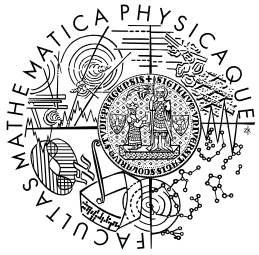

FACULTY
OF MATHEMATICS
AND PHYSICS
Charles University

DOCTORAL THESIS

Eda Oktay

Mixed-Precision Computations in Numerical Linear Algebra

Department of Numerical Mathematics

Supervisor of the doctoral thesis:	Assist. Prof. Erin Claire Carson, Ph.D.
Study programme:	Mathematics
Study branch:	Scientific and Technical Calculations

Prague 2024

I declare that I carried out this doctoral thesis independently, and only with the cited sources, literature and other professional sources. It has not been used to obtain another or the same degree.

I understand that my work relates to the rights and obligations under the Act No. 121/2000 Sb., the Copyright Act, as amended, in particular the fact that the Charles University has the right to conclude a license agreement on the use of this work as a school work pursuant to Section 60 subsection 1 of the Copyright Act.

In Prague March 8, 2024 signature of the author

With this thesis, I come to the end of my studies at Charles University, and I would like to thank those who helped me to finish this chapter of my life.

First, I would like to express my gratitude to my supervisor, Assist. Prof. Erin Carson, for her patience, motivation, and kindness during these four years. I am grateful that I was able to work with such an inspiration, and I will always appreciate her guidance throughout my studies. Not only did I learn so much from her about this field, but I was also motivated to become a researcher like her.

Secondly, I am thankful to work with Prof. Martin Stoll. Exploring a new field and research group at TU Chemnitz was a great pleasure. I will never forget how he encouraged and motivated me, even when I was incredibly anxious.

Last but not least, I would like to thank Dr. Kathryn Lund for her contributions to my thesis and for being there for me as a friend who supports and motivates me. It is special to have someone who knows what you are going through and helps you in the best way they can.

I would also like to thank Prof. Laura Grigori, Dr. Oleg Balabanov, and Prof. Stefan Güttel for contributing to my studies. Collaborating with them and learning so much about their fields was a great pleasure.

Finally, I am so happy to meet and spend time with the great people in the Department of Numerical Mathematics at Charles University and the Department of Mathematics at TU Chemnitz. I feel lucky to be in such a friendly environment. For that, I thank my fellow doctoral students and the professors who created this space. They made it much easier to work and motivate.

Everything aside, I dedicate this thesis to my loving parents, who never stopped supporting and encouraging me during my studies and this chapter of my life in the Czech Republic (and Germany). I will always be grateful for their faith in me.

Thank you for everything.

The work presented in the thesis was supported by the Charles University Research Program UNCE/SCI/023, PRIMUS project PRIMUS/19/SCI/11, ERC Starting Grant 101075632, by the Charles University Grant Agency project - 202722, and the Exascale Computing Project (17-SC-20-SC), a collaborative effort of the U.S. Department of Energy Office of Science and the National Nuclear Security Administration.

Title: Mixed-Precision Computations in Numerical Linear Algebra

Author: Eda Oktay

Department: Department of Numerical Mathematics

Supervisor: Assist. Prof. Erin Claire Carson, Ph.D., Department of Numerical Mathematics

Abstract: Modern commercial hardware natively supports arithmetic in multiple different precisions, including half (16-bit) precision in addition to the usual single (32-bit) and double (64-bit). Using lower precision has clear performance advantages; since we are moving, storing, and computing with fewer bits, we can save on both time and energy. However, low precision is also subject to greater roundoff errors and a smaller range of representable numbers, which can cause algorithms to become unstable. This has motivated research into mixed-precision algorithms for numerical linear algebra problems, where low and high precisions are used selectively within an algorithm to improve performance while maintaining the desired accuracy. The design of new mixed-precision algorithms and techniques for numerical linear algebra is the topic of my thesis research.

Keywords: mixed-precision, Krylov subspace methods, iterative methods, least-squares problems, block orthogonalization processes

Contents

List of Abbreviations and Notation

$\forall y$	for all vectors y
$\mathbb{R}$	set of real numbers
$\mathbb{R}^n$	set of real vectors of length n
$\mathbb{R}^{m\times n}$	set of real matrices of size $m \times n$
$(\cdot,\cdot)$	Euclidean inner product
$\|\cdot\|$	Euclidean norm
$\|\cdot\|_\infty$	Infinity norm
$\|\cdot\|_F$	Frobenius norm
$\text{span}\{\cdots\}$	subspace spanned by vectors
A^T	transpose of the matrix A
A^{-1}	inverse of the matrix A
$A^\dagger$	Moore-Penrose pseudoinverse of the matrix A
$\bar{A}$	computed matrix A in finite precision arithmetic
$\hat{A}$	approximately computed matrix A
I_n	identity matrix of size $n \times n$
$\sigma(A)$	singular values of A
$\sigma_{\min}(A)$	minimum singular value of A
$\lambda(A)$	eigenvalues of A
$\lambda_{\min}(A)$	minimum eigenvalue of A
$\kappa(\cdot)$	condition number
$\kappa_2(\cdot)$	condition number in Euclidean norm
$\kappa_\infty(\cdot)$	condition number in infinity norm
$\kappa_F(\cdot)$	condition number in Frobenius norm
$\text{rank}(\cdot)$	matrix rank
u	machine epsilon
$\mathcal{O}$	computation complexity of an algorithm (bigO)
$\mathcal{O}(u)$	elements of order u
(F)GMRES	(Flexible) Generalized Minimum Residual
GMRES-DR	GMRES with deflated restarting
GMRES(m)	Restarted GMRES with m restarts
GMRES-IR	GMRES-based iterative refinement
GMRES-LSIR	GMRES-IR for least squares problems
GCR	Generalized conjugate residual algorithm
GCRO	GCR with implicit inner orthogonalization
GCRO-DR	GCRO with deflated restarting
GCROT	Truncated GCRO
CG	Conjugate Gradient
GEPP	Gaussian Elimination with Partial Pivoting
SVD	Singular Value Decomposition
(M)GS	(Modified) Gram-Schmidt
(B)CGS	(Block) Classical Gram-Schmidt
(B)CGS(I+)	(B)CGS with reorthogonalization

List of publications

Journals

J2 Oktay, E., Carson, E. (2023). Mixed precision Rayleigh quotient iteration for total least squares problems. Numerical Algorithms. https://doi.org/10.1007/s11075-023-01665-z

J1 Oktay, E., Carson, E. (2022). Multistage mixed precision iterative refinement. Numerical Linear Algebra With Applications, 29(4), e2434. https://doi.org/10.1002/nla.2434

Peer-reviewed conference proceedings

P2 Oktay, E., Carson, E. (2023). Using Mixed Precision in Low-Synchronization Reorthogonalized Block Classical Gram-Schmidt. PAMM, 23(1), e202200060. https://doi.org/10.1002/pamm.202200060

P1 Oktay, E., Carson, E. (2022). Mixed precision GMRES-based iterative refinement with recycling. In: Chleboun, J., Kůs, P., Papež, J., Rozložník, M., Segeth, K. and Šístek, J. (eds.): Programs and Algorithms of Numerical Mathematics. Proceedings of Seminar. Jablonec nad Nisou, June 19-24, 2022. Institute of Mathematics CAS, Prague, 2023. pp. 149-162

Introduction

Numerical linear algebra is the core of scientific computing. To solve real-life problems, one needs to use matrix computations. However, with increasingly complex and large problems, current hardware and algorithms have become insufficient and the need for high-performance computing has risen. Advances in modern computer architectures, such as GPU Tensor Cores, have offered significant potential performance improvements. However, to take advantage of such improvements, we must design algorithms and software that target these high-performance architectures. This thesis aims to focus only on increasing the software performance.

An efficient algorithm should find accurate solutions as fast as possible. To enhance the efficiency of an algorithm one can use several modern techniques, such as mixed-precision arithmetic. For many years, people have already been using mixed-precision arithmetic to obtain speedups in numerical linear algebra and beyond.

Mixed-precision hardware has recently become commercially available, and more than 25% of the supercomputers in the TOP500 list (TOP500) now have mixed-precision capabilities. Using lower precision in algorithms can be beneficial in terms of reducing both computation and communication costs. According to the recently developed mixed-precision benchmark, (HPL-MxP), multiple supercomputers today can already exceed exascale (10^{18} floating-point operations per second) performance through the use of mixed-precision computations; see Kudo et al. [2020]. There are many current efforts towards developing mixed-precision numerical linear algebra algorithms, which can lead to speedups in real applications; see, e.g., Haidar et al. [2018]. These new algorithms are increasingly being implemented in libraries, such as the Matrix Algebra on GPU and Multicore Architectures (MAGMA) library (Tomov et al. [2010]).

Motivated by this, the aim of this thesis is to further the state of the art in developing and analyzing mixed-precision variants of iterative methods. Iterative methods for solving linear systems and least squares problems are useful in practice when the coefficient matrix is large and sparse, or not explicitly stored and/or when accuracy less than machine precision is sufficient. An iterative method starts with an initial guess x_0, then iteratively improves the solution to the desired accuracy. One can use stationary methods (e.g., iterative refinement), Krylov subspace methods, or some hybrid approach depending on the problem. For detailed information about iterative methods, see Chapter 6 in Saad [2003].

Mixed-precision computation is useful beyond merely decreasing the total cost of an algorithm via the use of lower precision. For example, in communication-avoiding approaches, we can use *higher* precision in select parts of the algorithm to *regain stability*. In this case, there is a trade-off between the computation cost and stability of the algorithm. For detailed information and examples of this, see Chapters 4 and 5.

A floating-point number is of the form,

$$y = \pm m \times \beta^{e-t}$$

where β is the base, t is the precision, and $e \in [e_{min}, e_{max}]$ is the exponent. The

Table 1: Various IEEE precisions and their units roundoff.

Precision	Unit Roundoff
fp16 (half)	$4.88 \cdot 10^{-4}$
fp32 (single)	$5.96 \cdot 10^{-8}$
fp64 (double)	$1.11 \cdot 10^{-16}$
fp128 (quad)	$9.63 \cdot 10^{-35}$

number of bits in each part of a floating-point number depends on the desired binary format. Table 1 shows the number of bits in a number in several precisions.

Using finite precision arithmetic restricts the representation of values in terms of magnitude and precision. Thus when one converts a number to a different precision, it introduces round-off errors. The rounding errors may propagate in each step of an algorithm or it can amplify at some point. This risk increases especially when we use a lower precision such as fp16 since they can cause underflow or overflow. To reduce this risk, one should analyze rounding errors a prior. To understand the perturbation behavior of an algorithm, one can analyze its numerical stability. In numerical linear algebra, numerical stability can be defined using forward and backward errors.

The organization of the thesis is as follows. Chapters 1-3 give the introductory material while the core of the thesis consists of Chapters 4-9 which contain novel results. Chapter 1 explains the motivation for using mixed-precision in numerical linear algebra computations and how to use different precisions in an algorithm. Chapter 2 gives background information on iterative methods and their classes such as stationary iterative, Krylov subspace, and iterative refinement methods. Chapter 3 explains the use of orthogonalization processes in iterative methods. Chapter 4 includes the publication Oktay and Carson [2023b] which introduces new low-synchronized and mixed-precision variants of the block Classical Gram-Schmidt algorithm. Chapter 5 introduces reorthogonalized variants of the block Classical Gram-Schmidt algorithm with the Pythagorean theorem. Chapter 6 includes the journal publication Oktay and Carson [2023a] which introduces an iterative method combining stationary and Krylov subspace approaches (hybrid) for solving total least squares problems in mixed-precision. Chapters 7-9 discuss recently developed variants of GMRES-based iterative refinement algorithm for various problems. Chapter 7 includes the publication Oktay and Carson [2022b] introducing a variant with recycling while Chapter 8 introduces a variant for solving weighted least squares problems using FGMRES. Finally, Chapter 9 includes the journal publication Oktay and Carson [2022a] introducing a multistage iterative refinement algorithm using GMRES-based approach. Chapters including publications are pre-copyedited, author-produced versions of the published articles following peer review due to format incompatibility of the published PDFs and copyright reasons. The versions of record can be found in the footnotes under each chapter title.

Bibliography

A. Haidar, S. Tomov, J. Dongarra, and N. J. Higham. Harnessing GPU tensor cores for fast fp16 arithmetic to speed up mixed-precision iterative refinement

solvers. In *SC18: International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 603–613, 2018. doi: 10.1109/SC.2018.00050.

HPL-MxP. HPL-MxP mixed-precision benchmark. `https://icl.bitbucket.io/hpl-ai/`, November 2019.

Shuhei Kudo, Keigo Nitadori, Takuya Ina, and Toshiyuki Imamura. Prompt report on exa-scale HPL-AI benchmark. In *2020 IEEE Int. Conf. Cluster Comput. (CLUSTER)*, pages 418–419. IEEE, 2020.

Eda Oktay and Erin Carson. Multistage mixed precision iterative refinement. *Numerical Linear Algebra with Applications*, 29(4):e2434, 2022a.

Eda Oktay and Erin Carson. Mixed precision GMRES-based iterative refinement with recycling. In Jan Chleboun, Pavel Kůs, Jan Papež, Miroslav Rozložník, Karel Segeth, and Jakub Šístek, editors, *Programs and Algorithms of Numerical Mathematics, Proceedings of Seminar*, pages 149–162. Institute of Mathematics CAS, 2022b.

Eda Oktay and Erin Carson. Mixed precision Rayleigh quotient iteration for total least squares problems. *Numerical Algorithms*, pages 1–22, 2023a. ISSN 1017-1398. doi: 10.1007/s11075-023-01665-z.

Eda Oktay and Erin Carson. Using mixed precision in low-synchronization reorthogonalized block classical Gram-Schmidt. *PAMM*, 23(1):e202200060, 2023b.

Yousef Saad. *Iterative methods for sparse linear systems*. SIAM, 2003.

Stanimire Tomov, Jack Dongarra, and Marc Baboulin. Towards dense linear algebra for hybrid GPU accelerated manycore systems. *Parallel Computing*, 36 (5-6):232–240, June 2010. ISSN 0167-8191. doi: 10.1016/j.parco.2009.12.005.

TOP500. TOP500. Online, June 2021. URL `https://www.top500.org/`.

1. Mixed-precision in numerical linear algebra

To solve large and complex problems in computational science, supercomputers and high-performance platforms are necessary. This need has become one of the motivations for the emergence of exascale computers, performing 10^{18} floating point operations per second (Bergman et al. [2008]). However, the current state-of-the-art algorithms are unsuitable for such hardware systems to solve problems efficiently since they were not designed for modern computer architecture. Thus, adaptation and development of new high-performance algorithms has become the key goal.

The increasing availability of half-precision, and even quarter-precision, in hardware has brought attention to the aim of using them to reduce computation and communication costs in numerical linear algebra. However, using lower precision may cause a loss of information and accuracy. The idea behind mixed-precision algorithms is that different precisions can be used in different parts of the algorithm to have both cheap and accurate algorithms. Section 1.3 outlines how to construct an algorithm in mixed-precision.

It was shown in Haidar et al. [2018] that using NVIDIA GPU Tensor Cores can provide up to 4× speed-up when mixed-precision iterative refinement algorithm is used with fp16. Furthermore, mixed-precision iterative refinement algorithms form the basis for the recently developed high-performance LINPACK – artificial intelligence benchmark (HPL-MxP), on which today's top supercomputers in the TOP500 list in TOP500 exceed exascale performance. With this motivation, there has been a growing interest in developing mixed-precision libraries, e.g., Abdelfattah et al. [2019a] and algorithms such as Higham and Pranesh [2021].

This chapter discusses how to devise a mixed-precision algorithm and the crucial points for stability. To study the stability of an algorithm in finite precision arithmetic, one needs to be familiar with floating point arithmetic. Section 1.1 describes the key ingredients of floating point arithmetic. Then, after discussing how to model the cost of algorithms in Section 1.2, we explain how to construct a mixed-precision algorithm in Section 1.3. Finally, Section 1.4 gives an overview of the mixed-precision numerical algorithms in linear algebra.

1.1 Numerical stability and floating point arithmetic

In an algorithm, due to round-off errors, the computed result can be seen as the exact solution to a nearby problem containing slightly perturbed input data. This perturbation is called the backward error. For a function $f : \mathbb{R}^n \to \mathbb{R}^n$, the backward error in the approximation y to $f(x)$ is the smallest Δx such that $y = f(x + \Delta x)$, i.e., (Higham [2002])

$$\eta(y) = \min\{\epsilon : y = f(x + \Delta x), \|\Delta x\| \leq \epsilon \|x\|\}.$$

Backward error analysis (Wilkinson [1963]) aims to derive a bound on the backward error. If the backward error is small, then we say the algorithm is backward

stable. The forward error measures the difference between the computed and the exact solution. As defined in Higham [2002], the relative forward error of $y \approx f(x)$ can bounded in terms of the relative backward error $\eta(y)$ by

$$\frac{\|y - f(x)\|}{\|f\|} \leq cond(f, x)\eta(y) + O(\eta(y))^2,$$

where

$$cond(f, x) = lim_{\epsilon \to 0} \sup_{\|\Delta x\| \leq \epsilon \|x\|} \frac{\|f(x + \Delta x) - f(x)\|}{\epsilon \|f(x)\|}$$

is the condition number, which measures the sensitivity of the solution to small perturbations in the input data. In other words, as a rule of thumb,

$$\text{forward error} \lesssim \text{condition number} \times \text{backward error}.$$

Error analysis is important for determining how rounding errors propagate in computations and identifying potential sources of amplification. Round-off errors are inevitable consequences of using finite precision arithmetic. A floating-point operation *op* between two real numbers a, b satisfies $fl(a \text{ op } b) = (a \text{ op } b)(1 + \delta), |\delta| \leq u$, where $fl(.)$ is the computed value of an expression, u is the working precision, and δ is the round-off error (Higham [2002]). Carrying through a complete rounding error analysis of matrix computations according to this basic model in a way that allows insight into the numerical behavior is often a complicated and highly technical task.

1.2 Modeling the cost of algorithms

The $\alpha - \beta - \gamma$ model in Chan et al. [2007] is used for estimating the cost of algorithms. In this performance model, γ (seconds per flop) corresponds to the cost of a floating-point operation (flop), α (seconds) is the latency cost of a message, and β (seconds/word) is the inverse bandwidth cost. Using these elements, the cost (seconds) of a computation that performs F flops and sends S messages consisting of W words is $\gamma F + \alpha S + \beta W$. If computation can be overlapped with communication, then the cost is $\max(\gamma F, \alpha S + \beta W)$ (Thakur et al. [2005]). For the performance analysis given in Chapter 6 we use this performance model.

1.3 Construction of a mixed-precision algorithm

There are various crucial points to be aware of when constructing a mixed-precision algorithm. Each part of the algorithm should be analyzed in terms of cost and required accuracy. Lower precisions can be used in the most costly parts of an algorithm to reduce cost. However, the desired accuracy restricts the precision choice. To find a suitable precision setting, one must analyze the numerical stability of the algorithm.

For instance, the general iterative refinement (IR) approach is given in Algorithm 1. In the standard approach, line 1 is usually solved using LU factors of $A \in \mathbb{R}^{n \times n}$ computed in precision u_f, i.e., precision having unit roundoff u_f.

However, since LU decomposition performs $\mathcal{O}(n^3)$ operations, if A is large, the cost of the IR algorithm will be dominated by LU factorization. Thus, it is desirable that u_f is a precision lower than the working precision u. On the other hand, the residual r should be computed accurately since it affects the attainable forward error. Thus, the precision to calculate residual, u_r, should be at least u. With the same idea, the linear system in line 4 can be solved in a different effective precision, u_s, which depends on the solver used and the precisions therein. One obtains different mixed-precision IR schemes depending on which precision is used in which part of the algorithm.

The standard iterative refinement algorithm proposed in Wilkinson [1963] uses only two precisions: $u_f = u_s = u$ and $u_r = u^2$. The author analyzed this approach in fixed-point arithmetic. Later, the analysis using floating-point arithmetic was performed in Moler [1967]. After that, there have been several works on the error analysis of uniform and mixed precision IR using different precision settings based on componentwise or normwise errors, such as Stewart [1973], Higham [1997], and Langou et al. [2006]. For more details on these analyses, see Carson and Higham [2018]. The details of mixed precision IR are explained in Section 2.1.

Algorithm 1 General Iterative Refinement Scheme

Input: matrix $A^{n\times n}$; right-hand side b^n; maximum number of refinement steps i_{max}.

Output: Approximate solution x_{i+1} to $Ax = b$.

1: Solve $Ax_0 = b$ in precision u_f, store in precision u
2: **for** $i = 0 : i_{max} - 1$ **do**
3: Compute $r_i = b - Ax_i$ in precision u_r, store in precision u
4: Solve $Ad_{i+1} = r_i$ in precision u_s, store in precision u
5: Compute $x_{i+1} = x_i + d_{i+1}$ in precision u
6: **if** converged **then** return x_{i+1}. **end if**

1.4 Mixed-precision algorithms in numerical linear algebra

With modern hardware and floating-point arithmetic, there has been a variety of mixed-precision algorithms developed in recent years to improve performance of classical algorithms in numerical linear algebra. In particular, advancements in the development of mixed-precision precision BLAS operations (Li et al. [2002]) have enabled improved performance of algorithms via half precision GEMM (HGEMM) operations while preserving accuracy of the overall algorithm (Haidar et al. [2017]). For instance, using mixed-precision LU decomposition can accelerate the solution of linear systems in double precision (Haidar et al. [2018]). Hardware support also plays an important role in achieving the desired performance advantage. For example, Tensor Core units on NVIDIA GPUs, available in multiple different precision formats, can perform a 4×4 matrix multiplication in one cycle (Abdelfattah et al. [2019b]).

