. CARSON, *Communication-Avoiding Krylov Subspace Methods in Theory and Practice*, PhD thesis, Department of Computer Science, University of California, Berkeley, 2015.

[10] E. C. Carson, *An Adaptive s-step Conjugate Gradient Algorithm with Dynamic Basis Updating*, Appl. Math., To appear (2020), http://arxiv.org/abs/1908.04081, https://arxiv.org/abs/1908.04081.

[11] J. W. Daniel, W. B. Gragg, L. Kaufman, and G. W. Stewart, *Reorthogonalization and Stable Algorithms for Updating the Gram-Schmidt QR Factorization*, Math. Comput., 30 (1976), p. 772, https://doi.org/10.2307/2005398.

[12] J. W. Demmel, L. Grigori, M. Hoemmen, and J. Langou, *Communication-optimal parallel and sequential QR and LU factorizations*, tech. report, University of California, Berkeley, 2008, https://arxiv.org/abs/0808.2664.

[13] J. W. Demmel, L. Grigori, M. Hoemmen, and J. Langou, *Communication-optimal parallel and sequential QR and LU factorizations*, SIAM J. Sci. Comput., 34 (2012), pp. A206–A239, https://doi.org/10.1137/080731992, https://epubs.siam.org/doi/abs/10.1137/080731992.

[14] M. Eiermann, O. G. Ernst, and S. Güttel, *Deflated restarting for matrix functions*, SIAM J. Matrix Anal. Appl., 32 (2011), pp. 621–641, https://doi.org/10.1137/090774665, http://link.aip.org/link/?SJMAEL/32/621/1.

[15] A. Frommer, S. Güttel, and M. Schweitzer, *Convergence of restarted Krylov subspace methods for Stieltjes functions of matrices*, SIAM J. Matrix Anal. Appl., 35 (2014), pp. 1602–1624, https://doi.org/10.1137/140973463.

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

[17] A. Frommer, K. Lund, and D. B. Szyld, *Block Krylov subspace methods for functions of matrices II: Modified block FOM*, SIAM J. Matrix Anal. Appl., To appear (2020).

[18] T. Fukaya, R. Kannan, Y. Nakatsukasa, Y. Yamamoto, and Y. Yanagisawa, *Shifted Cholesky QR for computing the QR factorization of ill-conditioned matrices*, SIAM J. Sci. Comput., 42 (2020), pp. A477–A503, https://doi.org/10.1137/18M1218212.

[19] W. Gander, *Algorithms for the QR-decomposition*, Tech. Report 80-02, April 1980, Seminar für Angewandte Mathematik, Eidgenössische Technische Hochschule, 1980, http://people.inf.ethz.ch/gander/papers/qrneu.pdf.

[20] L. Giraud and J. Langou, *When modified Gram-Schmidt generates a well-conditioned set of vectors*, IMA J. Numer. Anal., 22 (2002), pp. 521–528, https://doi.org/10.1093/imanum/22.4.521.

[21] L. Giraud, J. Langou, M. Rozložník, and J. Van Den Eshof, *Rounding error analysis of the classical Gram-Schmidt orthogonalization process*, Numer. Math., 101 (2005), pp. 87–100, https://doi.org/10.1007/s00211-005-0615-4.

[22] G. H. Golub and R. Underwood, *The block Lanczos method for computing eigenvalues*, in Math. Softw. III Proc. a Symp. Conduct. by Math. Res. Center, Univ. Wisconsin-Madison, New York, 1977, Academic Press, pp. 361–377.

[23] G. H. Golub and C. F. van Loan, *Matrix Computations*, Johns Hopkins University Press, Baltimore, 4th ed., 2013, https://doi.org/10.1063/1.3060478.

[24] L. Grigori, S. Moufawad, and F. Nataf, *Enlarged Krylov subspace conjugate gradient methods for reducing communicaiton*, SIAM J. Matrix Anal. Appl., 37 (2016), pp. 744–773, https://doi.org/10.1137/140989492.

[25] M. H. Gutknecht, *Block Krylov space methods for linear systems with multiple right-hand sides: An introduction*, in Mod. Math. Model. Methods Algorithms Real World Syst., A. H. Siddiqi, I. S. Duff, and O. Christensen, eds., New Delhi, 2007, Anamaya, pp. 420–447.