Mixed-precision can also be used to solve nonlinear equations efficiently. Newton's method is one of the most popular approaches for solving such problems.

Performing the initial iterations in low precision and gradually increasing the precision as the iterations converge can reduce the overall cost of the algorithm while keeping the accuracy on the level of the higher precision. For the analysis of mixed-precision Newton's method, see Tisseur [2001]. As explained in Section 2.1, this played a crucial role in developing mixed-precision iterative refinement schemes to solve linear systems since IR can be thought as a variation of the Newton's method. There have been several mixed-precision IR schemes introduced for solving linear systems effectively such as Langou et al. [2006] and Carson and Higham [2017]. For more information on mixed-precision iterative (refinement) approaches, see Chapter 2.

As explained above, direct methods such as LU and QR decompositions can also be performed in mixed precision. For analysis of mixed-precision LU and QR factorizations, see Higham [2002]. These decompositions can also be used within an IR scheme as explained above (Buttari et al. [2007]). If the matrix is symmetric positive definite, one can also perform a Cholesky decomposition in mixed precision (Yamazaki et al. [2015]). On the other hand, although using half precision will reduce cost of Cholesky significantly, it can suffer from overflow during rounding due to limited representation of numbers in half precision. To overcome overflow, the authors in Higham and Pranesh [2019] introduced scaling and shifting for such cases.

For more detailed information on mixed-precision numerical linear algebra algorithms, see the surveys Abdelfattah et al. [2020] and Higham and Mary [2022].

Bibliography

Ahmad Abdelfattah, Stanimire Tomov, and Jack Dongarra. Optimizing batch HGEMM on small sizes using tensor cores. 2019-03 2019a.

Ahmad Abdelfattah, Stanimire Tomov, and Jack Dongarra. Fast batched matrix multiplication for small sizes using half-precision arithmetic on GPUs. In *2019 IEEE international parallel and distributed processing symposium (IPDPS)*, pages 111–122. IEEE, 2019b.

Ahmad Abdelfattah, Hartwig Anzt, Erik G Boman, Erin Carson, Terry Cojean, Jack Dongarra, Mark Gates, Thomas Grützmacher, Nicholas J Higham, Sherry Li, et al. A survey of numerical methods utilizing mixed precision arithmetic. *arXiv preprint arXiv:2007.06674*, 2020.

Keren Bergman, Shekhar Borkar, Dan Campbell, William Carlson, William Dally, Monty Denneau, Paul Franzon, William Harrod, Kerry Hill, Jon Hiller, et al. Exascale computing study: Technology challenges in achieving exascale systems. *Defense Advanced Research Projects Agency Information Processing Techniques Office (DARPA IPTO), Tech. Rep*, 15:181, 2008.

Alfredo Buttari, Jack Dongarra, Julie Langou, Julien Langou, Piotr Luszczek, and Jakub Kurzak. Mixed precision iterative refinement techniques for the solution of dense linear systems. *The International Journal of High Performance Computing Applications*, 21(4):457–466, 2007.

Erin Carson and Nicholas J. Higham. A new analysis of iterative refinement and its application to accurate solution of ill-conditioned sparse linear systems. *SIAM Journal on Scientific Computing*, 39(6):A2834–A2856, 2017. doi: 10.1137/17M1122918.

Erin Carson and Nicholas J. Higham. Accelerating the solution of linear systems by iterative refinement in three precisions. *SIAM Journal on Scientific Computing*, 40(2):A817–A847, 2018. doi: 10.1137/17M1140819.

Ernie Chan, Marcel Heimlich, Avi Purkayastha, and Robert Van De Geijn. Collective communication: theory, practice, and experience. *Concurrency and Computation: Practice and Experience*, 19(13):1749–1783, 2007.

A. Haidar, S. Tomov, J. Dongarra, and N. J. Higham. Harnessing GPU tensor cores for fast fp16 arithmetic to speed up mixed-precision iterative refinement solvers. In *SC18: International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 603–613, 2018. doi: 10.1109/SC.2018.00050.

Azzam Haidar, Panruo Wu, Stanimire Tomov, and Jack Dongarra. Investigating half precision arithmetic to accelerate dense linear system solvers. In *Proceedings of the 8th workshop on latest advances in scalable algorithms for large-scale systems*, pages 1–8, 2017.

Nicholas J Higham. Iterative refinement for linear systems and LAPACK. *IMA Journal of Numerical Analysis*, 17(4):495–509, 1997.

Nicholas J Higham. *Accuracy and stability of numerical algorithms*. SIAM, 2002.

Nicholas J Higham and Theo Mary. Mixed precision algorithms in numerical linear algebra. *Acta Numerica*, 31:347–414, 2022.

Nicholas J Higham and Srikara Pranesh. Simulating low precision floating-point arithmetic, mims eprint 2019.4. *Manchester Institute for Mathematical Sciences, The University of Manchester, UK*, 2019.

Nicholas J. Higham and Srikara Pranesh. Exploiting lower precision arithmetic in solving symmetric positive definite linear systems and least squares problems. *SIAM Journal on Scientific Computing*, 43(1):A258–A277, 2021. doi: 10.1137/19M1298263.

HPL-MxP. HPL-MxP mixed-precision benchmark. `https://icl.bitbucket.io/hpl-ai/`, November 2019.

Julie Langou, Julien Langou, Piotr Luszczek, Jakub Kurzak, Alfredo Buttari, and Jack Dongarra. Exploiting the performance of 32 bit floating point arithmetic in obtaining 64 bit accuracy (revisiting iterative refinement for linear systems). In *Proceedings of the 2006 ACM/IEEE Conference on Supercomputing*, pages 113–es, 2006.

Xiaoye S Li, James W Demmel, David H Bailey, Greg Henry, Yozo Hida, Jimmy Iskandar, William Kahan, Suh Y Kang, Anil Kapur, Michael C Martin, et al.

Design, implementation and testing of extended and mixed precision BLAS. *ACM Transactions on Mathematical Software (TOMS)*, 28(2):152–205, 2002.

Cleve B Moler. Iterative refinement in floating point. *Journal of the ACM (JACM)*, 14(2):316–321, 1967.

Gilbert W Stewart. *Introduction to matrix computations*. Elsevier, 1973.

Rajeev Thakur, Rolf Rabenseifner, and William Gropp. Optimization of collective communication operations in MPICH. *The International Journal of High Performance Computing Applications*, 19(1):49–66, 2005.

Françoise Tisseur. Newton's method in floating point arithmetic and iterative refinement of generalized eigenvalue problems. *SIAM Journal on Matrix Analysis and Applications*, 22(4):1038–1057, 2001.

TOP500. TOP500. Online, June 2021. URL `https://www.top500.org/`.

James Hardy Wilkinson. *Rounding errors in algebraic processes*. Prentice-Hall, 1963.

Ichitaro Yamazaki, Stanimire Tomov, and Jack Dongarra. Mixed-precision Cholesky QR factorization and its case studies on multicore CPU with multiple GPUs. *SIAM Journal on Scientific Computing*, 37(3):C307–C330, 2015.

2. Iterative methods

Although direct methods are robust for solving sparse linear systems, they suffer from runtime and memory requirements and are hard to parallelize. To overcome these problems, iterative approaches can be used. Iterative methods have a lower memory requirement in addition to the reduced computation cost and time since the solution can often be achieved in a few iterations.

This chapter discusses various iterative methods which we focus on in this thesis. In Section 2.1, we discuss Richardson iteration and Newton's method with their connection to Rayleigh Quotient Iteration (RQI) (see, Section 6) and iterative refinement algorithms (see, Sections 7, 8, and 9). We also give insights into the motivation for using mixed precision in such methods. Then we define Krylov subspace methods in Section 2.2 and explain why these methods can be preferable over stationary methods. Lastly, we define "hybrid" iterative methods in Section 2.3 in which a Krylov subspace method is used as the inner solver in a stationary iterative method.

2.1 Stationary methods

Stationary methods are the simplest iterative methods for solving linear systems. The basic principle of stationary iterative methods is that they convert the system $Ax = b$ to $x = x + M^{-1}(b - Ax)$ by splitting $A = M - K$ so that the approximate solution in each step can be found via $x_{n+1} = Rx_n + c$, where $R = M^{-1}K$ is called the iteration matrix, x_n is the approximate solution at step n, and c is a constant (Kelley [1995]).

One of the simplest stationary methods is using $R = I - \omega A$, where ω is a scalar parameter. In this case, M becomes proportional to the identity I and hence the formula becomes $x_{n+1} = x_n + \omega(b - Ax_n)$, which is called Richardson iteration (Young [2014]).

Preconditioning is one of the common tools in iterative numerical linear algebra used for decreasing the runtime of an algorithm. In this case, M is called the preconditioner, and the preconditioned system becomes $MAx = Mb$. To solve the preconditioned system, one can again use the "preconditioned" Richardson iteration

$$x_{n+1} = x_n + M(b - Ax_n).$$

Using the preconditioner, such as the LU factors of A, the solution can be refined iteratively. Thus, this technique is also called *iterative refinement* (Wilkinson [1963]). Using IR to solve the linear system can be cost-effective since, depending on A, the solution can be refined in a few steps, reducing the computation cost, especially when A is large.

The refinement can be accomplished, for instance, by means of Newton's algorithm, which computes the zero of a function $f(x)$ according to the iterative formula

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}.$$

Mixed-precision IR methods have a long history, dating back to Wilkinson [1963]. The author gave fixed-point analysis of a two-precision variant that uses

high precision only in computing the residual. Later, Moler [1967] analyzed the same approach using floating-point arithmetic. More recently, to reduce the computation cost, researchers developed two-precision approaches in which low precision is used in the factorization step with Gaussian elimination with partial pivoting (GEPP); see, e.g., Smoktunowicz et al. [2006], Arioli and Duff [2009], Hogg and Scott [2010]. In Demmel et al. [2006], error bounds and stopping criteria were derived for IR with two precisions. The working precision is used to find the approximate solution, whereas a higher precision is used only to calculate the residual. In their study, the working precision is doubled if the convergence is too slow or the method is diverging. Parallel performance analysis of mixed-precision IR techniques is also discussed in Haidar et al. [2018]. Further details about mixed-precision in numerical methods are discussed in Abdelfattah et al. [2020].

On the other hand, due to the use of GEPP, if A is very ill-conditioned or badly scaled, the IR process with repeated GEPP solves can fail, especially in the case that GEPP is computed in lower precision. One of the methods that can overcome this problem was introduced in Carson and Higham [2018]. The authors devised an IR approach using three precisions and preconditioned GMRES to solve the correction step in IR.

For stability analysis of methods such as IR variants, we can derive forward and error bounds under a constraint on the conditioning of the coefficient matrix, $\kappa(A)$. For a non-singular square matrix, the condition number is defined as

$$\kappa_p(A) = \|A\|_p \|A^{-1}\|_p$$

with the associated norm p. The authors showed that as long as $\kappa_\infty(A) \leq u^{-1/2} u_f^{-1}$ and $u_r = u^2$, GMRES-IR provides accurate solutions with the forward and (normwise) backward errors

$$\frac{\|\hat{x} - x\|_\infty}{\|x\|_\infty} \approx \mathcal{O}(u) \quad \text{and} \quad \frac{\|b - A\hat{x}\|_\infty}{\|A\|_\infty \|x\|_\infty + \|b\|_\infty} \approx \mathcal{O}(u),$$

respectively, while what we refer to as the standard IR (SIR) algorithm in Wilkinson [1963] is guaranteed to have this forward error only if $\kappa_\infty(A) \leq u_f^{-1}$ and $u_r = u^2$.

Newton's method can also be used in solving eigenvalue problems, $x\alpha = Ax$, which is similar to solving least squares problems $Ax \approx b$. In eigenvalue problems, one seeks the eigenvalue $\alpha(x)$ of A corresponding to the eigenvector x. The eigenvalue can be found iteratively using an inverse iteration, called Rayleigh Quotient Iteration (RQI). In a few iterations, the Rayleigh quotient

$$\rho(x) = \frac{x^T A x}{x^T x}$$

can converge to the eigenvalue $\alpha(x)$. It is shown in Simoncini and Eldén [2002] that inexact RQI is equivalent to performing an inexact Newton algorithm in the unit sphere. Moreover, Tapia et al. [2018] discusses that one iteration of RQI can be viewed as performing one Newton's iteration with a normalization when the function is defined as $F(x, c) = (A - \lambda_c(x)I)x$, where $c \neq 0$ and

$$\lambda_c(x) = \rho(x) - \frac{c}{2}(x^T x - 1).$$

2.2 Krylov subspace methods

One of the most important drawbacks of the stationary methods is the linear convergence. To increase the speed, we can approximate the solution using a polynomial approximation

$$x_k = x_0 + q_{k-1}(A)r_0,$$

such that q_{k-1} is a polynomial of degree $k-1$, x_0 and $r_0 = b - Ax_0$ are the initial solution and residual, respectively, and $q_{k-1}(A)r_0 \in \mathcal{K}_k(A, r_0)$. The space

$$\mathcal{K}_k(A, r_0) = \text{span}\{r_0, Ar_0, A^2r_0, \ldots, A^{k-1}r_0\}$$

is called the Krylov subspace of order k.

Krylov subspace solvers are nonlinear methods for solving linear systems. Contrary to stationary methods, Krylov solvers' convergence rates are adapted to the problem because of nonlinearity. Thus, they can provide faster than linear convergence.

Since a basis for $\mathcal{K}_k(A, r_0)$ can be ill-conditioned, we construct an orthogonal basis $V_k = [v_1, \cdots, v_k]$ for the Krylov space $\mathcal{K}_k(A, v_1)$ using the Lanzos or Arnoldi process. Using projection processes one finds the approximate solution efficiently since instead of dealing with a large dense matrix, one will work on an upper Hessenberg matrix. More detailed information about Lanczos and Arnoldi recurrences are given in Chapter 3.

Krylov subspace methods work by selecting approximate solutions from a Krylov subspace. The search space is formed via nested Krylov subspaces, and the solution is obtained from a sequence of projections onto the search space. Depending on the algorithm and properties of the coefficient matrix A, Krylov subspace methods may involve short recurrences (e.g., CG) or long recurrences (e.g., GMRES).

Imposing the (Petrov-)Galerkin condition on the residual r_k in the Krylov space $\mathcal{K}_k$ along with the orthogonal basis V_k, Krylov subspace methods guarantee certain minimization properties. The Galerkin condition on the residual corresponds to minimizing the error vector in the norm associated with symmetric positive definite (SPD) coefficient matrix A over the approximation space, i.e.,

$$b - Ax_k = r_k \perp v, \quad \forall v \in \mathcal{K}_k.$$

This condition ensures convergence in exact arithmetic in Krylov subspace methods such as CG. If A is not SPD, the generalized version of the Galerkin condition, namely the Petrov-Galerkin condition, minimizes the residual norm over the space $A\mathcal{K}_k$ by imposing the condition

$$b - Ax_k = r_k \perp v, \quad \forall v \in A\mathcal{K}_k.$$

This condition is therefore used in Krylov subspace methods for non-symmetric coefficient matrices, such as GMRES. Thus, while CG minimizes the A-norm of the error, GMRES minimizes the 2-norm of the residual.

As discussed in Section 1.2, the total cost of an algorithm consists of computation cost and communication cost. Computation cost can be calculated via the

number of floating point operations performed per second (FLOPs) and communication cost can be found via

$$\frac{\text{number of words moved}}{\text{bandwidth}} \quad \text{and} \quad \text{number of messages} \times \text{latency}.$$

When compared in terms of time and energy, computation is cheaper than communication. Therefore, to reduce runtime of an algorithm communication should be minimized. There are several ways to reduce the communication cost of an algorithm such as using communication-avoiding (i.e., s-step or block) methods. In parallel computing, global reductions (i.e., MPI Allreduce operations), also called synchronization points, are expensive in terms of communication. One of the main ideas behind communication-avoiding algorithms, therefore, is to reduce the number of global reductions in an algorithm. There have been various communication-avoiding Krylov subspace methods introduced over time, e.g., Chronopoulos and Gear [1989] and Hoemmen [2010]. In communication-avoiding Krylov subspace methods, one computes a block of $s > 1$ iterations at once, which requires only one global reduction per s iterations. Such methods, and block Krylov subspace methods in general, necessitate the use of a block orthogonalization process such as Block Classical Gram-Schmidt (BCGS). For detailed information, see Chapters 3 and 4.

2.3 Hybrid iterative methods

Throughout this thesis, we refer to stationary iterative methods which use a Krylov subspace method as an inner solver as *hybrid iterative methods*. We give a brief motivation for such methods below.

SIR first computes the initial approximation $\hat{x_0}$ using Gaussian elimination with partial pivoting (GEPP). It then saves the approximate factorization $A \approx \hat{L}\hat{U}$ and uses these factors to solve for the correction term $A\hat{d} = (\hat{L}\hat{U})\hat{d} = \hat{r}$. Using $\hat{d}$, SIR finally refines the current solution $\hat{x} = \hat{x} + \hat{d}$.

SIR has various drawbacks depending on the input matrix. If the matrix is very ill-conditioned with respect to the precision in which the LU factorization is computed or badly scaled, the IR process with repeated GEPP solves can fail. Moreover, the error can grow with each refinement step if the matrix is extremely ill-conditioned. To overcome these drawbacks, various IR variants have been developed. For instance, the authors in Carson and Higham [2017] introduced an example of a hybrid iterative method, GMRES-based IR (GMRES-IR).

To solve the correction step in IR, GMRES-IR uses preconditioned GMRES as an inner solver. For the preconditioner, the algorithm computes LU factors of the input matrix $A \approx \hat{L}\hat{U}$ in the beginning. GMRES-IR is analyzed in Carson and Higham [2018] with three precisions, (u_r, u, u_f), where u_r is the precision to compute the residual, u is the working precision, and u_f is the LU factorization precision. There are also studies involving a five-precision variant, GMRES-IR5 (Amestoy et al. [2021]). It is shown in Carson and Higham [2018] that GMRES-IR can solve more ill-conditioned linear systems than SIR for a given set of precisions. Constraints on the condition number for SIR and GMRES-IR to provide accurate solutions in the accuracy of u are given in Tables 2.1 and 2.2, respectively.

Table 2.1: Bounds on $\kappa_\infty(A)$ for the relative normwise and columnwise backward and forward errors of SIR

u_f	u	u_r	$\kappa_\infty(A)$	Backward error		
				Normwise	Componentwise	Forward error
half	single	double	10^4	single	single	single
half	double	quad	10^4	double	double	double
single	double	quad	10^8	double	double	double

Table 2.2: Bounds on $\kappa_\infty(A)$ for the relative normwise and columnwise backward and forward errors of GMRES-IR

u_f	u	u_r	$\kappa_\infty(A)$	Backward error		
				Normwise	Componentwise	Forward error
half	single	double	10^8	single	single	single
half	double	quad	10^{12}	double	double	double
single	double	quad	10^{16}	double	double	double

Bibliography

Ahmad Abdelfattah, Hartwig Anzt, Erik G Boman, Erin Carson, Terry Cojean, Jack Dongarra, Mark Gates, Thomas Grützmacher, Nicholas J Higham, Sherry Li, et al. A survey of numerical methods utilizing mixed precision arithmetic. *arXiv preprint arXiv:2007.06674*, 2020.

Patrick Amestoy, Alfredo Buttari, Nicholas J. Higham, Jean-Yves L'Excellent, Theo Mary, and Bastien Vieublé. Five-precision GMRES-based iterative refinement. Technical Report 2021.5, April 2021. URL `http://eprints.maths.manchester.ac.uk/2807/`.

Mario Arioli and Iain S Duff. Using FGMRES to obtain backward stability in mixed precision. *Electronic Transactions on Numerical Analysis*, 33:31–44, 2009.

Erin Carson and Nicholas J. Higham. A new analysis of iterative refinement and its application to accurate solution of ill-conditioned sparse linear systems. *SIAM Journal on Scientific Computing*, 39(6):A2834–A2856, 2017. doi: 10.1137/17M1122918.

Erin Carson and Nicholas J. Higham. Accelerating the solution of linear systems by iterative refinement in three precisions. *SIAM Journal on Scientific Computing*, 40(2):A817–A847, 2018. doi: 10.1137/17M1140819.

A.T Chronopoulos and C.W Gear. On the efficient implementation of preconditioned s-step conjugate gradient methods on multiprocessors with memory hierarchy. *Parallel Computing*, 11(1):37–53, 1989. ISSN 0167-8191. doi: https://doi.org/10.1016/0167-8191(89)90062-8.

James Demmel, Yozo Hida, William Kahan, Xiaoye S. Li, Sonil Mukherjee, and E. Jason Riedy. Error bounds from extra-precise iterative refinement. *ACM Trans. Math. Softw.*, 32(2):325–351, June 2006. ISSN 0098-3500. doi: 10.1145/1141885.1141894.

A. Haidar, S. Tomov, J. Dongarra, and N. J. Higham. Harnessing GPU tensor cores for fast fp16 arithmetic to speed up mixed-precision iterative refinement solvers. In *SC18: International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 603–613, 2018. doi: 10.1109/SC.2018.00050.

Mark Hoemmen. *Communication-avoiding Krylov subspace methods*. University of California, Berkeley, 2010.

Jonathan D Hogg and Jennifer A Scott. A fast and robust mixed-precision solver for the solution of sparse symmetric linear systems. *ACM Transactions on Mathematical Software (TOMS)*, 37(2):1–24, 2010.

C. T. Kelley. *Iterative Methods for Linear and Nonlinear Equations*. Society for Industrial and Applied Mathematics, 1995. doi: 10.1137/1.9781611970944.

Cleve B Moler. Iterative refinement in floating point. *Journal of the ACM (JACM)*, 14(2):316–321, 1967.

Valeria Simoncini and Lars Eldén. Inexact Rayleigh quotient-type methods for eigenvalue computations. *BIT Numerical Mathematics*, 42(1):159–182, 2002.

Alicja Smoktunowicz, Jesse Barlow, and Julien Langou. A note on the error analysis of classical Gram-Schmidt. *Numerische Mathematik*, 105, 07 2006. doi: 10.1007/s00211-006-0042-1.

Richard A. Tapia, J. E. Dennis, and Jan P. Schäfermeyer. Inverse, shifted inverse, and Rayleigh quotient iteration as Newton's method. *SIAM Review*, 60(1):3–55, 2018. doi: 10.1137/15M1049956.

James Hardy Wilkinson. *Rounding errors in algebraic processes*. Prentice-Hall, 1963.

David M Young. *Iterative solution of large linear systems*. Elsevier, 2014.

3. Orthogonalization processes

As discussed in Section 2.2, Krylov subspace methods form an orthogonal Krylov basis, $\mathcal{K}_k = \mathcal{K}_k(A, r_0) = \text{span}\{r_0, Ar_0, A^2r_0, \ldots, A^{k-1}r_0\}$, where $r_0 = b - Ax_0$ is the initial residual. Using this basis together with the Petrov-Galerkin condition guarantees certain minimization properties of the method.

$\mathcal{K}_k$ is generated by the Arnoldi or Lanczos methods or their variants. The Arnoldi process with full orthogonalization computes the recurrence

$$AV_m = V_m H_m + h_{m+1,m} v_{m+1} e_m^T = V_{m+1} \bar{H}_m,$$

where

$$\bar{H}_m = \begin{bmatrix} H_m \\ h_{m+1} e_m^T \end{bmatrix}$$

is an upper Hessenberg matrix and $V_m = [v_1, v_2, \ldots, v_m]$ has orthonormal basis vectors as columns for $m = 1, \ldots$.

In exact arithmetic, the Arnoldi vectors $[v_1, v_2, \ldots, v_m]$ are orthogonal. There are several orthogonalization methods that can be used in Arnoldi, such as Householder, Gram-Schmidt, and Givens rotations which differ in their cost and numerical properties. The Gram-Schmidt process is the cheapest and easiest approach to parallelize whereas Householder and Givens rotations better preserve orthogonality, i.e., $\|I - \bar{Q}^T \bar{Q}\|_2 < \mathcal{O}(u)$, where $A \approx \bar{Q}\bar{R}$ are computed QR factors in finite precision, and u is unit round-off. In long-recurrence methods such as GMRES, one needs to use an explicit orthogonalization scheme such as Gram-Schmidt to orthonormalize the vectors generated.

The Gram-Schmidt process is the oldest method for $A = QR$, where $Q^TQ = I$ and R is an upper triangular matrix. The Classical Gram-Schmidt (CGS) process in Algorithm 2 can be geometrically defined as using orthogonal projection of the vectors onto a subspace generated by the previously computed orthogonal vectors. The main drawback of the CGS algorithm is that the loss of orthogonality is bounded by

$$\|I - \bar{Q}^T \bar{Q}\|_2 \leq \mathcal{O}(u)\kappa^{n-1}(A)$$

as long as $\mathcal{O}(u)\kappa(A) \leq 1$ Kiełbasiński [1974] and thus, it is not stable, as explained later below.

Algorithm 2 CGS

1: **for** $j = 1$: n **do**
2: **for** $i = 1$: $j - 1$ **do**
3: $r_{ij} = q_i^T a_j$
4: $q'_j = a_j - \sum_{k=1}^{j-1} r_{kj} q_k$
5: $r_{jj} = \|q'_j\|_2$
6: $q_j = q'_j / r_{jj}$

To have a more stable algorithm, one can change the order of computations in CGS and obtain a mathematically equivalent variant. The Modified Gram-Schmidt (MGS) process is given in Algorithm 3. Contrary to CGS, which updates all vectors at once and calculates the R-factor using the original input A, MGS

updates each vector once in each step and calculates the R-factor using the previously orthogonalized vectors. These differences make MGS more stable than CGS with loss of orthogonality given by $\|I - \bar{Q}^T \bar{Q}\|_2 < \mathcal{O}(u)\kappa_2(A)$. In fact, Paige et al. [2006] proved that the GMRES algorithm is backward stable if it is used with MGS (MGS-GMRES).

Algorithm 3 Modified Gram-Schmidt

1: $a_k^{(1)} = a_k$
2: **for** $k = 1{:}\ n$ **do**
3: $\quad r_{kk} = \|a_k^{(k)}\|_2$
4: $\quad q_k = a_k^{(k)} / r_{kk}$
5: $\quad$ **for** $j = k+1{:}\ n$ **do**
6: $\quad\quad r_{kj} = q_k^T a_j^{(k)}$
7: $\quad\quad a_j^{(k+1)} = a_j^{(k)} - r_{kj} q_k$

Although CGS is preferable to MGS due to the fewer number of synchronization points, one of the drawbacks of CGS is that in finite precision, orthogonality can be easily lost (Giraud et al. [2005]). This loss is due to the accumulation of minor round-off errors occurring in every step, resulting in Q no longer having orthonormal columns and hence QR no longer giving the same matrix as A. In Abdelmalek [1971], the authors show that using reorthogonalization in CGS, one can obtain vectors that are orthogonal to the level of machine precision. This idea was extended to the block setting in Barlow and Smoktunowicz [2013], in which reorthogonalized BCGS (BCGS2) is introduced and analyzed.

Block Gram-Schmidt (BGS) algorithms are necessary for block and s-step Krylov subspace methods. As a standalone orthogonalization scheme, the block approach also has performance benefits. Instead of operating column by column, which requires the use of BLAS1 operations (vector-vector operations), BGS uses blocks of columns enabling the use of BLAS3 operations (matrix-matrix operations) which reduces communication cost and improves performance significantly (Dongarra et al. [1990]).

There are several variants of BGS. To define a BGS algorithm, one needs an orthogonalization method for interblock orthogonalization, and a non-block orthogonalization algorithm for intrablock orthogonalization (Hoemmen [2010]). There are several choices for intrablock and interblock algorithms, and combinations of these methods will result in different numerical and performance properties. For instance, one of the interblock methods Block MGS (BMGS) or Block CGS (BCGS) can be used with an intrablock method such as CGS, MGS, or CholeskyQR . The authors in Yamazaki et al. [2015] use mixed-precision CholeskyQR and BMGS together to balance performance and accuracy.

Bibliography

Nabih N. Abdelmalek. Round off error analysis for Gram-Schmidt method and solution of linear least squares problems. *BIT Numerical Mathematics*, 11: 345–367, 1971.

Jesse L Barlow and Alicja Smoktunowicz. Reorthogonalized block classical Gram–Schmidt. *Numerische Mathematik*, 123(3):395–423, 2013.

J. J. Dongarra, Jeremy Du Croz, Sven Hammarling, and I. S. Duff. A set of level 3 basic linear algebra subprograms. *ACM Trans. Math. Softw.*, 16(1):1–17, 1990. ISSN 0098-3500. doi: 10.1145/77626.79170.

Luc Giraud, Julien Langou, Miroslav Rozložník, and Jasper Eshof. Rounding error analysis of the classical Gram–Schmidt orthogonalization. *Numerische Mathematik*, 101:87–100, 01 2005. doi: 10.1007/s00211-005-0615-4.

Mark Hoemmen. *Communication-avoiding Krylov subspace methods*. University of California, Berkeley, 2010.

A Kiełbasiński. Analiza numeryczna algorytmu ortogonalizacji Grama-Schmidta. *Mathematica Applicanda*, 2(2), 1974.

Christopher C. Paige, Miroslav Rozložník, and Zdeněk Strakoš. Modified Gram-Schmidt (MGS), least squares, and backward stability of MGS-GMRES. *SIAM Journal on Matrix Analysis and Applications*, 28(1):264–284, 2006. doi: 10.1137/050630416.

Ichitaro Yamazaki, Stanimire Tomov, Jakub Kurzak, Jack Dongarra, and Jesse Barlow. Mixed-precision block Gram Schmidt orthogonalization. In *Proceedings of the 6th Workshop on Latest Advances in Scalable Algorithms for Large-Scale Systems*, pages 1–8, 2015.

4. Using mixed-precision in low-synchronization reorthogonalized block classical Gram-Schmidt[1]

Block Gram-Schmidt (BGS) algorithms are used for computing the QR factorization of a given matrix $\mathcal{X} \in \mathbb{R}^{m\times n}$, $\mathcal{X} = \mathcal{Q}\mathcal{R}$, where $\mathcal{Q}$ is an orthogonal matrix and $\mathcal{R}$ is an upper triangular matrix. We assume $\mathcal{X}$ is partitioned into blocks of size $m \times s$, i.e., $\mathcal{X} = [X_1, \ldots, X_p]$, where $X_i \in \mathbb{R}^{m\times s}$, under the assumption of $n/s = p$. The matrices $\mathcal{Q}$ and $\mathcal{R}$ are partitioned in a similar manner, i.e., $\mathcal{Q} = [Q_1, \ldots, Q_p]$, where $Q_i \in \mathbb{R}^{m\times s}$ and $\mathcal{R}_{j,k} = R_{j,k}$, where $R_{j,k} \in \mathbb{R}^{s\times s}$. BGS algorithms are widely used in block Krylov subspace methods such as block GMRES (Baker et al. [2006]), and in communication-avoiding Krylov subspace methods such as CA-GMRES (Ballard et al. [2014]). Using a block approach in Krylov subspace methods can improve performance by enabling the use of BLAS-3 operations, improving convergence behavior, and/or reducing the communication cost.

In parallel settings, the inner product and norm operations within BGS algorithms require global reductions, i.e., synchronizations. This means that all compute nodes of a machine need to synchronize and exchange information. In large-scale settings, this can become a major computational bottleneck depending on the particular BGS variant and particular parallel setting. BGS algorithms also suffer from loss of orthogonality, i.e., the matrix $\mathcal{Q}$ is not exactly orthogonal when computed in finite precision. The loss of orthogonality is defined as the quantity $\|I - \bar{\mathcal{Q}}^T\bar{\mathcal{Q}}\|_2$, where $\bar{\mathcal{Q}}$ is the computed Q factor of $\mathcal{X}$. The loss of orthogonality plays an important role in the stability of Krylov subspace methods. For instance, it was proved in Paige et al. [2006] that the level of orthogonality provided by the modified Gram-Schmidt (MGS) algorithm, i.e., $\|I - \bar{\mathcal{Q}}^T\bar{\mathcal{Q}}\|_2 < \mathcal{O}(u)\kappa(\mathcal{X})$, where u is the unit round-off (for double precision, $u \approx 10^{-16}$), is sufficient to guarantee a backward stable GMRES algorithm. This motivates us to find a BGS variant that achieves at least $\mathcal{O}(u)\kappa(\mathcal{X})$ loss of orthogonality and simultaneously requires a small number of global synchronizations.

One of the most common BGS variants is the block classical Gram-Schmidt (BCGS) algorithm (Saad [2003]) given in Algorithm 4. Like all BGS variants, BCGS uses a non-block orthogonalization algorithm referred to as `IntraOrtho` for intrablock orthogonalization, such as Householder QR, CGS, MGS, or Cholesky QR. Note that BCGS requires one synchronization in line 3 and potentially one or more synchronizations in line 5, depending on what is used as the `IntraOrtho`.

[1]This chapter is a pre-copyedited, author-produced version of an article accepted for publication in Wiley: Proceedings in Applied Mathematics and Mechanics following peer review. The version of record [Proceedings in Applied Mathematics and Mechanics, Oktay, E., Carson, E.: Using Mixed Precision in Low-Synchronization Reorthogonalized Block Classical Gram-Schmidt, 2023] is available online at `https://onlinelibrary.wiley.com/doi/pdf/10.1002/pamm.202200060`.

Algorithm 4 BCGS (Saad [2003])

1: $[Q_1, R_{11}] = \texttt{IntraOrtho}(X_1)$
2: **for** $k = 1\colon p-1$ **do**
3: $\quad R_{1:k,k+1} = Q_{1:k}^T X_{k+1}$
4: $\quad W = X_{k+1} - Q_{1:k} R_{1:k,k+1}$
5: $\quad [Q_{k+1}, R_{k+1,k+1}] = \texttt{IntraOrtho}(W)$

According to the conjecture in Kiełbasiński [1974], the loss of orthogonality in BCGS is bounded by $\|I - \bar{\mathcal{Q}}^T \bar{\mathcal{Q}}\|_2 < \mathcal{O}(u)\kappa^{n-1}(\mathcal{X})$ as long as $\mathcal{O}(u)\kappa(\mathcal{X}) < 1$. Moreover, if the diagonal blocks of $\mathcal{R}$ are computed in a special way using Cholesky, the authors in Carson et al. [2021] proved that $\|I - \bar{\mathcal{Q}}^T \bar{\mathcal{Q}}\|_2 < \mathcal{O}(u)\kappa^2(\mathcal{X})$ holds provided that $\mathcal{O}(u)\kappa^2(\mathcal{X}) < 1$.

Reorthogonalized Variants (BCGSI+)

Round-off errors and cancellation causes BCGS to lose orthogonality which makes the algorithm unstable. To overcome this problem, one can use reorthogonalization.

To use reorthogonalization in the BCGS algorithm, instead of calculating the final R in lines 3-5 of Algorithm 4, the orthogonalization is performed two times and the resulting R factors are combined. This variant of BCGS (BCGSI+) given in Algorithm 5 was analyzed by the authors in Barlow and Smoktunowicz [2013]. The algorithm orthogonalizes for the first time in lines 4-6. Then using the previously computed $\hat{Q}$, the vectors are orthogonalized for the second time in lines 7-9. The R factors are then combined in lines 10-11. Note that BCGSI+ requires twice as many synchronizations as BCGS.

Algorithm 5 BCGSI+ (Barlow and Smoktunowicz [2013])

1: Allocate memory for Q and R
2: $[Q_1, R_{11}] = \texttt{IntraOrtho}(X_1)$
3: **for** $k = 1\colon p-1$ **do**
4: $\quad R^{(1)}_{1:k,k+1} = Q_{1:k}^T X_{k+1}$
5: $\quad W = X_{k+1} - Q_{1:k} R^{(1)}_{1:k,k+1}$
6: $\quad [\hat{Q}, R^{(1)}_{k+1,k+1}] = \texttt{IntraOrtho}(W)$
7: $\quad R^{(2)}_{1:k,k+1} = Q_{1:k}^T \hat{Q}$
8: $\quad W = \hat{Q} - Q_{1:k} R^{(2)}_{1:k,k+1}$
9: $\quad [Q_{k+1}, R^{(2)}_{k+1,k+1}] = \texttt{IntraOrtho}(W)$
10: $\quad R_{1:k,k+1} = R^{(1)}_{1:k,k+1} + R^{(2)}_{1:k,k+1} R^{(1)}_{k+1,k+1}$
11: $\quad R_{k+1,k+1} = R^{(2)}_{k+1,k+1} R^{(1)}_{k+1,k+1}$

The authors in Barlow and Smoktunowicz [2013] proved that if a method with $\|I - \bar{\mathcal{Q}}^T \bar{\mathcal{Q}}\|_2 \leq \mathcal{O}(u)$ is used as `IntraOrtho` and $\mathcal{O}(u)\kappa(\mathcal{X}) < 1$, then BCGSI+ has a loss of orthogonality bounded by $\|I - \bar{\mathcal{Q}}^T \bar{\mathcal{Q}}\|_2 \leq \mathcal{O}(u)$.

Low-Synchronization Variants (BCGSI+LS)

The goal of reducing the number of synchronizations required in Gram-Schmidt algorithms motivated work into developing so-called "low-synchronization" variants of BGS and other orthogonalization routines, which require only a single synchronization per block; see, e.g., Yamazaki et al. [2020].

The low-sync BCGSI+ algorithm (BCGSI+LS) given in Algorithm 6 was recently introduced in Yamazaki et al. [2020] for use within GMRES. The BCGSI+ LS algorithm is a block generalization of the CGSI+LS algorithm in Świrydowicz et al. [2021], which is based on computing a strictly lower triangular matrix one block of rows at a time in a single global reduction, lagging the normalization step, and merging it into this single reduction. The development of this approach was based on the work of Ruhe (Ruhe [1983]); the authors of Świrydowicz et al. [2021] observed that MGS/CGS could be interpreted as a variant of Gauss-Seidel/Gauss-Jacobi iterations for solving the normal equations where the associated orthogonal projector is given as

$$I - Q_{1:k-1} T_{1:k-1,1:k-1} Q_{1:k-1}^T, \quad \text{for} \quad T_{1:k-1,1:k-1} \approx (Q_{1:k-1}^T Q_{1:k-1})^{-1}.$$

For CGSI+LS, this T is computed iteratively via $T_{1:k-1,1:k-1} \approx I - L_{1:k-1,1:k-1} - L_{1:k-1,1:k-1}^T$. According to Carson et al. [2022], the algorithm can also be thought of as splitting $T_{1:k-1,1:k-1}$ into two parts, $I - L_{1:k-1,1:k-1}$ and a delayed reorthogonalization step $R_{1:k-1,k-1} = R_{1:k-2,k-1} - L_{k-1,1:k-2}^T$, the latter of which is applied in the next iteration.

The block generalization of this idea leads to BCGSI+LS. We note in Algorithm 6 that the block analogs of T and L above are not explicitly computed. We also note that the resulting BCGSI+LS algorithm has no explicit `IntraOrtho`, and lines 5, 8, and 18 are computed via Cholesky factorization. Asymptotically, BCGSI+LS has only one synchronization point per block, which occurs in lines 5 and 7 in Algorithm 6.

A conjecture in Carson et al. [2022] states that if $\mathcal{O}(u)\kappa^2(\mathcal{X}) < 1$, then the loss of orthogonality of BCGSI+LS satisfies $\|I - \bar{Q}^T \bar{Q}\|_2 < \mathcal{O}(u)\kappa^2(\mathcal{X})$. Thus although BCGSI+LS has the performance advantage that it only requires one synchronization per block, such a significant loss of orthogonality may make it unsuitable for use within GMRES. This motivates us to try to selectively use higher precision in some parts of the BCGSI+LS algorithm to decrease the loss of orthogonality while still maintaining a single synchronization per block.

4.1 Mixed-precision BCGSI+LS (BCGSI+LS-MP)

Our mixed precision approach, which we call BCGSI+LS-MP, is given in Algorithm 7. For a working precision u, we use the higher precision u^2 in two aspects of the algorithm: for computing the Cholesky factorizations in lines 5, 7/8, and 17/18, and in applying the corresponding inverses of the R factors in lines 9, 11, 15, and 20.

We note that BCGSI+LS-MP still has only one synchronization point per block, which occurs in lines 5 and 7. Since we now use precision u^2 in these lines,

Algorithm 6 BCGSI+LS (Yamazaki et al. [2020])

Allocate memory for $\mathcal{Q}$ and $\mathcal{R}$
$U = X_1$
for $k = 2, \ldots, p$ **do**
    if $k = 2$ **then**
        $\begin{bmatrix} R_{k-1,k-1}^T R_{k-1,k-1} & P \end{bmatrix} = U^T \begin{bmatrix} U & X_k \end{bmatrix}$
    else if $k > 2$ **then**
        $\begin{bmatrix} W & Z \\ \Omega & Y \end{bmatrix} = \begin{bmatrix} \mathcal{Q}_{1:k-2} & U \end{bmatrix}^T \begin{bmatrix} U & X_k \end{bmatrix}$
        $\begin{bmatrix} R_{k-1,k-1}^T R_{k-1,k-1} & P \end{bmatrix} = \begin{bmatrix} \Omega & Y \end{bmatrix} - W^T \begin{bmatrix} W & Z \end{bmatrix}$
    $R_{k-1,k} = R_{k-1,k-1}^{-T} P$
    if $k = 2$ **then**
        $Q_{k-1} = U R_{k-1,k-1}^{-1}$
    else if $k > 2$ **then**
        $\mathcal{R}_{1:k-2,k-1} = \mathcal{R}_{1:k-2,k-1} + W$
        $\mathcal{R}_{1:k-2,k} = Z$
        $Q_{k-1} = (U - \mathcal{Q}_{1:k-2} W) R_{k-1,k-1}^{-1}$
    $U = X_k - \mathcal{Q}_{1:k-1} \mathcal{R}_{1:k-1,k}$
$\begin{bmatrix} W \\ \Omega \end{bmatrix} = \begin{bmatrix} \mathcal{Q}_{1:s-1} & U \end{bmatrix}^T U$
$R_{s,s}^T R_{s,s} = \Omega - W^T W$
$\mathcal{R}_{1:s-1,s} = \mathcal{R}_{1:s-1,s} + W$
$Q_s = (U - \mathcal{Q}_{1:s-1} W) R_{s,s}^{-1}$
return $\mathcal{Q} = [Q_1, \ldots, Q_s], \mathcal{R} = (R_{jk})$

this means that we have doubled the size of the reduction; i.e., we have doubled the bandwidth and computation costs. In the latency-bound regime, where low-synchronization algorithms are most beneficial, this overhead may not be significant, in particular, in cases where precisions u and u^2 are both implemented in hardware. The higher precision computations in lines 8, 9, 11, and 15 are all local computations, and thus we expect the extra overhead to be insignificant. We note that lines 17, 18, and 20 are only computed once at the very end of the algorithm. Overall, the resulting overhead of using mixed precision will be highly dependent on the particular problem size and machine parameters; a performance study will be the subject of future work.