[26] M. H. Gutknecht and T. Schmelzer, *Updating the QR decomposition of block tridiagonal and block Hessenberg matrices*, Appl. Numer. Math., 58 (2008), pp. 871–883, https://doi.org/10.1016/j.apnum.2007.04.010.

[27] S. Güttel, *Rational Krylov approximation of matrix functions: Numerical methods and optimal pole selection*, GAMM Mitteilungen, 36 (2013), pp. 8–31, https://doi.org/10.1002/gamm.201310002.

[28] N. J. Higham, *Accuracy and Stability of Numerical Algorithms*, Society for Industrial and Applied Mathematics, Philadelphia, 2nd ed., 2002.

[29] M. Hoemmen, *Communication-avoiding Krylov subspace methods*, PhD thesis, Department of Computer Science, University of California at Berkeley, 2010.

[30] W. Hoffmann, *Iterative algorithms for Gram-Schmidt orthogonalization*, Computing, 41 (1989), pp. 335–348, https://doi.org/10.1007/BF02241222.

[31] W. Jalby and B. Philippe, *Stability analysis and improvement of the block Gram-Schmidt algorithm*, SIAM J. Sci. Comput., 12 (1991), pp. 1058–1073, https://doi.org/10.1137/0912056, https://epubs.siam.org/doi/abs/10.1137/0912056.

[32] P. Jolivet, J. E. Roman, and S. Zampini, *KSPHPDDM and PCHPDDM: extending PETSc*

with advanced Krylov methods and robust multilevel overlapping Schwarz preconditioners, 2020. in review.

[33] A. Kiełbasiński, *Analiza numeryczna algorytmu ortogonalizacji Grama-Schmidta*, Ser. III Mat. Stosow. II, (1974), pp. 15–35.

[34] S. K. Kim and A. T. Chronopoulos, *An efficient parallel algorithm for extreme eigenvalues of sparse nonsymmetric matrices*, International Journal of Supercomputer Applications, 6 (1992), pp. 98–111.

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

[36] S. J. Leon, Å. Björck, and W. Gander, *Gram-Schmidt orthogonalization: 100 years and more*, Numer. Linear Algebr. with Appl., 20 (2013), pp. 492–532, https://doi.org/10.1002/nla, https://arxiv.org/abs/arXiv:1112.5346v3.

[37] R. B. Morgan, *Restarted block-GMRES with deflation of eigenvalues*, Appl. Numer. Math., 54 (2005), pp. 222–236, https://doi.org/10.1016/j.apnum.2004.09.028.

[38] D. Mori, Y. Yamamoto, and S. L. Zhang, *Backward error analysis of the AllReduce algorithm for householder QR decomposition*, Jpn. J. Ind. Appl. Math., 29 (2012), pp. 111–130, https://doi.org/10.1007/s13160-011-0053-x.

[39] D. P. O'Leary, *The block conjugate gradient algorithm and related methods*, Linear Algebra Appl., 29 (1980), pp. 293–322, https://doi.org/10.1016/0024-3795(80)90247-5, https://doi.org/10.1016/0024-3795(80)90247-5.

[40] C. C. Paige, M. Rozložník, and Z. Strakoš, *Modified Gram-Schmidt (MGS), least squares, and backward stability of MGS-GMRES*, SIAM J. Matrix Anal. Appl., 28 (2006), pp. 264–284, https://doi.org/10.1137/050630416.

[41] M. L. Parks, E. De Sturler, G. Mackey, D. D. Johnson, and S. Maiti, *Recycling Krylov subspaces for sequences of linear systems*, SIAM J. Sci. Comput., 28 (2006), pp. 1651–1674, https://doi.org/10.1137/040607277.

[42] M. L. Parks, K. M. Soodhalter, and D. B. Szyld, *A block recycled GMRES method with investigations into aspects of solver performance*, tech. report, Department of Mathematics, Temple University, 2016, https://arxiv.org/abs/arXiv:1604.01713.

[43] B. N. Parlett, *The Symmetric Eigenvalue Problem*, SIAM, Philadelphia, 1998, https://doi.org/10.1137/1.9781611971163, https://epubs.siam.org/doi/abs/10.1137/1.9781611971163.

[44] A. Ruhe, *Numerical aspects of Gram-Schmidt orthogonalization of vectors*, Linear Algebra and its Applications, 52 (1983), pp. 591–601.