Algorithm 7 BCGSI+LS-MP

1: Allocate memory for $\mathcal{Q}$ and $\mathcal{R}$
2: $U = X_1$
3: **for** $k = 2, \ldots, p$ **do**
4: **if** $k = 2$ **then**
5: $\begin{bmatrix} R_{k-1,k-1}^T R_{k-1,k-1} & P \end{bmatrix} = U^T \begin{bmatrix} U & X_k \end{bmatrix}$ in precision u^2
6: **else if** $k > 2$ **then**
7: $\begin{bmatrix} W & Z \\ \Omega & Y \end{bmatrix} = \begin{bmatrix} \mathcal{Q}_{1:k-2} & U \end{bmatrix}^T \begin{bmatrix} U & X_k \end{bmatrix}$ in precision u^2
8: $\begin{bmatrix} R_{k-1,k-1}^T R_{k-1,k-1} & P \end{bmatrix} = \begin{bmatrix} \Omega & Y \end{bmatrix} - W^T \begin{bmatrix} W & Z \end{bmatrix}$ in precision u^2
9: $R_{k-1,k} = R_{k-1,k-1}^{-T} P$ in precision u^2
10: **if** $k = 2$ **then**
11: $Q_{k-1} = U R_{k-1,k-1}^{-1}$ in precision u^2
12: **else if** $k > 2$ **then**
13: $\mathcal{R}_{1:k-2,k-1} = \mathcal{R}_{1:k-2,k-1} + W$ in precision u
14: $\mathcal{R}_{1:k-2,k} = Z$ in precision u
15: $Q_{k-1} = (U - \mathcal{Q}_{1:k-2} W) R_{k-1,k-1}^{-1}$ in precision u^2
16: $U = X_k - \mathcal{Q}_{1:k-1} \mathcal{R}_{1:k-1,k}$ in precision u
17: $\begin{bmatrix} W \\ \Omega \end{bmatrix} = \begin{bmatrix} \mathcal{Q}_{1:s-1} & U \end{bmatrix}^T U$ in precision u^2
18: $R_{s,s}^T R_{s,s} = \Omega - W^T W$ in precision u^2
19: $\mathcal{R}_{1:s-1,s} = \mathcal{R}_{1:s-1,s} + W$ in precision u
20: $Q_s = (U - \mathcal{Q}_{1:s-1} W) R_{s,s}^{-1}$ in precision u^2
21: **return** $\mathcal{Q} = [Q_1, \ldots, Q_s], \mathcal{R} = (R_{jk})$

4.2 Numerical experiments

We now seek to demonstrate numerically, on a set of challenging test problems, that our mixed precision approach BCGSI+LS-MP improves the loss of orthogonality relative to the uniform precision approach BCGSI+LS.

To illustrate the comparison of the methods in terms of the loss of orthogonality, we performed numerical experiments in MATLAB using the block Gram-Schmidt variants available at `github.com/katlund/BlockStab` with Läuchli, monomial, and glued matrices. Each of the matrices has dimensions $[m, p, s]$, where m is the number of rows, p is the number of block vectors, and s is the number

of columns per block vector. The widths of blocks are specified by the input `svec`. For a detailed investigation of these matrices on BGS variants, see Carson et al. [2022]. The experiments are performed on a computer with AMD Ryzen 5 4500U having 6 CPUs and 8 GB RAM with OS system Ubuntu 22.04 LTS. In our numerical experiments, we used double precision in MATLAB for the working precision u, and quadruple precision for u^2. We used the Advanpix toolbox (Advanpix LLC.) to simulate quadruple precision.

Each plot shows the loss of orthogonality versus condition number for matrices $\mathcal{X}$ of a given type. The dashed black line represents the $\mathcal{O}(u)\kappa(\mathcal{X})$ bound and the black solid line represents the $\mathcal{O}(u)\kappa^2(\mathcal{X})$ bound. The dashed black line thus represents the loss of orthogonality bound for MGS, which is notable since MGS-GMRES is known to be backward stable (Paige et al. [2006]). We can thus conjecture that under certain constraints on the input matrix, an orthogonality scheme that provides this level of orthogonality can be expected to result in a backward stable GMRES implementation. The algorithm following the $\circ$ notation in the legends indicates the algorithm that is used as the `IntraOrtho` (which in our experiments, is always Householder QR).

The glued matrices introduced in Smoktunowicz et al. [2006] are $m \times n$ matrices, where $n = nglued \times nbglued$, *nglued* is the number of columns in a block, and *nbglued* is the number of blocks that are glued together. For this study, we use glued matrices with dimension $[m, p, s] = [1000, 50, 4]$ with `svec=1:12`. From Figure 4.1, we see that even when an unconditionally stable intrablock orthogonalization method is used, the loss of orthogonality in BCGS can exceed the $\mathcal{O}(u)\kappa^2(\mathcal{X})$ bound. On the other hand, when reorthogonalization is used, the loss of orthogonality remains on the level $\mathcal{O}(u)$ (note that the red and purple markers overlap). Whereas the loss of orthogonality for BCGSI+LS starts to deviate from the level $\mathcal{O}(u)$ for larger condition numbers, for BCGSI+LS-MP it remains on the level $\mathcal{O}(u)$.

The monomial test matrices are matrices $\mathcal{X}$ consisting of p block vectors $X_k = [v_k|Av_k|\cdots|A^{s-1}v_k]$, $k = 1, \ldots, p$, where A is a diagonal $m \times m$ operator with evenly distributed eigenvalues in $(0.1, 10)$, and v_k are normalized randomly generated vectors from the uniform distribution. For this study, we use monomial matrices with dimension $[m, p, s] = [1000, 120, 2]$ with `svec=2:2:12`. We see from Figure 4.2 that the behavior of BCGS and reorthogonalized variants are the same as for the glued matrices: BCGS exceeds the $\mathcal{O}(u)\kappa^2(\mathcal{X})$ bound, and using reorthogonalization helps to decrease the loss of orthogonality to below $\mathcal{O}(u)\kappa(\mathcal{X})$. Also similarly to the glued matrices, the loss of orthogonality in BCGSI+LS deviates from $\mathcal{O}(u)$ (more significantly in this case), whereas for BCGSI+LS-MP, it remains on the level $\mathcal{O}(u)$.

Our final, most interesting test case are Läuchli matrices (Läuchli [1961]) of the form

$$\mathcal{X} = \begin{bmatrix} 1 & 1 & \dots & 1 \\ \eta & & & \\ & \eta & & \\ & & \ddots & \\ & & & \eta \end{bmatrix}, \eta \in (\epsilon, \sqrt{\epsilon}),$$

where η is drawn randomly from a scaled uniform distribution. For Läuchli matrices, the columns are only barely independent, and the entries are either

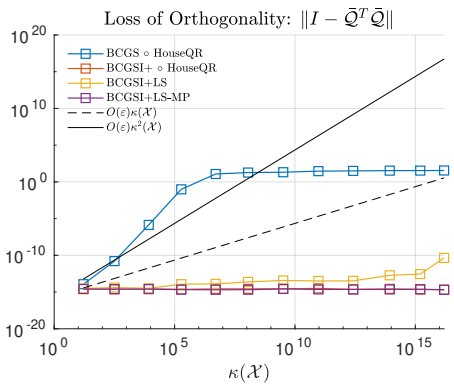

Figure 4.1: The loss of orthogonality of BCGS (blue), BCGSI+ (red), BCGSI+LS (yellow), and BCGSI+LS-MP (purple) versus condition number for glued matrices.

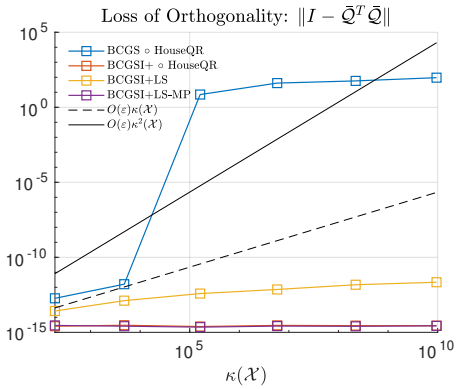

Figure 4.2: The loss of orthogonality of BCGS (blue), BCGSI+ (red), BCGSI+LS (yellow), and BCGSI+LS-MP (purple) versus condition number for monomial matrices.

zero or close to ϵ which causes a high cancellation rate. Because of its structure, Läuchli matrices are often used in numerical experiments for illuminating the effects of finite precision error in Gram-Schmidt algorithms. For this study, we use Läuchli matrices with dimension $[m, p, s] = [1000, 100, 5]$ with `svec=logspace(-1,-16,10)`. Figure 4.3 shows the loss of orthogonality for each BCGS variant mentioned above. From the figure, we see that the loss of orthogonality of BCGS is above the $\mathcal{O}(u)\kappa(\mathcal{X})$ bound. When reorthogonalization is used, we observe that BCGSI+ stays on the level $\mathcal{O}(u)$ as expected. However, unlike in previous test cases, the loss of orthogonality in BCGSI+LS is now significantly worse. We see here that it exceeds the level $\mathcal{O}(u)\kappa(\mathcal{X})$, and the scaling behavior is on par with $\mathcal{O}(u)\kappa^2(\mathcal{X})$ (which is the bound that was conjectured in Carson et al. [2022]). The use of mixed precision in BCGSI+LS-MP seems to remedy this problem. The loss of orthogonality for BCGSI+LS-MP stays on the level $\mathcal{O}(u)$ at least until $\kappa(\mathcal{X}) \approx 10^{12}$.

4.3 Conclusion and discussion

Orthogonalization processes are the core of Krylov subspace algorithms. For block Krylov subspace methods, block orthogonalization processes must be used to improve the performance. There are several block Gram-Schmidt variants, and each of these algorithms has different properties in terms of loss of orthogonality, communication, and computation costs. Recent work has focused on developing low-synchronized variants of (block) Gram-Schmidt algorithms, which require only a single global synchronization per (block) column (Yamazaki et al. [2020]). However, as our numerical experiments demonstrate, this reduced synchronization can come at the cost of decreased stability in terms of loss of orthogonality.

We present a new block Gram-Schmidt variant called BCGSI+LS-MP, a variant of BCGSI+LS which uses higher precision in certain parts of the computation. Our numerical experiments demonstrate that this use of mixed precision

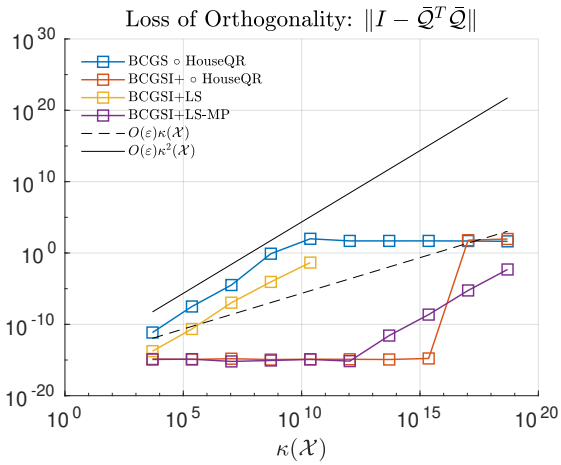

Figure 4.3: The loss of orthogonality of BCGS (blue), BCGSI+ (red), BCGSI+LS (yellow), and BCGSI+LS-MP (purple) versus condition number for Läuchli matrices.

can lead to a more stable algorithm than uniform precision BCGSI+LS while still requiring only a single global synchronization per block. We expect that in the latency-bound regime, where such low-synchronization algorithms are used, the increased bandwidth and computation costs due to the use of higher precision may be negligible.

We note that here we have only provided empirical results; a rigorous proof of the loss of orthogonality in BCGSI+LS-MP remains future work. A first step towards this will be to prove a bound on the loss of orthogonality for uniform precision BCGSI+LS, which is currently missing in the literature. Furthermore, this preliminary study is based on the numerical experiments performed in MATLAB, which cannot give a good indication of resulting performance in large-scale parallel settings. A thorough performance study measuring the overhead of the use of higher precision within BCGSI+LS-MP is needed.

Bibliography

Advanpix LLC. Multiprecision computing toolbox for MATLAB. URL `http://www.advanpix.com/`.

Allison H Baker, John M Dennis, and Elizabeth R Jessup. On improving linear solver performance: A block variant of GMRES. *SIAM Journal on Scientific Computing*, 27(5):1608–1626, 2006.

G. Ballard, E. Carson, J. Demmel, M. Hoemmen, N. Knight, and O. Schwartz. Communication lower bounds and optimal algorithms for numerical linear algebra. *Acta Numerica*, 23:1–155, 2014. doi: 10.1017/S0962492914000038.

Jesse L Barlow and Alicja Smoktunowicz. Reorthogonalized block classical Gram–Schmidt. *Numerische Mathematik*, 123(3):395–423, 2013.

Erin Carson, Kathryn Lund, and Miroslav Rozložník. The stability of block

variants of classical Gram–Schmidt. *SIAM Journal on Matrix Analysis and Applications*, 42(3):1365–1380, 2021. doi: 10.1137/21M1394424.

Erin Carson, Kathryn Lund, Miroslav Rozložník, and Stephen Thomas. Block Gram-Schmidt algorithms and their stability properties. *Linear Algebra and Its Applications*, 638:150–195, 2022.

A Kiełbasiński. Analiza numeryczna algorytmu ortogonalizacji Grama-Schmidta. *Mathematica Applicanda*, 2(2), 1974.

Peter Läuchli. Jordan-elimination und ausgleichung nach kleinsten quadraten. *Numerische Mathematik*, 3(1):226–240, 1961.

Christopher C. Paige, Miroslav Rozložník, and Zdeněk Strakoš. Modified Gram-Schmidt (MGS), least squares, and backward stability of MGS-GMRES. *SIAM Journal on Matrix Analysis and Applications*, 28(1):264–284, 2006. doi: 10.1137/050630416.

Axel Ruhe. Numerical aspects of Gram-Schmidt orthogonalization of vectors. *Linear algebra and its applications*, 52:591–601, 1983.

Yousef Saad. *Iterative methods for sparse linear systems*. SIAM, 2003.

Alicja Smoktunowicz, Jesse Barlow, and Julien Langou. A note on the error analysis of classical Gram-Schmidt. *Numerische Mathematik*, 105, 07 2006. doi: 10.1007/s00211-006-0042-1.

Katarzyna Świrydowicz, Julien Langou, Shreyas Ananthan, Ulrike Yang, and Stephen Thomas. Low synchronization Gram–Schmidt and generalized minimal residual algorithms. *Numerical Linear Algebra with Applications*, 28(2): e2343, 2021.

Ichitaro Yamazaki, Stephen Thomas, Mark Hoemmen, Erik G Boman, Katarzyna Świrydowicz, and James J Elliott. Low-synchronization orthogonalization schemes for s-step and pipelined Krylov solvers in Trilinos. In *Proceedings of the 2020 SIAM Conference on Parallel Processing for Scientific Computing*, pages 118–128. SIAM, 2020.

5. BCGSI+P variants

In Smoktunowicz et al. [2006], the authors show that the upper triangular matrix R resulting from the QR-decomposition of a tall skinny full-rank matrix $A \in \mathbb{R}^{m\times n}$ with CGS satisfies

$$R_j^T R_j = A_j^T A_j + E_j, \quad \|E_j\|_2 \leq c(m,j)\|A\|_2^2 u \quad \text{with}$$
$$c(m,j) = \begin{cases} m+2 & j=1 \\ 3.5mj^2 - 1.5mj + 16j & j+2,\ldots,n, \end{cases}$$

where A_j, R_j, and E_j are the first j columns of A, R, and the error matrix E_j, respectively. This relation holds only if the diagonals of R are computed using a Cholesky-like formula with the help of the Pythagorean theorem. This method is denoted as CGS-P. Similarly, in Carson et al. [2021], the authors introduced two variants of a block vector analogy of CGS-P: BCGS-PIP (BCGS with Pythagorean Inner Product) and BCGS-PIO (BCGS with Pythagorean Intra-Orthogonalization). These are called BCGS-P variants.

The aim of BCGS-P variants is to reduce the loss of orthogonality BCGS has, which is $\|I - \bar{Q}^T\bar{Q}\|_2 \leq \mathcal{O}(u)\kappa_2^{n-1}(A)$, where $\bar{Q}$ is the computed Q-factor. Using the Pythagorean theorem under the assumption $\mathcal{O}(u)\kappa_2^2(A) < 1$, the authors guarantee $\|I - \bar{Q}^T\bar{Q}\|_2 \leq \mathcal{O}(u)\kappa_2^2(A)$.

In the reorthogonalized BCGS (BCGSI+) approach, the authors in Barlow and Smoktunowicz [2013] perform two BCGS steps consecutively and then combine the resulting R-factors to guarantee $\|I - \bar{Q}^T\bar{Q}\|_2 \leq \mathcal{O}(u)$ as long as $\mathcal{O}(u)\kappa_2(A) < 1$. As in the BCGS approach, the stability of an orthogonalization process can be improved via reorthogonalization. To improve the stability of BCGS-P variants, we introduce two more stable reorthogonalized BCGS-P variants in Section 5.1. In this section, we focus on improving only the BCGS-PIP algorithm and call our new variants BCGS-PIP+ and BCGS-PIPI+.

In Section 5.2, we develop mixed-precision variants of BCGS-PIP+ and BCGS -PIPI+. Although we cannot prove it theoretically, we show via numerical experiments in Section 5.3 that there is often a numerical advantage to using mixed precision. In particular, the mixed precision variants satisfy the loss of orthogonality bound for more ill-conditioned matrices than the corresponding uniform precision variants.

5.1 Reorthogonalized Pythagorean variants of BCGS

Let $\boldsymbol{X} \in \mathbb{R}^{m\times s}$ denote a block vector with block size s as a concatenation of s column vectors. The QR factorization $\boldsymbol{\mathcal{X}} = \boldsymbol{\mathcal{Q}}\mathcal{R}$ of the concatenation of p block vectors

$$\boldsymbol{\mathcal{X}} = \begin{bmatrix} \boldsymbol{X}_1 & \boldsymbol{X}_2 & \cdots \boldsymbol{X}_p \end{bmatrix} \in \mathbb{R}^{m\times ps}$$

can be found using a BGS method, where $\boldsymbol{\mathcal{Q}} \in \mathbb{R}^{m\times ps}$ is an orthonormal basis for the column space of $\boldsymbol{\mathcal{X}}$ and $\mathcal{R} \in \mathbb{R}^{ps\times ps}$ is an upper triangular matrix. The goal

of block methods is to compute QR factors block-wise, i.e., s columns of $\boldsymbol{\mathcal{Q}}$ are computed in each iteration instead of one column.

Throughout this chapter, uppercase Roman letters (R_{ij}, S_{ij}, T_{ij}) denote $s \times s$ block entries of a $ps \times ps$ matrix, which will be denoted by uppercase Roman script ($\mathcal{R}, \mathcal{S}, \mathcal{T}$). A block column of such matrices will be written as

$$\mathcal{R}_{1:k-1,k} = \begin{bmatrix} R_{1,k} \\ R_{2,k} \\ \vdots \\ R_{k-1,k} \end{bmatrix}.$$

For simplicity, we also abbreviate $ks \times ks$ submatrices as $\mathcal{R}_k := \mathcal{R}_{1:k,1:k}$.

Bold uppercase Roman letters ($\boldsymbol{Q}_k$, $\boldsymbol{X}_k$, $\boldsymbol{U}_k$) denote $m \times s$ block vectors, and bold, uppercase Roman script ($\boldsymbol{\mathcal{Q}}, \boldsymbol{\mathcal{X}}, \boldsymbol{\mathcal{U}}$) denotes an indexed concatenation of p such vectors. Similar to above, $m \times ks$ submatrices are abbreviated as

$$\boldsymbol{\mathcal{Q}}_k := \boldsymbol{\mathcal{Q}}_{1:k} = \begin{bmatrix} \boldsymbol{Q}_1 & \boldsymbol{Q}_2 & \cdots & \boldsymbol{Q}_k \end{bmatrix}.$$

BCGS-PIP is a variant of BCGS, where the block diagonal entries of the R factor are computed via the block Pythagorean theorem given in Corollary 1.

Corollary 1 (Carson et al. [2021])**.** *Let $\boldsymbol{U}, \boldsymbol{V}, \boldsymbol{W} \in \mathbb{R}^{m \times s}$ be block vectors such that $\boldsymbol{U} = \boldsymbol{V} + \boldsymbol{W}$ and $\boldsymbol{V} \perp \boldsymbol{W}$, in the sense that the spaces spanned by the columns of each block vector are perpendicular to each other. Suppose the QR factorizations for each block vector are given as*

$$\boldsymbol{U} = \boldsymbol{Q}_{\boldsymbol{U}} R_{\boldsymbol{U}}, \quad \boldsymbol{V} = \boldsymbol{Q}_{\boldsymbol{V}} R_{\boldsymbol{V}}, \quad \text{and } \boldsymbol{W} = \boldsymbol{Q}_{\boldsymbol{W}} R_{\boldsymbol{W}}.$$

Then

$$R_{\boldsymbol{U}}^T R_{\boldsymbol{U}} = R_{\boldsymbol{V}}^T R_{\boldsymbol{V}} + R_{\boldsymbol{W}}^T R_{\boldsymbol{W}}^T.$$

All BGS variants use a non-block orthogonalization scheme within blocks, such as Householder QR, CGS, MGS, or Cholesky QR. Throughout this chapter we will refer to this scheme as an `IntraOrtho`. The BCGS-PIP approach is given in Algorithm 8.

Algorithm 8 $[\boldsymbol{\mathcal{Q}}, \mathcal{R}] = \text{BCGS-PIP}(\boldsymbol{\mathcal{X}}, \texttt{IntraOrtho})$ (Carson et al. [2021])

1: Allocate memory for $\boldsymbol{\mathcal{Q}}$, $\mathcal{R}$
2: $[\boldsymbol{Q}_1, R_{11}] = \texttt{IntraOrtho}\,(\boldsymbol{X}_1)$
3: **for** $k = 2, \ldots, p$ **do**
4: $\quad \begin{bmatrix} \mathcal{R}_{1:k-1,k} \\ P_k \end{bmatrix} = \begin{bmatrix} \boldsymbol{\mathcal{Q}}_{k-1} & \boldsymbol{X}_k \end{bmatrix}^T \boldsymbol{X}_k$
5: $\quad R_{kk} = \texttt{chol}\Big(P_k - \mathcal{R}_{1:k-1,k}^T \mathcal{R}_{1:k-1,k}\Big)$
6: $\quad \boldsymbol{V}_k = \boldsymbol{X}_k - \boldsymbol{\mathcal{Q}}_{k-1} \mathcal{R}_{1:k-1,k}$
7: $\quad \boldsymbol{Q}_k = \boldsymbol{V}_k R_{kk}^{-1}$
8: **return** $\boldsymbol{\mathcal{Q}} = [\boldsymbol{Q}_1, \ldots, \boldsymbol{Q}_p]$, $\mathcal{R} = (R_{ij})$

The stability of the orthogonalization processes can be determined via the Cholesky residual

$$\left\| \boldsymbol{\mathcal{X}}^T \boldsymbol{\mathcal{X}} - \bar{\mathcal{R}}^T \bar{\mathcal{R}} \right\| \tag{5.1}$$

and the loss of orthogonality

$$\left\|I - \bar{\boldsymbol{\mathcal{Q}}}^T \bar{\boldsymbol{\mathcal{Q}}}\right\|. \tag{5.2}$$

Using the block Pythagorean approach, BCGS-PIP keeps (5.1) close to machine precision u. On the other hand, (5.2) can be quite high and there is a constraint on the condition number of $\boldsymbol{\mathcal{X}}$: for $\mathcal{O}(u)\,\kappa^2(\boldsymbol{\mathcal{X}}) \leq 1$, BCGS-PIP can only guarantee

$$\left\|I - \bar{\boldsymbol{\mathcal{Q}}}^T \bar{\boldsymbol{\mathcal{Q}}}\right\| \leq \mathcal{O}(u)\,\kappa^2(\boldsymbol{\mathcal{X}}).$$

This bound shows that when $\kappa(\boldsymbol{\mathcal{X}})$ is close to $u^{-1/2}$, BCGS-PIP may completely lose all orthogonality. To prevent the loss of orthogonality of the algorithm, we can use reorthogonalization.

Reorthogonalization can be implemented in several ways. The easiest one is to run BCGS-PIP twice in a row. Using this technique, we introduce Algorithm 9, which we refer to as BCGS-PIP+.

Algorithm 9 $[\boldsymbol{\mathcal{Q}}, \mathcal{R}] = \text{BCGS-PIP+}(\boldsymbol{\mathcal{X}}, \texttt{IntraOrtho})$

1: $[\boldsymbol{\mathcal{U}}, \mathcal{S}] = \texttt{BCGS-PIP}(\boldsymbol{\mathcal{X}}, \texttt{IntraOrtho})$
2: $[\boldsymbol{\mathcal{Q}}, \mathcal{T}] = \texttt{BCGS-PIP}(\boldsymbol{\mathcal{U}}, \texttt{IntraOrtho})$
3: $\mathcal{R} = \mathcal{T}\mathcal{S}$;
4: **return** $\boldsymbol{\mathcal{Q}} = [\boldsymbol{\mathcal{Q}}_1, \ldots, \boldsymbol{\mathcal{Q}}_p]$, $\mathcal{R} = (R_{ij})$

Theorem 2 and Corollary 3 gives the loss of orthogonality and Cholesky residual bound for BSCG-PIP+, respectively.

Theorem 2. *Let $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{m\times ps}$ with $\mathcal{O}(u)\,\kappa^2(\boldsymbol{\mathcal{X}}) \leq \frac{1}{2}$ and $\boldsymbol{\mathcal{X}} = \bar{\boldsymbol{\mathcal{Q}}}\bar{\mathcal{R}}$, where $\bar{\boldsymbol{\mathcal{Q}}}$ and $\bar{\mathcal{R}}$ are obtained via Algorithm 9. Assuming that for all $\boldsymbol{X} \in \mathbb{R}^{m\times s}$, $[\bar{\boldsymbol{Q}}, \bar{R}] =$ $\texttt{IntraOrtho}(\boldsymbol{X})$ satisfy*

$$\begin{aligned} \bar{R}^T\bar{R} &= \boldsymbol{X}^T\boldsymbol{X} + \Delta E, \quad \|\Delta E\| \leq \mathcal{O}(u)\,\|\boldsymbol{X}\|^2, \text{ and} \\ \bar{\boldsymbol{Q}}\bar{R} &= \boldsymbol{X} + \Delta\boldsymbol{D}, \quad \|\Delta\boldsymbol{D}\| \leq \mathcal{O}(u)\left(\|\boldsymbol{X}\| + \left\|\bar{\boldsymbol{Q}}\right\|\left\|\bar{R}\right\|\right), \end{aligned}$$

then $\bar{\boldsymbol{\mathcal{Q}}}$ and $\bar{\mathcal{R}}$ satisfy

$$\left\|I - \bar{\boldsymbol{\mathcal{Q}}}^T \bar{\boldsymbol{\mathcal{Q}}}\right\| \leq \mathcal{O}(u), \text{ and} \tag{5.3}$$

$$\bar{\boldsymbol{\mathcal{Q}}}\bar{\mathcal{R}} = \boldsymbol{\mathcal{X}} + \Delta\boldsymbol{\mathcal{D}}, \quad \|\Delta\boldsymbol{\mathcal{D}}\| \leq \mathcal{O}(u)\,\|\boldsymbol{\mathcal{X}}\|. \tag{5.4}$$

Proof. We first apply [Carson et al., 2021, Theorem 3.4] to line 1 and get

$$\left\|I - \bar{\boldsymbol{\mathcal{U}}}^T\bar{\boldsymbol{\mathcal{U}}}\right\| \leq \mathcal{O}(u)\,\kappa^2(\boldsymbol{\mathcal{X}}), \text{ and} \tag{5.5}$$

$$\bar{\boldsymbol{\mathcal{U}}}\bar{\mathcal{S}} = \boldsymbol{\mathcal{X}} + \Delta\boldsymbol{\mathcal{D}}_1, \quad \|\Delta\boldsymbol{\mathcal{D}}_1\| \leq \mathcal{O}(u)\,\|\boldsymbol{\mathcal{X}}\|. \tag{5.6}$$

Applying it for the second time to line 2 gives

$$\left\|I - \bar{\boldsymbol{\mathcal{Q}}}^T \bar{\boldsymbol{\mathcal{Q}}}\right\| \leq \mathcal{O}(u)\,\kappa^2(\boldsymbol{\mathcal{U}}), \text{ and} \tag{5.7}$$

$$\bar{\boldsymbol{\mathcal{Q}}}\bar{\mathcal{T}} = \bar{\boldsymbol{\mathcal{U}}} + \Delta\boldsymbol{\mathcal{D}}_2, \quad \|\Delta\boldsymbol{\mathcal{D}}_2\| \leq \mathcal{O}(u)\left\|\bar{\boldsymbol{\mathcal{U}}}\right\|. \tag{5.8}$$

Using [Carson et al., 2021, Theorem 3.1], we can also derive $\left\|\bar{\boldsymbol{\mathcal{U}}}\right\| \leq 3$. Using the perturbation theory of singular values [Golub and Van Loan, 2013, Corollary 8.6.2], (5.5) and the assumption $\mathcal{O}(u)\,\kappa^2(\boldsymbol{\mathcal{X}}) \leq \frac{1}{2}$ we can calculate a lower bound on $\sigma_{\min}(\bar{\boldsymbol{\mathcal{U}}})$:

$$\sigma_{\min}(\bar{\boldsymbol{\mathcal{U}}}) \geq \frac{1}{\sqrt{2}},$$

which yields

$$\kappa(\bar{\boldsymbol{\mathcal{U}}}) = \frac{\sigma_{\max}}{\sigma_{\min}} \leq 3\sqrt{2}. \tag{5.9}$$

Applying (5.9) to (5.7) automatically gives (5.3).

To prove (5.4), we first use [Carson et al., 2021, Theorem 3.1] to get $\left\|\bar{\boldsymbol{\mathcal{Q}}}\right\| \leq 3$. Then using this bound we write

$$\bar{\boldsymbol{\mathcal{Q}}}\bar{\mathcal{R}} = \bar{\boldsymbol{\mathcal{Q}}}\bar{\mathcal{T}}\bar{\mathcal{S}} + \Delta\boldsymbol{\mathcal{D}}_3, \quad \|\Delta\boldsymbol{\mathcal{D}}_3\| \leq \mathcal{O}(u)\left\|\bar{\mathcal{T}}\right\|\left\|\bar{\mathcal{S}}\right\|. \tag{5.10}$$

Plugging (5.6) and (5.8) in (5.10) yields

$$\bar{\boldsymbol{\mathcal{Q}}}\bar{\mathcal{R}} = (\bar{\boldsymbol{\mathcal{U}}} + \Delta\boldsymbol{\mathcal{D}}_2)\bar{\mathcal{S}} + \Delta\boldsymbol{\mathcal{D}}_3 = \boldsymbol{\mathcal{X}} + \underbrace{\Delta\boldsymbol{\mathcal{D}}_1 + \Delta\boldsymbol{\mathcal{D}}_2\bar{\mathcal{S}} + \Delta\boldsymbol{\mathcal{D}}_3}_{=:\Delta\boldsymbol{\mathcal{D}}},$$

where

$$\begin{aligned}\|\Delta\boldsymbol{\mathcal{D}}\| &\leq \|\Delta\boldsymbol{\mathcal{D}}_1\| + \|\Delta\boldsymbol{\mathcal{D}}_2\|\left\|\bar{\mathcal{S}}\right\| + \|\Delta\boldsymbol{\mathcal{D}}_3\| \\ &\leq \mathcal{O}(u)\,\|\boldsymbol{\mathcal{X}}\| + 3\mathcal{O}(u)\left\|\bar{\mathcal{S}}\right\| + \mathcal{O}(u)\left\|\bar{\mathcal{T}}\right\|\left\|\bar{\mathcal{S}}\right\|\end{aligned} \tag{5.11}$$

when we use $\left\|\bar{\boldsymbol{\mathcal{U}}}\right\| \leq 3$.

Lastly, we apply [Carson et al., 2021, Theorem 3.2] again with $\|\boldsymbol{\mathcal{U}}\| = \mathcal{O}(1)$ to get

$$\begin{aligned}\bar{\mathcal{S}}^T\bar{\mathcal{S}} &= \boldsymbol{\mathcal{X}}^T\boldsymbol{\mathcal{X}} + \Delta\mathcal{E}_1, \quad \|\Delta\mathcal{E}_1\| \leq \mathcal{O}(u)\,\|\boldsymbol{\mathcal{X}}\|^2, \text{ and} \\ \bar{\mathcal{T}}^T\bar{\mathcal{T}} &= \bar{\boldsymbol{\mathcal{U}}}^T\bar{\boldsymbol{\mathcal{U}}} + \Delta\mathcal{E}_2, \quad \|\Delta\mathcal{E}_2\| \leq \mathcal{O}(u),\end{aligned}$$

which with another application of [Golub and Van Loan, 2013, Corollary 8.6.2] gives

$$\left\|\bar{\mathcal{S}}\right\| \leq \mathcal{O}(1)\,\|\boldsymbol{\mathcal{X}}\| \text{ and } \left\|\bar{\mathcal{T}}\right\| \leq \mathcal{O}(1). \tag{5.12}$$

Substituting (5.12) into (5.11) proves (5.4) and concludes the proof. □

Corollary 3. *Let $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{m\times ps}$ with $\mathcal{O}(u)\,\kappa^2(\boldsymbol{\mathcal{X}}) \leq \frac{1}{2}$ and $\boldsymbol{\mathcal{X}} = \bar{\boldsymbol{\mathcal{Q}}}\bar{\mathcal{R}}$, where $\bar{\boldsymbol{\mathcal{Q}}}$ and $\bar{\mathcal{R}}$ are obtained via Algorithm 9. Assuming that for all $\boldsymbol{X} \in \mathbb{R}^{m\times s}$, $[\bar{\boldsymbol{Q}}, \bar{R}] =$ `IntraOrtho`$(\boldsymbol{X})$ satisfy*

$$\begin{aligned}\bar{R}^T\bar{R} &= \boldsymbol{X}^T\boldsymbol{X} + \Delta E, \quad \|\Delta E\| \leq \mathcal{O}(u)\,\|\boldsymbol{X}\|^2, \text{ and} \\ \bar{\boldsymbol{Q}}\bar{R} &= \boldsymbol{X} + \Delta\boldsymbol{D}, \quad \|\Delta\boldsymbol{D}\| \leq \mathcal{O}(u)\left(\|\boldsymbol{X}\| + \left\|\bar{\boldsymbol{Q}}\right\|\left\|\bar{R}\right\|\right),\end{aligned}$$

then $\bar{\mathcal{R}}$ satisfies

$$\bar{\mathcal{R}}^T\bar{\mathcal{R}} = \boldsymbol{\mathcal{X}}^T\boldsymbol{\mathcal{X}} + \Delta\mathcal{R}, \quad \|\Delta\mathcal{R}\| \leq \mathcal{O}(u)\,\|\boldsymbol{\mathcal{X}}\|^2. \tag{5.13}$$

We can prove Corollary 3 directly via applying [Carson et al., 2021, Theorem 3.2] twice.

One drawback to Algorithm 9 is the use of two for-loops (one in each BCGS-PIP implementation) which results in additional synchronization points. As mentioned in Chapter 4, each synchronization point can cause an increase in the communication cost of the overall algorithm. To reduce this cost, we can combine the for-loops without an additional synchronization point. We denote this new variant given in Algorithm 10 as BCGS-PIPI+.

Algorithm 10 $[\boldsymbol{\mathcal{Q}}, \mathcal{R}] = \text{BCGS-PIPI+}(\boldsymbol{\mathcal{X}}, \texttt{IntraOrtho})$

1: Allocate memory for $\boldsymbol{\mathcal{Q}}$, $\mathcal{R}$
2: $[\boldsymbol{Q}_1, R_{11}] = \texttt{IntraOrtho}\,(\boldsymbol{X}_1)$ ▷ $S_{11} = R_{11}$, $\boldsymbol{U}_1 = \boldsymbol{Q}_1$, $T_{11} = I$
3: **for** $k = 2, \ldots, p$ **do**
4: $\quad \begin{bmatrix} \mathcal{S}_{1:k-1,k} \\ \Omega_k \end{bmatrix} = \begin{bmatrix} \boldsymbol{\mathcal{Q}}_{k-1} & \boldsymbol{X}_k \end{bmatrix}^T \boldsymbol{X}_k$ ▷ First BCGS-PIP step
5: $\quad S_{kk} = \texttt{chol}\Big(\Omega_k - \mathcal{S}_{1:k-1,k}^T \mathcal{S}_{1:k-1,k}\Big)$
6: $\quad \boldsymbol{V}_k = \boldsymbol{X}_k - \boldsymbol{\mathcal{Q}}_{k-1} \mathcal{S}_{1:k-1,k}$
7: $\quad \boldsymbol{U}_k = \boldsymbol{V}_k S_{kk}^{-1}$
8: $\quad \begin{bmatrix} \mathcal{T}_{1:k-1,k} \\ P_k \end{bmatrix} = \begin{bmatrix} \boldsymbol{\mathcal{Q}}_{k-1} & \boldsymbol{U}_k \end{bmatrix}^T \boldsymbol{U}_k$ ▷ Second BCGS-PIP step
9: $\quad T_{kk} = \texttt{chol}\Big(P_k - \mathcal{T}_{1:k-1,k}^T \mathcal{T}_{1:k-1,k}\Big)$
10: $\quad \boldsymbol{W}_k = \boldsymbol{U}_k - \boldsymbol{\mathcal{Q}}_{k-1} \mathcal{T}_{1:k-1,k}$
11: $\quad \boldsymbol{Q}_k = \boldsymbol{W}_k T_{kk}^{-1}$
12: $\quad \mathcal{R}_{1:k-1,k} = \mathcal{S}_{1:k-1,k} + \mathcal{T}_{1:k-1,k} S_{kk}$ ▷ Finalize $\mathcal{R}$ entries
13: $\quad R_{kk} = T_{kk} S_{kk}$
14: **return** $\boldsymbol{\mathcal{Q}} = [\boldsymbol{Q}_1, \ldots, \boldsymbol{Q}_p]$, $\mathcal{R} = (R_{ij})$

While we reduce the communication cost via combining for-loops, we cannot use the results from Carson et al. [2021] immediately anymore to prove the stability of BCGS-PIPI+. We provide Lemmas 4 and 5 to prepare what we need to prove stability.

Lemma 4. *Suppose* $\mathcal{O}(u)\,\kappa^2(\boldsymbol{\mathcal{X}}) \leq \frac{1}{2}$ *and* $\bar{\boldsymbol{\mathcal{Q}}} \in \mathbb{R}^{m \times ps}$ *is such that* $\left\| I - \bar{\boldsymbol{\mathcal{Q}}}^T \bar{\boldsymbol{\mathcal{Q}}} \right\| \leq \frac{\mathcal{O}(u)}{1 - \mathcal{O}(u)\kappa^2(\boldsymbol{\mathcal{X}})}$. *Then*

$$\left\| \bar{\boldsymbol{\mathcal{Q}}} \right\| \leq 1 + \mathcal{O}(u).$$

Proof. The triangle inequality gives

$$\left\| \bar{\boldsymbol{\mathcal{Q}}} \right\|^2 \leq \frac{1 - \mathcal{O}(u)\,\kappa^2(\boldsymbol{\mathcal{X}}) + \mathcal{O}(u)}{1 - \mathcal{O}(u)\,\kappa^2(\boldsymbol{\mathcal{X}})}.$$

Using the assumption on $\mathcal{O}(u)\,\kappa^2(\boldsymbol{\mathcal{X}})$, we have

$$\left\| \bar{\boldsymbol{\mathcal{Q}}} \right\|^2 \leq 2\left(\frac{1}{2} + \mathcal{O}(u)\right) \leq 1 + \mathcal{O}(u).$$

Taking the square root and approximating it by a first-order Taylor expansion around 1 achieves the desired result. □

Lemma 5. *Assume* $\mathcal{O}(u)\,\kappa^2(\boldsymbol{\mathcal{X}}) \le \frac{1}{2}$. *Fix* $k \in \{2,\dots,p\}$ *and suppose* $\bar{\boldsymbol{\mathcal{Q}}}_{k-1} \in \mathbb{R}^{m\times ps}$ *computed by Algorithm 10 satisfies the following:*

$$\boldsymbol{\mathcal{X}}_{k-1} + \Delta\boldsymbol{\mathcal{X}}_{k-1} = \bar{\boldsymbol{\mathcal{Q}}}_{k-1}\bar{\mathcal{R}}_{k-1}, \quad \|\Delta\boldsymbol{\mathcal{X}}_{k-1}\| \le \mathcal{O}(u)\,\|\boldsymbol{\mathcal{X}}_{k-1}\|, \text{ and} \tag{5.14}$$

$$\left\|I - \bar{\boldsymbol{\mathcal{Q}}}_{k-1}^T\bar{\boldsymbol{\mathcal{Q}}}_{k-1}\right\| \le \frac{\mathcal{O}(u)}{1-\mathcal{O}(u)\,\kappa^2(\boldsymbol{\mathcal{X}}_{k-1})}. \tag{5.15}$$

Then for the computed quantities $\bar{\mathcal{S}}_{1:k-1,k}$, $\bar{S}_{kk}$, $\bar{\boldsymbol{V}}_k$, *and* $\bar{\boldsymbol{U}}_k$ *from Algorithm 10, it holds that*

$$\bar{S}_{kk}^T\bar{S}_{kk} = \boldsymbol{X}_k^T\boldsymbol{X}_k - \boldsymbol{X}_k^T\bar{\boldsymbol{\mathcal{Q}}}_{k-1}\bar{\boldsymbol{\mathcal{Q}}}_{k-1}^T\boldsymbol{X}_k + \Delta S_{kk}, \text{ and} \tag{5.16}$$

$$\bar{\boldsymbol{U}}_k\bar{S}_{kk} = \boldsymbol{X}_k - \bar{\boldsymbol{\mathcal{Q}}}_{k-1}\bar{\boldsymbol{\mathcal{Q}}}_{k-1}^T\boldsymbol{X}_k + \Delta\boldsymbol{U}_k, \tag{5.17}$$

where

$$\|\Delta S_{kk}\| \le \mathcal{O}(u)\,\|\boldsymbol{X}_k\|^2, \tag{5.18}$$

$$\left\|\bar{S}_{kk}\right\| \le \mathcal{O}(1)\,\|\boldsymbol{X}_k\|, \tag{5.19}$$

$$\left\|\bar{S}_{kk}^{-1}\right\|^2 \le \frac{1}{\sigma_{\min}^2(\boldsymbol{\mathcal{X}}_k) - \mathcal{O}(u)\,\|\boldsymbol{\mathcal{X}}_k\|^2} \tag{5.20}$$

$$\|\Delta\boldsymbol{U}_k\| \le \mathcal{O}(u)\,\|\boldsymbol{X}_k\|\left(1+\left\|\bar{\boldsymbol{U}}_k\right\|\right), \text{ and} \tag{5.21}$$

$$\left\|\bar{\boldsymbol{U}}_k\right\| \le 1.7. \tag{5.22}$$

Proof. Line 4 of Algorithm 10 with Lemma 4 gives

$$\bar{\mathcal{S}}_{1:k-1,k} = \bar{\boldsymbol{\mathcal{Q}}}_{k-1}^T\boldsymbol{X}_k + \Delta\boldsymbol{S}_k, \quad \|\Delta\boldsymbol{S}_k\| \le \mathcal{O}(u)\,\|\boldsymbol{X}_k\|; \tag{5.23}$$

$$\bar{\Omega}_k = \boldsymbol{X}_k^T\boldsymbol{X}_k + \Delta\Omega_k, \quad \|\Delta\Omega_k\| \le \mathcal{O}(u)\,\|\boldsymbol{X}_k\|^2; \tag{5.24}$$

$$\left\|\bar{\mathcal{S}}_{1:k-1,k}\right\| \le (1+\mathcal{O}(u))\,\|\boldsymbol{X}_k\|; \text{ and} \tag{5.25}$$

$$\left\|\bar{\Omega}_k\right\| \le (1+\mathcal{O}(u))\,\|\boldsymbol{X}_k\|^2. \tag{5.26}$$

Applying [Higham, 2002, Theorem 10.3] to $\bar{S}_{kk}$ yields

$$\bar{S}_{kk}^T\bar{S}_{kk} = \bar{\Omega}_k - \bar{\mathcal{S}}_{1:k-1,k}^T\bar{\mathcal{S}}_{1:k-1,k} + \Delta F_k + \Delta C_k \tag{5.27}$$

with

$$\|\Delta F_k\| \le \mathcal{O}(u)\,\|\boldsymbol{X}_k\|^2 \text{ and } \|\Delta C_k\| \le \mathcal{O}(u)\,\|\boldsymbol{X}_k\|^2, \tag{5.28}$$

where ΔF_k denotes the floating-point error from the sum and product of $\bar{\Omega}_k - \bar{\mathcal{S}}_{1:k-1,k}^T\bar{\mathcal{S}}_{1:k-1,k}$, while ΔC_k is the Cholesky error.