[45] Y. Saad, *Iterative methods for sparse linear systems*, SIAM, Philadelphia, 2nd ed., 2003.

[46] M. Sadkane, *Block-Arnoldi and Davidson methods for unsymmetric large eigenvalue problems*, Numer. Math., 64 (1993), pp. 195–211, https://doi.org/10.1007/BF01388687.

[47] T. Sakurai and S. H., *A projection method for generalized eigenvalue problems using numerical integration*, Journal of Computational and Applied Mathematics, 159 (2003), p. 119–128.

[48] R. Schreiber and C. Van Loan, *A Storage-Efficient WY Representation for Products of Householder Transformations*, SIAM J. Sci. Stat. Comput., 10 (1989), pp. 53–57, https://doi.org/10.1137/0910005.

[49] V. Simoncini and E. Gallopoulos, *Convergence properties of block GMRES and matrix polynomials*, Linear Algebra Appl., 247 (1996), pp. 97–119, https://doi.org/10.1016/0024-3795(95)00093-3.

[50] A. Smoktunowicz, J. L. Barlow, and J. Langou, *A note on the error analysis of classical Gram-Schmidt*, Numer. Math., 105 (2006), pp. 299–313, https://doi.org/10.1007/s00211-006-0042-1.

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

[52] G. W. Stewart, *A Krylov-Schur algorithm for large eigenproblems*, SIAM J. Matrix Anal. Appl., 23 (2001), pp. 601–614, https://doi.org/10.1137/S0895479800371529, http://epubs.siam.org/doi/abs/10.1137/S0895479800371529.

[53] G. W. Stewart, *Block Gram-Schmidt orthogonalization*, SIAM J. Sci. Comput., 31 (2008), pp. 761–775, https://doi.org/10.1137/070682563, https://epubs.siam.org/doi/abs/10.1137/070682563.

[54] X. Sun, *Aggregations of elementary transformations.*, Tech. Report Technical Report DUKE-TR-1996-03, Duke University, 1996.

[55] K. Świrydowicz, J. Langou, S. Ananthan, U. Yang, and S. Thomas, *Low synchronization Gram-Schmidt and Generalized Minimum Residual algorithms*, Numerical Linear Algebra with Applications, (2020), http://arxiv.org/abs/1809.05805, https://arxiv.org/abs/1809.

05805.

[56] D. Vanderstraeten, *An accurate parallel block Gram-Schmidt algorithm without reorthogonalization*, Numer. Linear Algebr. with Appl., 7 (2000), pp. 219–236, https://doi.org/10.1002/1099-1506(200005)7:4⟨219::AID-NLA196⟩3.0.CO;2-L.

[57] B. Vital, *Étude de quelques méthodes de résolution de problèmes linéaires de grande taille sur multiprocesseur*, PhD thesis, Université de Rennes, 1990.

[58] H. F. Walker, *Implementation of the GMRES method using Householder transformations*, SIAM Journal on Scientific and Statistical Computing, 9 (1988), pp. 152–163.

[59] Y. Yamamoto, Y. Nakatsukasa, Y. Yanagisawa, and T. Fukaya, *Roundoff error analysis of the Cholesky QR2 algorithm*, Electron. Trans. Numer. Anal., 44 (2015), pp. 306–326.

[60] I. Yamazaki and M. Hoemmen, *Communication-avoiding & pipelined Krylov solvers in Trilinos*. SIAM Conference on Computational Science and Engineering, Spokane, Washington, 2019. Presentation.

[61] I. Yamazaki, S. Thomas, M. Hoemmen, E. G. Boman, K. Świrydowicz, and 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. Contributed paper.

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

[63] I. Yamazaki, S. Tomov, J. Kurzak, J. Dongarra, and J. L. Barlow, *Mixed-precision block gram schmidt orthogonalization*, Proc. ScalA 2015 6th Work. Latest Adv. Scalable Algorithms Large-Scale Syst. - Held conjunction with SC 2015 Int. Conf. High Perform. Comput. Networking, Storage Anal., (2015), https://doi.org/10.1145/2832080.2832082.

[64] L. M. Yang, A. Fox, and G. Sanders, *Mixed-precision analysis of Householder QR algorithms*, (2019), https://arxiv.org/abs/1912.06217.