Substituting (5.23)-(5.25) into (5.27) gives (5.16) with

$$\Delta S_{kk} := \Delta\Omega_k + \Delta F_k + \Delta C_k - \boldsymbol{X}_k^T\bar{\boldsymbol{\mathcal{Q}}}_{k-1}\Delta\boldsymbol{S}_k - \Delta\boldsymbol{S}_k^T\bar{\boldsymbol{\mathcal{Q}}}_{k-1}^T\boldsymbol{X}_k,$$

after dropping $\mathcal{O}(u^2)$ terms. From Lemma 4 and bounds (5.23), (5.24), and (5.28), we can derive (5.18) and (5.19).

For the bound (5.20), we rewrite (5.16) as

$$\bar{S}_{kk}^T\bar{S}_{kk} = \boldsymbol{X}_k^T(I - \bar{\boldsymbol{\mathcal{Q}}}_{k-1}\bar{\boldsymbol{\mathcal{Q}}}_{k-1}^T)(I - \bar{\boldsymbol{\mathcal{Q}}}_{k-1}\bar{\boldsymbol{\mathcal{Q}}}_{k-1}^T)\boldsymbol{X}_k + \Delta\widetilde{S}_k, \tag{5.29}$$

where $\Delta\widetilde{S}_k := \boldsymbol{X}_k^T \bar{\boldsymbol{\mathcal{Q}}}_{k-1} \bar{\boldsymbol{\mathcal{Q}}}_{k-1}^T (I - \bar{\boldsymbol{\mathcal{Q}}}_{k-1} \bar{\boldsymbol{\mathcal{Q}}}_{k-1}^T) \boldsymbol{X}_k + \Delta S_{kk}$ satisfies

$$\left\|\Delta\widetilde{S}_k\right\| \leq \|\boldsymbol{X}_k\|^2 \left\|\bar{\boldsymbol{\mathcal{Q}}}_{k-1}\right\|^2 \left\|I - \bar{\boldsymbol{\mathcal{Q}}}_{k-1} \bar{\boldsymbol{\mathcal{Q}}}_{k-1}^T\right\| + \mathcal{O}(u) \|\boldsymbol{X}_k\|^2 \leq \mathcal{O}(u) \|\boldsymbol{X}_k\|^2 \quad (5.30)$$

via Lemma 4 and the assumption (5.15).

Using (5.29) and (5.30) we can write

$$\begin{aligned} \sigma_{\min}^2(\bar{S}_{kk}) &\geq \sigma_{\min}^2((I - \bar{\boldsymbol{\mathcal{Q}}}_{k-1} \bar{\boldsymbol{\mathcal{Q}}}_{k-1}^T) \boldsymbol{X}_k) - \left\|\Delta\widetilde{S}_k\right\| \\ &\geq \sigma_{\min}^2((I - \bar{\boldsymbol{\mathcal{Q}}}_{k-1} \bar{\boldsymbol{\mathcal{Q}}}_{k-1}^T) \boldsymbol{X}_k) - \mathcal{O}(u) \|\boldsymbol{X}_k\|^2 . \end{aligned} \quad (5.31)$$

To bound $\sigma_{\min}^2((I - \bar{\boldsymbol{\mathcal{Q}}}_{k-1} \bar{\boldsymbol{\mathcal{Q}}}_{k-1}^T) \boldsymbol{X}_k)$, we first use the assumption (5.14) to obtain

$$\begin{aligned} (I - \bar{\boldsymbol{\mathcal{Q}}}_{k-1} \bar{\boldsymbol{\mathcal{Q}}}_{k-1}^T) \boldsymbol{X}_k &= \boldsymbol{X}_k - \bar{\boldsymbol{\mathcal{Q}}}_{k-1} \bar{\mathcal{R}}_{k-1} \bar{\mathcal{R}}_{k-1}^{-1} \bar{\boldsymbol{\mathcal{Q}}}_{k-1}^T \boldsymbol{X}_k \\ &= \begin{bmatrix} \boldsymbol{\mathcal{X}}_{k-1} + \Delta\boldsymbol{\mathcal{X}}_{k-1} & \boldsymbol{X}_k \end{bmatrix} \boldsymbol{Y}^T, \end{aligned} \quad (5.32)$$

where $\boldsymbol{Y} := \begin{bmatrix} -(\bar{\mathcal{R}}_{k-1}^{-1} \bar{\boldsymbol{\mathcal{Q}}}_{k-1}^T \boldsymbol{X}_k)^T & I_s \end{bmatrix} \in \mathbb{R}^{sk \times s}$, and I_s is the $s \times s$ identity matrix. Using (5.32), (5.14), and perturbation theory of singular values [Golub and Van Loan, 2013, Corollary 8.6.2], we can write

$$\begin{aligned} \sigma_{\min}^2((I - \bar{\boldsymbol{\mathcal{Q}}}_{k-1} \bar{\boldsymbol{\mathcal{Q}}}_{k-1}^T) \boldsymbol{X}_k) &= \min_{\boldsymbol{y} \in \mathbb{R}^{sk} \setminus \boldsymbol{0}} \left(\frac{\left\|(I - \bar{\boldsymbol{\mathcal{Q}}}_{k-1} \bar{\boldsymbol{\mathcal{Q}}}_{k-1}^T) \boldsymbol{X}_k \boldsymbol{y}\right\|^2}{\|\boldsymbol{y}\|^2} \right) \\ &= \min_{\boldsymbol{y} \in \mathbb{R}^{sk} \setminus \boldsymbol{0}} \left(\frac{\left\|\begin{bmatrix} \boldsymbol{\mathcal{X}}_{k-1} + \Delta\boldsymbol{\mathcal{X}}_{k-1} & \boldsymbol{X}_k \end{bmatrix} \boldsymbol{Y}^T \boldsymbol{y}\right\|^2}{\|\boldsymbol{y}\|^2} \right) \\ &= \min_{\boldsymbol{y} \in \mathbb{R}^{sk} \setminus \boldsymbol{0}} \left(\frac{\left\|\begin{bmatrix} \boldsymbol{\mathcal{X}}_{k-1} + \Delta\boldsymbol{\mathcal{X}}_{k-1} & \boldsymbol{X}_k \end{bmatrix} \boldsymbol{Y}^T \boldsymbol{y}\right\|^2}{\|\boldsymbol{Y}^T \boldsymbol{y}\|^2} \frac{\left\|\boldsymbol{Y}^T \boldsymbol{y}\right\|^2}{\|\boldsymbol{y}\|^2} \right). \end{aligned} \quad (5.33)$$

We can bound (5.33) from below by

$$\min_{\boldsymbol{y} \in \mathbb{R}^{sk} \setminus \boldsymbol{0}} \left(\frac{\left\|\begin{bmatrix} \boldsymbol{\mathcal{X}}_{k-1} + \Delta\boldsymbol{\mathcal{X}}_{k-1} & \boldsymbol{X}_k \end{bmatrix} (\boldsymbol{Y}^T \boldsymbol{y})\right\|^2}{\|\boldsymbol{Y}^T \boldsymbol{y}\|^2} \right) \min_{\boldsymbol{y} \in \mathbb{R}^{sk} \setminus \boldsymbol{0}} \left(\frac{\left\|\boldsymbol{Y}^T \boldsymbol{y}\right\|^2}{\|\boldsymbol{y}\|^2} \right),$$

while

$$\begin{aligned} \sigma_{\min}^2 \left(\begin{bmatrix} \boldsymbol{\mathcal{X}}_{k-1} + \Delta\boldsymbol{\mathcal{X}}_{k-1} & \boldsymbol{X}_k \end{bmatrix} \right) &\geq (\sigma_{\min}(\boldsymbol{\mathcal{X}}_k) - \|\Delta\boldsymbol{\mathcal{X}}_{k-1}\|)^2 \\ &\geq \sigma_{\min}^2(\boldsymbol{\mathcal{X}}_k) - 2 \|\boldsymbol{\mathcal{X}}_k\| \|\Delta\boldsymbol{\mathcal{X}}_{k-1}\| \\ &\geq \sigma_{\min}^2(\boldsymbol{\mathcal{X}}_k) - \mathcal{O}(u) \|\boldsymbol{\mathcal{X}}_k\| \|\boldsymbol{\mathcal{X}}_{k-1}\| \\ &\geq \sigma_{\min}^2(\boldsymbol{\mathcal{X}}_k) - \mathcal{O}(u) \|\boldsymbol{\mathcal{X}}_k\|^2 . \end{aligned} \quad (5.34)$$

Finally, using (5.34) and (5.31) with $\left\|\bar{S}_{kk}^{-1}\right\| = 1/\sigma_{\min}(\bar{S}_{kk})$ we get (5.20).

To prove (5.21) we first use line 6 of Algorithm 10. Substitution of (5.23) gives

$$\bar{\boldsymbol{V}}_k = \boldsymbol{X}_k - \bar{\boldsymbol{\mathcal{Q}}}_{k-1} \bar{\boldsymbol{\mathcal{Q}}}_{k-1}^T \boldsymbol{X}_k + \Delta\boldsymbol{V}_k, \quad \|\Delta\boldsymbol{V}_k\| \leq \mathcal{O}(u) \|\boldsymbol{X}_k\| . \quad (5.35)$$

Using similar logic as in pp. 1376 of Carson et al. [2021] (using [Higham, 2002, Theorem 8.5 & Lemma 6.6]) and applying (5.19) we get

$$\bar{\boldsymbol{U}}_k \bar{\mathcal{S}}_{kk} = \bar{\boldsymbol{V}}_k + \Delta \boldsymbol{G}_k, \quad \|\Delta \boldsymbol{G}_k\| \leq \mathcal{O}(u) \left\|\bar{\boldsymbol{U}}_k\right\| \|\boldsymbol{X}_k\|. \tag{5.36}$$

Substituting (5.35) into (5.36) gives (5.17), with

$$\Delta \boldsymbol{U}_k := \Delta \boldsymbol{G}_k + \Delta \boldsymbol{V}_k,$$

which satisfies (5.21).

For (5.22) we first multiply $\bar{\boldsymbol{U}}_k \bar{S}_{kk}$ with its transpose and obtain

$$\begin{aligned}(\bar{\boldsymbol{U}}_k \bar{S}_{kk})^T (\bar{\boldsymbol{U}}_k \bar{S}_{kk}) = &\boldsymbol{X}_k^T \boldsymbol{X}_k - 2\boldsymbol{X}_k^T \bar{\boldsymbol{Q}}_{k-1} \bar{\boldsymbol{Q}}_{k-1}^T \boldsymbol{X}_k \\ &+ \boldsymbol{X}_k^T \bar{\boldsymbol{Q}}_{k-1} \bar{\boldsymbol{Q}}_{k-1}^T \bar{\boldsymbol{Q}}_{k-1} \bar{\boldsymbol{Q}}_{k-1}^T \boldsymbol{X}_k + \Delta H_k \\ = &\boldsymbol{X}_k^T \boldsymbol{X}_k - \boldsymbol{X}_k^T \bar{\boldsymbol{Q}}_{k-1} \bar{\boldsymbol{Q}}_{k-1}^T \boldsymbol{X}_k \\ &+ \boldsymbol{X}_k^T \bar{\boldsymbol{Q}}_{k-1} (I - \bar{\boldsymbol{Q}}_{k-1}^T \bar{\boldsymbol{Q}}_{k-1}) \bar{\boldsymbol{Q}}_{k-1}^T \boldsymbol{X}_k + \Delta H_k,\end{aligned} \tag{5.37}$$

where

$$\Delta H_k := \boldsymbol{X}_k^T \Delta \boldsymbol{U}_k + \left(\Delta \boldsymbol{U}_k\right)^T \boldsymbol{X}_k + \boldsymbol{X}_k^T \bar{\boldsymbol{Q}}_{k-1} \bar{\boldsymbol{Q}}_{k-1}^T \Delta \boldsymbol{U}_k + \left(\Delta \boldsymbol{U}_k\right)^T \bar{\boldsymbol{Q}}_{k-1} \bar{\boldsymbol{Q}}_{k-1}^T \boldsymbol{X}_k,$$

after dropping $\mathcal{O}(u^2)$ terms. Applying Lemma 4 and (5.21) leads to

$$\|\Delta H_k\| \leq \mathcal{O}(u) \|\boldsymbol{X}_k\|^2 \left(1 + \left\|\bar{\boldsymbol{U}}_k\right\|\right).$$

Substituting (5.16) into (5.37) gives

$$(\bar{\boldsymbol{U}}_k \bar{S}_{kk})^T (\bar{\boldsymbol{U}}_k \bar{S}_{kk}) = \bar{S}_{kk}^T \bar{S}_{kk} + \boldsymbol{X}_k^T \bar{\boldsymbol{Q}}_{k-1} (I - \bar{\boldsymbol{Q}}_{k-1}^T \bar{\boldsymbol{Q}}_{k-1}) \bar{\boldsymbol{Q}}_{k-1}^T \boldsymbol{X}_k + \Delta H_k - \Delta S_{kk},$$

and then multiplying by S_{kk}^{-T} on the left and S_{kk}^{-1} on the right, leads to

$$\bar{\boldsymbol{U}}_k^T \bar{\boldsymbol{U}}_k = I + \bar{S}_{kk}^{-T} \boldsymbol{X}_k^T \bar{\boldsymbol{Q}}_{k-1} (I - \bar{\boldsymbol{Q}}_{k-1}^T \bar{\boldsymbol{Q}}_{k-1}) \bar{\boldsymbol{Q}}_{k-1}^T \boldsymbol{X}_k \bar{S}_{kk}^{-1} + \bar{S}_{kk}^{-T} (\Delta H_k - \Delta S_{kk}) \bar{S}_{kk}^{-1}. \tag{5.38}$$

Using the assumption

$$\mathcal{O}(u) \kappa^2(\boldsymbol{\mathcal{X}}_k) \leq \mathcal{O}(u) \kappa^2(\boldsymbol{\mathcal{X}}) \leq 1/2,$$

applying (5.15) together with (5.20), and taking norms gives us

$$\begin{aligned}\left\|\bar{\boldsymbol{U}}_k\right\|^2 = \left\|\bar{\boldsymbol{U}}_k^T \bar{\boldsymbol{U}}_k\right\| &\leq 1 + \mathcal{O}(u) \left\|\bar{S}_{kk}^{-1}\right\|^2 \|\boldsymbol{X}_k\|^2 + \mathcal{O}(u) \left\|\bar{S}_{kk}^{-1}\right\|^2 \|\boldsymbol{X}_k\|^2 \left(1 + \left\|\bar{\boldsymbol{U}}_k\right\|\right) \\ &\leq 1 + \mathcal{O}(u) \kappa^2(\boldsymbol{\mathcal{X}}_k) \left(2 + \left\|\bar{\boldsymbol{U}}_k\right\|\right) \\ &\leq 1 + \frac{1}{2} \left\|\bar{\boldsymbol{U}}_k\right\|.\end{aligned} \tag{5.39}$$

Finally, we solve the quadratic inequality (5.39) to get $\left\|\bar{\boldsymbol{U}}_k\right\| \leq \frac{1+\sqrt{33}}{4} \leq 1.7$. □

Using the proven intermediate terms in Lemma 5, we prove a bound for the loss of orthogonality of Algorithm 10 in Theorem 6. Notice that, one of the most crucial differences between BCGS-PIP+ and BCGS-PIPI+ is the loss of orthogonality condition on the `IntraOrtho` in BCGS-PIPI+. This condition limits the choice of `IntraOrtho` in BCGS-PIPI+ (Householder QR (Higham [2002]), TSQR (Mori et al. [2012]), or Cholesky QR2 (Yamamoto et al. [2015])) while no such condition exists in BCGS-PIP+.

Theorem 6. *Assume that* $\mathcal{O}(u)\,\kappa^2(\boldsymbol{\mathcal{X}}) \leq \frac{1}{2}$ *and that for all* $\boldsymbol{X} \in \mathbb{R}^{m\times s}$ *with* $\mathcal{O}(u)\,\kappa^2(\boldsymbol{X}) \leq \frac{1}{2}$, $[\bar{\boldsymbol{Q}}, \bar{\mathcal{R}}] = \texttt{IntraOrtho}(\boldsymbol{X})$ *satisfy*

$$\begin{aligned} \bar{R}^T\bar{R} &= \boldsymbol{X}^T\boldsymbol{X} + \Delta E, \quad \|\Delta E\| \leq \mathcal{O}(u)\,\|\boldsymbol{X}\|^2 \\ \bar{\boldsymbol{Q}}\bar{R} &= \boldsymbol{X} + \Delta\boldsymbol{D}, \quad \|\Delta\boldsymbol{D}\| \leq \mathcal{O}(u)\,\|\boldsymbol{X}\| \ \text{ and} \\ \left\|I - \bar{\boldsymbol{Q}}^T\bar{\boldsymbol{Q}}\right\| &\leq \frac{\mathcal{O}(u)}{1-\mathcal{O}(u)\,\kappa^2(\boldsymbol{X})}. \end{aligned}$$

Assume also that for $k \in \{2,\ldots,p\}$

$$\boldsymbol{\mathcal{X}}_{k-1} + \Delta\boldsymbol{\mathcal{X}}_{k-1} = \bar{\boldsymbol{\mathcal{Q}}}_{k-1}\bar{\mathcal{R}}_{k-1}, \quad \|\Delta\boldsymbol{\mathcal{X}}_{k-1}\| \leq \mathcal{O}(u)\,\|\boldsymbol{\mathcal{X}}_{k-1}\|.$$

Then for all $k \in \{1,\ldots,p\}$, $\bar{\boldsymbol{\mathcal{Q}}}_k$ *computed by Algorithm 10 satisfies*

$$\left\|I - \bar{\boldsymbol{\mathcal{Q}}}_k^T\bar{\boldsymbol{\mathcal{Q}}}_k\right\| \leq \frac{\mathcal{O}(u)}{1-\mathcal{O}(u)\,\kappa^2(\boldsymbol{\mathcal{X}}_k)} \leq \mathcal{O}(u). \tag{5.40}$$

Proof. $\mathcal{O}(u)\,\kappa^2(\boldsymbol{\mathcal{X}}) \leq \frac{1}{2}$ implies that for all $k \in \{1,\ldots,p\}$, $\mathcal{O}(u)\,\kappa^2(\boldsymbol{\mathcal{X}}_k) \leq \frac{1}{2}$. Then for the base case, the assumptions on `IntraOrtho` directly yield

$$\left\|I - \bar{\boldsymbol{\mathcal{Q}}}_1^T\bar{\boldsymbol{\mathcal{Q}}}_1\right\| = \left\|I - \bar{\boldsymbol{Q}}_1^T\bar{\boldsymbol{Q}}_1\right\| \leq \frac{\mathcal{O}(u)}{1-\mathcal{O}(u)\,\kappa^2(\boldsymbol{X}_1)} = \frac{\mathcal{O}(u)}{1-\mathcal{O}(u)\,\kappa^2(\boldsymbol{\mathcal{X}}_1)}.$$

Suppose that (5.40) holds for all $j \in \{1,\ldots,k-1\}$. We will look at $I - \bar{\boldsymbol{\mathcal{Q}}}_k^T\bar{\boldsymbol{\mathcal{Q}}}_k$ block-by-block:

$$I - \bar{\boldsymbol{\mathcal{Q}}}_k^T\bar{\boldsymbol{\mathcal{Q}}}_k = \begin{bmatrix} I - \bar{\boldsymbol{\mathcal{Q}}}_{k-1}^T\bar{\boldsymbol{\mathcal{Q}}}_{k-1} & \bar{\boldsymbol{\mathcal{Q}}}_{k-1}^T\bar{\boldsymbol{Q}}_k \\ \bar{\boldsymbol{Q}}_k^T\bar{\boldsymbol{\mathcal{Q}}}_{k-1} & I - \bar{\boldsymbol{Q}}_k^T\bar{\boldsymbol{Q}}_k \end{bmatrix}, \tag{5.41}$$

where $\bar{\boldsymbol{\mathcal{Q}}}_k = \begin{bmatrix}\bar{\boldsymbol{\mathcal{Q}}}_{k-1} & \bar{\boldsymbol{Q}}_k\end{bmatrix}$.

We can directly prove the upper left block via the induction hypothesis. For the off-diagonal blocks, using standard floating-point bounds, Lemmas 4 and 5 give

$$\bar{\mathcal{T}}_{1:k-1,k} = \bar{\boldsymbol{\mathcal{Q}}}_{k-1}^T\bar{\boldsymbol{U}}_k + \Delta\boldsymbol{T}_k, \|\Delta\boldsymbol{T}_k\| \leq \mathcal{O}(u) \tag{5.42}$$

$$\bar{P}_k = \bar{\boldsymbol{U}}_k^T\bar{\boldsymbol{U}}_k + \Delta P_k, \|\Delta P_k\| \leq \mathcal{O}(u) \tag{5.43}$$

$$\left\|\bar{\mathcal{T}}_{1:k-1,k}\right\| \leq \mathcal{O}(1) \tag{5.44}$$

$$\left\|\bar{P}_k\right\| \leq \mathcal{O}(1).$$

As in Lemma 5, using [Higham, 2002, Theorem 10.3] on $\bar{T}_{kk}$ we get

$$\bar{T}_{kk}^T\bar{T}_{kk} = \bar{P}_k - \bar{\mathcal{T}}_{1:k-1,k}^T\bar{\mathcal{T}}_{1:k-1,k} + \Delta F_k + \Delta C_k \tag{5.45}$$

with

$$\|\Delta F_k\| \leq \mathcal{O}(u) \text{ and } \|\Delta C_k\| \leq \mathcal{O}(u),$$

and

$$\left\|\bar{T}_{kk}\right\| \leq \mathcal{O}(1), \tag{5.46}$$

where ΔF_k denotes the floating-point error from the sum and product of $\bar{P}_k - \bar{\mathcal{T}}_{1:k-1,k}^T \bar{\mathcal{T}}_{1:k-1,k}$, and ΔC_k is the Cholesky error.

Then, substituting (5.42) and (5.43) into (5.45) gives

$$\bar{T}_{kk}^T \bar{T}_{kk} = \bar{\boldsymbol{U}}_k^T (I - \bar{\boldsymbol{\mathcal{Q}}}_{k-1} \bar{\boldsymbol{\mathcal{Q}}}_{k-1}^T)(I - \bar{\boldsymbol{\mathcal{Q}}}_{k-1} \bar{\boldsymbol{\mathcal{Q}}}_{k-1}^T) \bar{\boldsymbol{U}}_k + \Delta \widetilde{T}_k \quad \left\|\Delta \widetilde{T}_k\right\| \leq \mathcal{O}(u). \tag{5.47}$$

To bound $\left\|\bar{T}_{kk}^{-1}\right\|$ we use (5.47) and thus have

$$\sigma_{\min}^2(\bar{T}_{kk}) \geq \sigma_{\min}^2\Big((I - \bar{\boldsymbol{\mathcal{Q}}}_{k-1} \bar{\boldsymbol{\mathcal{Q}}}_{k-1}^T) \bar{\boldsymbol{U}}_k\Big) - \left\|\Delta \widetilde{T}_k\right\|. \tag{5.48}$$

Together with (5.35) and (5.36), we can write

$$\bar{\boldsymbol{U}}_k = \Big((I - \bar{\boldsymbol{\mathcal{Q}}}_{k-1} \bar{\boldsymbol{\mathcal{Q}}}_{k-1}^T) \boldsymbol{X}_k + \Delta \widetilde{\boldsymbol{U}}_k\Big) \bar{S}_{kk}^{-1}. \tag{5.49}$$

Multiplying (5.49) by $I - \bar{\boldsymbol{\mathcal{Q}}}_{k-1} \bar{\boldsymbol{\mathcal{Q}}}_{k-1}^T$ on the left leads to

$$\begin{aligned}(I - \bar{\boldsymbol{\mathcal{Q}}}_{k-1} \bar{\boldsymbol{\mathcal{Q}}}_{k-1}^T) \bar{\boldsymbol{U}}_k = \bar{\boldsymbol{U}}_k &- \bar{\boldsymbol{\mathcal{Q}}}_{k-1} (I - \bar{\boldsymbol{\mathcal{Q}}}_{k-1}^T \bar{\boldsymbol{\mathcal{Q}}}_{k-1}) \bar{\boldsymbol{\mathcal{Q}}}_{k-1}^T \boldsymbol{X}_k \bar{S}_{kk}^{-1} \\ &- \bar{\boldsymbol{\mathcal{Q}}}_{k-1} \bar{\boldsymbol{\mathcal{Q}}}_{k-1}^T \Delta \widetilde{\boldsymbol{U}}_k \bar{S}_{kk}^{-1}.\end{aligned} \tag{5.50}$$

Define $\Delta \boldsymbol{E}_k := \bar{\boldsymbol{\mathcal{Q}}}_{k-1} (I - \bar{\boldsymbol{\mathcal{Q}}}_{k-1}^T \bar{\boldsymbol{\mathcal{Q}}}_{k-1}) \bar{\boldsymbol{\mathcal{Q}}}_{k-1}^T \boldsymbol{X}_k \bar{S}_{kk}^{-1} + \bar{\boldsymbol{\mathcal{Q}}}_{k-1} \bar{\boldsymbol{\mathcal{Q}}}_{k-1}^T \Delta \widetilde{\boldsymbol{U}}_k \bar{S}_{kk}^{-1}$. Using (5.20) and the assumption $\mathcal{O}(u) \kappa^2(\boldsymbol{\mathcal{X}}) \leq \frac{1}{2}$, we can write

$$\|\boldsymbol{X}_k\| \left\|\bar{S}_{kk}^{-1}\right\| \leq \sqrt{\|\boldsymbol{\mathcal{X}}_k\|^2 \left\|\bar{S}_{kk}^{-1}\right\|^2} \leq \sqrt{\frac{\kappa^2(\boldsymbol{\mathcal{X}}_k)}{1 - \mathcal{O}(u) \kappa^2(\boldsymbol{\mathcal{X}}_k)}} \leq \sqrt{2} \kappa(\boldsymbol{\mathcal{X}}_k).$$

Using the induction hypothesis and Lemma 4, we can get

$$\|\Delta \boldsymbol{E}_k\| \leq \mathcal{O}(u) \|\boldsymbol{X}_k\| \left\|\bar{S}_{kk}^{-1}\right\| \leq \mathcal{O}(u) \kappa(\boldsymbol{\mathcal{X}}_k). \tag{5.51}$$

Using (5.38), (5.39), (5.50), and (5.51), we write the following bounds:

$$\begin{aligned}\sigma_{\min}^2(\bar{\boldsymbol{U}}_k) &\geq 1 - \left\|\bar{S}_{kk}^{-T} \boldsymbol{X}_k^T \bar{\boldsymbol{\mathcal{Q}}}_{k-1} (I - \bar{\boldsymbol{\mathcal{Q}}}_{k-1}^T \bar{\boldsymbol{\mathcal{Q}}}_{k-1}) \bar{\boldsymbol{\mathcal{Q}}}_{k-1}^T \boldsymbol{X}_k \bar{S}_{kk}^{-1}\right\| \\ &\quad - \left\|\bar{S}_{kk}^{-T} (\Delta H_k - \Delta S_{kk}) \bar{S}_{kk}^{-1}\right\| \\ &\geq 1 - \mathcal{O}(u) \kappa^2(\boldsymbol{\mathcal{X}}_k),\end{aligned}$$

and

$$\begin{aligned}\sigma_{\min}^2((I - \bar{\boldsymbol{\mathcal{Q}}}_{k-1} \bar{\boldsymbol{\mathcal{Q}}}_{k-1}^T) \bar{\boldsymbol{U}}_k) &\geq \Big(\sigma_{\min}(\bar{\boldsymbol{U}}_k) - \|\Delta \boldsymbol{E}_k\|\Big)^2 \\ &\geq \sigma_{\min}^2(\bar{\boldsymbol{U}}_k) - 2 \|\Delta \boldsymbol{E}_k\| \left\|\bar{\boldsymbol{U}}_k\right\| \\ &\geq 1 - \mathcal{O}(u) \kappa^2(\boldsymbol{\mathcal{X}}_k).\end{aligned} \tag{5.52}$$

Substituting (5.52) and (5.47) into (5.48), and using $\left\|\bar{T}_{kk}^{-1}\right\|^2 = 1/\sigma_{\min}^2(\bar{T}_{kk})$, we have

$$\left\|\bar{T}_{kk}^{-1}\right\| = \sqrt{\left\|\bar{T}_{kk}^{-1}\right\|^2} \leq \sqrt{\frac{1}{1-\mathcal{O}(u)\,\kappa^2(\boldsymbol{\mathcal{X}}_k)}} \leq \frac{1}{1-\mathcal{O}(u)\,\kappa^2(\boldsymbol{\mathcal{X}}_k)}. \tag{5.53}$$

As in the derivation of (5.36), line 11 together with Lemma 4 and (5.46) leads to

$$\bar{\boldsymbol{Q}}_k \bar{T}_{kk} = \bar{\boldsymbol{W}}_k + \Delta \boldsymbol{G}_k, \quad \|\Delta \boldsymbol{G}_k\| \leq \mathcal{O}(u) \left\|\bar{\boldsymbol{Q}}_k\right\| \left\|\bar{T}_{kk}\right\| \leq \mathcal{O}(u), \tag{5.54}$$

after dropping $\mathcal{O}(u^2)$ terms. Multiplying (5.54) by $\bar{\boldsymbol{\mathcal{Q}}}_{k-1}^T$ on the left leads to

$$\bar{\boldsymbol{\mathcal{Q}}}_{k-1}^T \bar{\boldsymbol{Q}}_k \bar{T}_{kk} = \bar{\boldsymbol{\mathcal{Q}}}_{k-1}^T \bar{\boldsymbol{W}}_k + \bar{\boldsymbol{\mathcal{Q}}}_{k-1}^T \Delta \boldsymbol{G}_k. \tag{5.55}$$

Next, substituting

$$\bar{\boldsymbol{W}}_k = \bar{\boldsymbol{U}}_k - \bar{\boldsymbol{\mathcal{Q}}}_{k-1} \bar{\boldsymbol{\mathcal{T}}}_{1:k-1,k} + \Delta \boldsymbol{W}_k, \|\Delta \boldsymbol{W}_k\| \leq \mathcal{O}(u) \text{ and } \left\|\bar{\boldsymbol{W}}_k\right\| \leq \mathcal{O}(1) \tag{5.56}$$

into (5.55), multiplying both sides by $\bar{T}_{kk}^{-1}$ along with (5.42) yields

$$\bar{\boldsymbol{\mathcal{Q}}}_{k-1}^T \bar{\boldsymbol{Q}}_k = (I - \bar{\boldsymbol{\mathcal{Q}}}_{k-1}^T \bar{\boldsymbol{\mathcal{Q}}}_{k-1}) \mathcal{T}_{1:k-1,k} \bar{T}_{kk}^{-1} + \left(\Delta \boldsymbol{T}_k + \bar{\boldsymbol{\mathcal{Q}}}_{k-1}^T (\Delta \boldsymbol{G}_k + \Delta \boldsymbol{W}_k)\right) \bar{T}_{kk}^{-1}.$$

Finally, the induction hypothesis (5.40), Lemmas 4 and 5, and (5.42), (5.53), (5.54), and (5.56) gives the bound

$$\begin{aligned} \left\|\bar{\boldsymbol{\mathcal{Q}}}_{k-1}^T \bar{\boldsymbol{Q}}_k\right\| &\leq \frac{\mathcal{O}(u)}{1-\mathcal{O}(u)\,\kappa^2(\boldsymbol{\mathcal{X}}_{k-1})} \cdot \frac{1}{1-\mathcal{O}(u)\,\kappa^2(\boldsymbol{\mathcal{X}}_k)} + \frac{\mathcal{O}(u)}{1-\mathcal{O}(u)\,\kappa^2(\boldsymbol{\mathcal{X}}_k)} \\ &\leq \frac{\mathcal{O}(u)}{1-\mathcal{O}(u)\,\kappa^2(\boldsymbol{\mathcal{X}}_k)}. \end{aligned} \tag{5.57}$$

For the bottom right entry, we can write

$$\begin{aligned} \bar{T}_{kk}^T (I - \bar{\boldsymbol{Q}}_k^T \bar{\boldsymbol{Q}}_k) \bar{T}_{kk} &= \bar{T}_{kk}^T \bar{T}_{kk} - \bar{T}_{kk}^T \bar{\boldsymbol{Q}}_k^T \bar{\boldsymbol{Q}}_k \bar{T}_{kk} \\ &= \bar{P}_k - \bar{\boldsymbol{\mathcal{T}}}_{1:k-1,k}^T \bar{\boldsymbol{\mathcal{T}}}_{1:k-1,k} + \Delta F_k + \Delta C_k \\ &\qquad - (\bar{\boldsymbol{W}}_k - \Delta \boldsymbol{G}_k)^T (\bar{\boldsymbol{W}}_k - \Delta \boldsymbol{G}_k) \\ &= \bar{P}_k - \bar{\boldsymbol{\mathcal{T}}}_{1:k-1,k}^T \bar{\boldsymbol{\mathcal{T}}}_{1:k-1,k} - \bar{\boldsymbol{W}}_k^T \bar{\boldsymbol{W}}_k \\ &\qquad - \underbrace{\bar{\boldsymbol{W}}_k^T \Delta \boldsymbol{G}_k - (\Delta \boldsymbol{G}_k)^T \bar{\boldsymbol{W}}_k + \Delta F_k + \Delta C_k}_{=:\Delta H_k}, \end{aligned} \tag{5.58}$$

where we eliminated $\mathcal{O}(u^2)$ terms and combined (5.45) and (5.54). Then substituting (5.43) and (5.56) into (5.58) gives

$$\begin{aligned} \bar{T}_{kk}^T (I - \bar{\boldsymbol{Q}}_k^T \bar{\boldsymbol{Q}}_k) \bar{T}_{kk} = &-\bar{\boldsymbol{\mathcal{T}}}_{1:k-1,k}^T \bar{\boldsymbol{\mathcal{T}}}_{1:k-1,k} + \bar{\boldsymbol{U}}_k^T \bar{\boldsymbol{\mathcal{Q}}}_{k-1} \bar{\boldsymbol{\mathcal{T}}}_{1:k-1,k} + \bar{\boldsymbol{\mathcal{T}}}_{1:k-1,k}^T \bar{\boldsymbol{\mathcal{Q}}}_{k-1}^T \bar{\boldsymbol{U}}_k \\ &- \bar{\boldsymbol{\mathcal{T}}}_{1:k-1,k}^T \bar{\boldsymbol{\mathcal{Q}}}_{k-1}^T \bar{\boldsymbol{\mathcal{Q}}}_{k-1} \bar{\boldsymbol{\mathcal{T}}}_{1:k-1,k} - \Delta J_k - \Delta H_k, \end{aligned} \tag{5.59}$$

where $\Delta J_k := \left(\bar{\boldsymbol{U}}_k - \bar{\boldsymbol{\mathcal{Q}}}_{k-1}\bar{\mathcal{T}}_{1:k-1,k}\right)^T \Delta \boldsymbol{W}_k - \left(\Delta \boldsymbol{W}_k\right)^T \left(\bar{\boldsymbol{U}}_k - \bar{\boldsymbol{\mathcal{Q}}}_{k-1}\bar{\mathcal{T}}_{1:k-1,k}\right)$. Substituting (5.42) into (5.59) leads to

$$\begin{aligned}\bar{T}_{kk}^T(I - \bar{\boldsymbol{Q}}_k^T\bar{\boldsymbol{Q}}_k)\bar{T}_{kk} = \bar{\mathcal{T}}_{1:k-1,k}^T\left(I - \bar{\boldsymbol{\mathcal{Q}}}_{k-1}^T\bar{\boldsymbol{\mathcal{Q}}}_{k-1}\right)\bar{\mathcal{T}}_{1:k-1,k} \\ - \left(\Delta \boldsymbol{T}_k\right)^T\bar{\mathcal{T}}_{1:k-1,k} - \bar{\mathcal{T}}_{1:k-1,k}^T\Delta \boldsymbol{T}_k - \Delta J_k - \Delta H_k.\end{aligned}$$

Next, we multiply (5.60) by $\bar{T}_{kk}^{-T}$ on the left and $\bar{T}_{kk}^{-1}$ on the right, take norms, and get

$$\left\|I - \bar{\boldsymbol{Q}}_k^T\bar{\boldsymbol{Q}}_k\right\| \leq \frac{\mathcal{O}(u)}{1 - \mathcal{O}(u)\,\kappa^2(\boldsymbol{\mathcal{X}}_k)} \tag{5.60}$$

after the application of previously derived bounds and the induction hypothesis.

Lastly, using (5.41) together with [Garcia and Horn, 2017, P.15.50], the induction hypothesis and bounds (5.57) and (5.60), we get

$$\begin{aligned}\left\|I - \bar{\boldsymbol{\mathcal{Q}}}_k^T\bar{\boldsymbol{\mathcal{Q}}}_k\right\| &= \left\|\begin{bmatrix} I - \bar{\boldsymbol{\mathcal{Q}}}_{k-1}^T\bar{\boldsymbol{\mathcal{Q}}}_{k-1} & \bar{\boldsymbol{\mathcal{Q}}}_{k-1}^T\bar{\boldsymbol{Q}}_k \\ \bar{\boldsymbol{Q}}_k^T\bar{\boldsymbol{\mathcal{Q}}}_{k-1} & I - \bar{\boldsymbol{Q}}_k^T\bar{\boldsymbol{Q}}_k\end{bmatrix}\right\| \\ &\leq \left\|\begin{bmatrix}\left\|I - \bar{\boldsymbol{\mathcal{Q}}}_{k-1}^T\bar{\boldsymbol{\mathcal{Q}}}_{k-1}\right\| & \left\|\bar{\boldsymbol{\mathcal{Q}}}_{k-1}^T\bar{\boldsymbol{Q}}_k\right\| \\ \left\|\bar{\boldsymbol{Q}}_k^T\bar{\boldsymbol{\mathcal{Q}}}_{k-1}\right\| & \left\|I - \bar{\boldsymbol{Q}}_k^T\bar{\boldsymbol{Q}}_k\right\|\end{bmatrix}\right\| \\ &\leq \left\|\begin{bmatrix}\left\|I - \bar{\boldsymbol{\mathcal{Q}}}_{k-1}^T\bar{\boldsymbol{\mathcal{Q}}}_{k-1}\right\| & \left\|\bar{\boldsymbol{\mathcal{Q}}}_{k-1}^T\bar{\boldsymbol{Q}}_k\right\| \\ \left\|\bar{\boldsymbol{Q}}_k^T\bar{\boldsymbol{\mathcal{Q}}}_{k-1}\right\| & \left\|I - \bar{\boldsymbol{Q}}_k^T\bar{\boldsymbol{Q}}_k\right\|\end{bmatrix}\right\|_{\mathrm{F}} \\ &\leq \left\|I - \bar{\boldsymbol{\mathcal{Q}}}_{k-1}^T\bar{\boldsymbol{\mathcal{Q}}}_{k-1}\right\| + 2\left\|\bar{\boldsymbol{\mathcal{Q}}}_{k-1}^T\bar{\boldsymbol{Q}}_k\right\| + \left\|I - \bar{\boldsymbol{Q}}_k^T\bar{\boldsymbol{Q}}_k\right\| \\ &\leq \frac{\mathcal{O}(u)}{1 - \mathcal{O}(u)\,\kappa^2(\boldsymbol{\mathcal{X}}_k)}.\end{aligned} \tag{5.61}$$

With the assumption $\mathcal{O}(u)\,\kappa(\boldsymbol{\mathcal{X}}) \leq \frac{1}{2}$ we conclude the proof. □

Next, we prove the standard residual for Algorithm 10 in Theorem 7.

Theorem 7. *Assume that $\mathcal{O}(u)\,\kappa^2(\boldsymbol{\mathcal{X}}) \leq \frac{1}{2}$ and that for all $\boldsymbol{X} \in \mathbb{R}^{m\times s}$ with $\mathcal{O}(u)\,\kappa^2(\boldsymbol{X}) \leq \frac{1}{2}$, $[\bar{\boldsymbol{Q}}, \bar{\mathcal{R}}] = \texttt{IntraOrtho}(\boldsymbol{X})$ satisfy*

$$\bar{\boldsymbol{Q}}\bar{R} = \boldsymbol{X} + \Delta \boldsymbol{D}, \quad \|\Delta \boldsymbol{D}\| \leq \mathcal{O}(u)\,\|\boldsymbol{X}\| \;\; \textit{and} \;\; \left\|I - \bar{\boldsymbol{Q}}^T\bar{\boldsymbol{Q}}\right\| \leq \frac{\mathcal{O}(u)}{1 - \mathcal{O}(u)\,\kappa^2(\boldsymbol{X})}.$$

Then for all $k \in \{1, \ldots, p\}$,

$$\boldsymbol{\mathcal{X}}_k + \Delta \boldsymbol{\mathcal{X}}_k = \bar{\boldsymbol{\mathcal{Q}}}_k\bar{\mathcal{R}}_k, \quad \|\Delta \boldsymbol{\mathcal{X}}_k\| \leq \mathcal{O}(u)\,\|\boldsymbol{\mathcal{X}}_k\|.$$

Proof. Using the assumption on `IntraOrtho`, we have for the base case,

$$\boldsymbol{\mathcal{X}}_1 + \Delta \boldsymbol{\mathcal{X}}_1 = \bar{\boldsymbol{\mathcal{Q}}}_1\bar{\mathcal{R}}_1, \quad \|\Delta \boldsymbol{\mathcal{X}}_1\| \leq \mathcal{O}(u)\,\|\boldsymbol{\mathcal{X}}_1\|.$$

Now, assume that for all $j \in \{1, \ldots, k-1\}$, we have

$$\boldsymbol{\mathcal{X}}_{k-1} + \Delta \boldsymbol{\mathcal{X}}_{k-1} = \bar{\boldsymbol{\mathcal{Q}}}_{k-1}\bar{\mathcal{R}}_{k-1}, \quad \|\Delta \boldsymbol{\mathcal{X}}_{k-1}\| \leq \mathcal{O}(u)\,\|\boldsymbol{\mathcal{X}}_{k-1}\|. \tag{5.62}$$

Then we can write

$$\begin{aligned}\Delta\boldsymbol{\mathcal{X}}_k &:= \begin{bmatrix}\Delta\boldsymbol{\mathcal{X}}_{k-1} & \Delta\boldsymbol{X}_k\end{bmatrix} \\ &= \bar{\boldsymbol{\mathcal{Q}}}_k\bar{\mathcal{R}}_k - \boldsymbol{\mathcal{X}}_k = \begin{bmatrix}\bar{\boldsymbol{\mathcal{Q}}}_{k-1}\bar{\mathcal{R}}_{k-1} - \boldsymbol{\mathcal{X}}_{k-1} & \bar{\boldsymbol{\mathcal{Q}}}_{k-1}\bar{\mathcal{R}}_{1:k-1,k} + \bar{\boldsymbol{Q}}_k\bar{R}_{kk} - \boldsymbol{X}_k\end{bmatrix}.\end{aligned} \tag{5.63}$$

We can directly prove the first element of (5.63) using the induction hypothesis. For the second element, we will work on $\bar{\boldsymbol{\mathcal{Q}}}_{k-1}\bar{\mathcal{R}}_{1:k-1,k}$ and $\bar{\boldsymbol{Q}}_k\bar{R}_{kk}$ separately. Using (5.25), (5.44), and Lemma 5 we can write

$$\bar{\mathcal{R}}_{1:k-1,k} = \bar{\mathcal{S}}_{1:k-1,k} + \bar{\mathcal{T}}_{1:k-1,k}\bar{S}_{kk} + \Delta\boldsymbol{R}_k,$$

$$\|\Delta\boldsymbol{R}_k\| \leq \mathcal{O}(u)\left(\left\|\bar{\mathcal{S}}_{1:k-1,k}\right\| + \left\|\bar{\mathcal{T}}_{1:k-1,k}\right\|\left\|\bar{S}_{kk}\right\|\right) \leq \mathcal{O}(u)\|\boldsymbol{X}_k\|.$$

Combining with (5.23), we have

$$\begin{aligned}\bar{\boldsymbol{\mathcal{Q}}}_{k-1}\bar{\mathcal{R}}_{1:k-1,k} &= \bar{\boldsymbol{\mathcal{Q}}}_{k-1}\left(\bar{\mathcal{S}}_{1:k-1,k} + \bar{\mathcal{T}}_{1:k-1,k}\bar{S}_{kk} + \Delta\boldsymbol{R}_k\right) \\ &= \bar{\boldsymbol{\mathcal{Q}}}_{k-1}\left(\bar{\boldsymbol{\mathcal{Q}}}_{k-1}^T\boldsymbol{X}_k + \Delta\boldsymbol{S}_k + \bar{\mathcal{T}}_{1:k-1,k}\bar{S}_{kk} + \Delta\boldsymbol{R}_k\right) \\ &= \bar{\boldsymbol{\mathcal{Q}}}_{k-1}\bar{\boldsymbol{\mathcal{Q}}}_{k-1}^T\boldsymbol{X}_k + \bar{\boldsymbol{\mathcal{Q}}}_{k-1}\bar{\mathcal{T}}_{1:k-1,k}\bar{S}_{kk} + \bar{\boldsymbol{\mathcal{Q}}}_{k-1}(\Delta\boldsymbol{S}_k + \Delta\boldsymbol{R}_k),\end{aligned} \tag{5.64}$$

with

$$\left\|\bar{\boldsymbol{\mathcal{Q}}}_{k-1}(\Delta\boldsymbol{S}_k + \Delta\boldsymbol{R}_k)\right\| \leq \mathcal{O}(u)\|\boldsymbol{X}_k\|.$$

Using line 13 of Algorithm 10 along with the bounds (5.19), and (5.46) gives

$$\bar{R}_{kk} = \bar{T}_{kk}\bar{S}_{kk} + \Delta R_{kk}, \quad \|\Delta R_{kk}\| \leq \mathcal{O}(u)\|\boldsymbol{X}_k\|. \tag{5.65}$$

Substituting (5.35), (5.36), (5.54), and (5.56) into (5.65), we can write

$$\begin{aligned}\bar{\boldsymbol{Q}}_k\bar{R}_{kk} &= \bar{\boldsymbol{Q}}_k\left(\bar{T}_{kk}\bar{S}_{kk} + \Delta R_{kk}\right) \\ &= \bar{\boldsymbol{Q}}_k\bar{T}_{kk}\bar{S}_{kk} + \bar{\boldsymbol{Q}}_k\Delta R_{kk} \\ &= \left(\bar{\boldsymbol{W}}_k + \Delta\boldsymbol{G}_k\right)\bar{S}_{kk} + \bar{\boldsymbol{Q}}_k\Delta R_{kk} \\ &= \left(\bar{\boldsymbol{U}}_k - \bar{\boldsymbol{\mathcal{Q}}}_{k-1}\bar{\mathcal{T}}_{1:k-1,k} + \Delta\boldsymbol{W}_k + \Delta\boldsymbol{G}_k\right)\bar{S}_{kk} + \bar{\boldsymbol{Q}}_k\Delta R_{kk} \\ &= \bar{\boldsymbol{U}}_k\bar{S}_{kk} - \bar{\boldsymbol{\mathcal{Q}}}_{k-1}\bar{\mathcal{T}}_{1:k-1,k}\bar{S}_{kk} + (\Delta\boldsymbol{W}_k + \Delta\boldsymbol{G}_k)\bar{S}_{kk} + \bar{\boldsymbol{Q}}_k\Delta R_{kk} \\ &= \boldsymbol{X}_k - \bar{\boldsymbol{\mathcal{Q}}}_{k-1}\bar{\boldsymbol{\mathcal{Q}}}_{k-1}^T\boldsymbol{X}_k + \Delta\boldsymbol{V}_k + \Delta\boldsymbol{G}_k - \bar{\boldsymbol{\mathcal{Q}}}_{k-1}\bar{\mathcal{T}}_{1:k-1,k}\bar{S}_{kk} \\ &\quad + (\Delta\boldsymbol{W}_k + \Delta\boldsymbol{G}_k)\bar{S}_{kk} + \bar{\boldsymbol{Q}}_k\Delta R_{kk},\end{aligned} \tag{5.66}$$

with

$$\left\|\Delta\boldsymbol{V}_k + \Delta\boldsymbol{G}_k + (\Delta\boldsymbol{W}_k + \Delta\boldsymbol{G}_k)\bar{S}_{kk} + \bar{\boldsymbol{Q}}_k\Delta R_{kk}\right\| \leq \mathcal{O}(u)\|\boldsymbol{X}_k\|.$$

Using (5.64), (5.66), and the induction hypothesis concludes the proof. □

The Cholesky residual bound is given in Corollary 8 whose proof follows directly from Theorems 6 and 7.

Corollary 8. *Assume that* $\mathcal{O}(u)\,\kappa^2(\boldsymbol{\mathcal{X}}) \leq \frac{1}{2}$ *and for all* $k \in \{1,\ldots,p\}$, $\bar{\boldsymbol{\mathcal{Q}}}_k$ *computed by Algorithm 10 satisfies*

$$\left\|I - \bar{\boldsymbol{\mathcal{Q}}}_k^T \bar{\boldsymbol{\mathcal{Q}}}_k\right\| \leq \frac{\mathcal{O}(u)}{1 - \mathcal{O}(u)\,\kappa^2(\boldsymbol{\mathcal{X}}_k)} \text{ and} \tag{5.67}$$

$$\boldsymbol{\mathcal{X}}_k + \Delta\boldsymbol{\mathcal{X}}_k = \bar{\boldsymbol{\mathcal{Q}}}_k \bar{\mathcal{R}}_k, \quad \|\Delta\boldsymbol{\mathcal{X}}_k\| \leq \mathcal{O}(u)\,\|\boldsymbol{\mathcal{X}}_k\|. \tag{5.68}$$

Then for all $k \in \{1,\ldots,p\}$,

$$\bar{\mathcal{R}}_k^T \bar{\mathcal{R}}_k = \boldsymbol{\mathcal{X}}_k^T \boldsymbol{\mathcal{X}}_k + \Delta\mathcal{R}_k, \quad \|\Delta\mathcal{R}_k\| \leq \mathcal{O}(u)\,\|\boldsymbol{\mathcal{X}}_k\|^2.$$

Proof. Using (5.68) gives

$$\bar{\mathcal{R}}_k^T \bar{\boldsymbol{\mathcal{Q}}}_k^T \bar{\boldsymbol{\mathcal{Q}}}_k \bar{\mathcal{R}}_k = \boldsymbol{\mathcal{X}}_k^T \boldsymbol{\mathcal{X}}_k + \Delta\mathcal{E}_k, \quad \|\Delta\mathcal{E}_k\| \leq \mathcal{O}(u)\,\|\boldsymbol{\mathcal{X}}_k\|^2,$$

where $\Delta\mathcal{E}_k = \boldsymbol{\mathcal{X}}_k^T \Delta\boldsymbol{\mathcal{X}}_k + (\Delta\boldsymbol{\mathcal{X}}_k)^T \boldsymbol{\mathcal{X}}_k + (\Delta\boldsymbol{\mathcal{X}}_k)^2$. Rearranging terms along with the assumption (5.67) leads to

$$\bar{\mathcal{R}}_k^T \bar{\mathcal{R}}_k = \boldsymbol{\mathcal{X}}_k^T \boldsymbol{\mathcal{X}}_k + \underbrace{\Delta\mathcal{E}_k + \bar{\mathcal{R}}_k^T (I - \bar{\boldsymbol{\mathcal{Q}}}_k^T \bar{\boldsymbol{\mathcal{Q}}}_k)\bar{\mathcal{R}}_k}_{=:\Delta\mathcal{R}_k}, \; \|\Delta\mathcal{R}_k\| \leq \mathcal{O}(u)\left(\|\boldsymbol{\mathcal{X}}_k\|^2 + \left\|\bar{\mathcal{R}}_k\right\|^2\right). \tag{5.69}$$

We then multiply (5.68) by $\bar{\boldsymbol{\mathcal{Q}}}_k^T$ on the left and rearrange terms to get

$$\bar{\mathcal{R}}_k = (I - \bar{\boldsymbol{\mathcal{Q}}}_k^T \bar{\boldsymbol{\mathcal{Q}}}_k)\bar{\mathcal{R}}_k + \bar{\boldsymbol{\mathcal{Q}}}_k^T \boldsymbol{\mathcal{X}}_k + \bar{\boldsymbol{\mathcal{Q}}}_k^T \Delta\boldsymbol{\mathcal{X}}_k.$$

Lastly, with the assumptions of this lemma together with Lemma 4, we can write the bound

$$\left\|\bar{\mathcal{R}}_k\right\| \leq \frac{1 + \mathcal{O}(u)}{1 - \mathcal{O}(u)}\,\|\boldsymbol{\mathcal{X}}_k\| \leq \mathcal{O}(1)\,\|\boldsymbol{\mathcal{X}}_k\|.$$

Substitution into (5.69) completes the proof. □

5.2 Mixed-precision reorthogonalized Pythagorean variants

For both BCGS-PIP+ and BCGS-PIPI+, the range of matrices to which the bound on the loss of orthogonality applies is limited by $\kappa(\boldsymbol{\mathcal{X}})$. For instance, using double precision can guarantee stability only when $\kappa(\boldsymbol{\mathcal{X}}) \leq \mathcal{O}(10^8)$. To extend the applicability of Algorithms 9 and 10, we examine the effects of using higher precision in certain parts of these algorithms, in a similar manner to what is done in Oktay and Carson [2023]. Our goal is to improve the accuracy of these algorithms using a higher precision without increasing communication cost significantly. We call these new variants BCGS-PIPI+$^{\mathrm{MP}}$ and BCGS-PIP+$^{\mathrm{MP}}$, respectively. We use double precision to store the data and solution and u^2 precision in the Cholesky factorization, the application of the corresponding inverses of R factors, and the inner products. We use precision u to perform the intrablock orthogonalization. Note again that precision u throughout this section corresponds to a precision having unit roundoff u.

To illustrate the use of higher precision in the new variants, BCGS-PIP+$^{\text{MP}}$ and BCGS-PIPI+$^{\text{MP}}$ are introduced in Algorithms 11 and 12, respectively. We obtain BCGS-PIP+$^{\text{MP}}$ via applying BCGS-PIP$^{\text{MP}}$ twice as in Algorithm 9 and performing line 3 in precision u. Note that the inner products in line 4 in Algorithm 8 and lines 4 and 8 in Algorithm 10 are split across two steps to handle different precisions. However we still regard these steps as one synchronization point.

Algorithm 11 $[\boldsymbol{\mathcal{Q}}, \mathcal{R}]$ =BCGS-PIP$^{\text{MP}}(\boldsymbol{\mathcal{X}}, \texttt{IntraOrtho})$

1: Allocate memory for $\boldsymbol{\mathcal{Q}}$, $\mathcal{R}$
2: $[\boldsymbol{Q}_1, R_{11}] = \texttt{IntraOrtho}\,(\boldsymbol{X}_1)$ in precision u
3: **for** $k = 2, \ldots, p$ **do**
4: $\quad \mathcal{R}_{1:k-1,k} = \boldsymbol{\mathcal{Q}}_{k-1}^T \boldsymbol{X}_k$ in precision u
5: $\quad P_k = \boldsymbol{X}_k^T \boldsymbol{X}_k$ in precision u^2
6: $\quad R_{kk} = \texttt{chol}\Big(P_k - \mathcal{R}_{1:k-1,k}^T \mathcal{R}_{1:k-1,k}\Big)$ in precision u^2, store in precision u
7: $\quad \boldsymbol{V}_k = \boldsymbol{X}_k - \boldsymbol{\mathcal{Q}}_{k-1} \mathcal{R}_{1:k-1,k}$ in precision u, store in precision u^2
8: $\quad \boldsymbol{Q}_k = \boldsymbol{V}_k R_{kk}^{-1}$ in precision u^2, store in precision u
9: **return** $\boldsymbol{\mathcal{Q}} = [\boldsymbol{Q}_1, \ldots, \boldsymbol{Q}_p]$, $\mathcal{R} = (R_{ij})$

Algorithm 12 $[\boldsymbol{\mathcal{Q}}, \mathcal{R}]$ = BCGS-PIPI+$^{\text{MP}}(\boldsymbol{\mathcal{X}}, \texttt{IntraOrtho})$

1: Allocate memory for $\boldsymbol{\mathcal{Q}}$, $\mathcal{R}$
2: $[\boldsymbol{Q}_1, R_{11}] = \texttt{IntraOrtho}\,(\boldsymbol{X}_1)$ in precision u
3: **for** $k = 2, \ldots, p$ **do**
4: $\quad \mathcal{S}_{1:k-1,k} = \boldsymbol{\mathcal{Q}}_{k-1}^T \boldsymbol{X}_k$ in precision u
5: $\quad \Omega_k = \boldsymbol{X}_k^T \boldsymbol{X}_k$ in precision u^2
6: $\quad S_{kk} = \texttt{chol}\Big(\Omega_k - \mathcal{S}_{1:k-1,k}^T \mathcal{S}_{1:k-1,k}\Big)$ in precision u^2
7: $\quad \boldsymbol{V}_k = \boldsymbol{X}_k - \boldsymbol{\mathcal{Q}}_{k-1} \mathcal{S}_{1:k-1,k}$ in precision u, store in precision u^2
8: $\quad \boldsymbol{U}_k = \boldsymbol{V}_k S_{kk}^{-1}$ in precision u^2, store in precision u
9: $\quad \mathcal{T}_{1:k-1,k} = \boldsymbol{\mathcal{Q}}_{k-1}^T \boldsymbol{U}_k$ in precision u
10: $\quad P_k = \boldsymbol{U}_k^T \boldsymbol{U}_k$ in precision u^2
11: $\quad T_{kk} = \texttt{chol}\Big(P_k - \mathcal{T}_{1:k-1,k}^T \mathcal{T}_{1:k-1,k}\Big)$ in precision u^2
12: $\quad \boldsymbol{W}_k = \boldsymbol{U}_k - \boldsymbol{\mathcal{Q}}_{k-1} \mathcal{T}_{1:k-1,k}$ in precision u, store in precision u^2
13: $\quad \boldsymbol{Q}_k = \boldsymbol{W}_k T_{kk}^{-1}$ in precision u^2, store in precision u
14: $\quad \mathcal{R}_{1:k-1,k} = \mathcal{S}_{1:k-1,k} + \mathcal{T}_{1:k-1,k} S_{kk}$ in precision u
15: $\quad R_{kk} = T_{kk} S_{kk}$ in precision u
16: **return** $\boldsymbol{\mathcal{Q}} = [\boldsymbol{Q}_1, \ldots, \boldsymbol{Q}_p]$, $\mathcal{R} = (R_{ij})$

We note that doubling the precision implies doubling the computation cost and the amount of data moved, which causes additional overhead. However, this overhead is highly dependent on the problem size and can be considered as a trade-off. Moreover, it can also be negligible in particular cases, such as latency-bound regimes and when both precisions are implemented in hardware. In such regimes, due to the high cost of communication, low-synchronization algorithms are beneficial and overhead can be insignificant. We also note that when higher precision computations are performed locally, such as line 7 in Algorithm 10, the extra overhead can be insignificant.

5.3 Numerical experiments

We perform several numerical experiments to study the stability of BCGS-PIP, BCGS-PIP+, BCGS-PIPI+, and their mixed-precision variants using three classes of matrices available in the `BlockStab` code suite[1], namely `glued`, `monomial`, and `default`. Each matrix class is created with dimensional inputs m, p, s, where m denotes the number of rows, p is the number of block vectors, and s is the number of columns in each block vector. The `glued` matrices are widely used in numerical experiments for CGS and its variants since they cause CGS to break down (Smoktunowicz et al. [2006]). The `monomial` matrices consist of p block vectors $\boldsymbol{X}_k = \begin{bmatrix} \boldsymbol{v}_k & A\boldsymbol{v}_k & \cdots & A^{s-1}\boldsymbol{v}_k \end{bmatrix}$, $k \in \{1, \ldots, p\}$, where each $\boldsymbol{v}_k$ is randomly generated from the uniform distribution and normalized, and A is an $m \times m$ diagonal operator having evenly distributed eigenvalues in $(0.1, 10)$. Lastly, the `default` matrices are constructed as $\boldsymbol{\mathcal{X}}_t = \boldsymbol{\mathcal{U}}\Sigma_t\boldsymbol{\mathcal{V}}^T$, where $\boldsymbol{\mathcal{U}} \in \mathbb{R}^{m\times ps}$ is orthonormal, $\boldsymbol{\mathcal{V}} \in \mathbb{R}^{ps\times ps}$ is unitary, and $\Sigma_t \in \mathbb{R}^{ps\times ps}$ is diagonal with entries drawn from the logarithmic interval $10^{[-t,0]}$. For our experiments, we set $m = 100$, $p = 10$, and $s = 2$ for `glued` and `default` matrices, and $m = 2000$, $p = 120$, $s = 10$ for the `monomial` matrices.

To illustrate the stability of algorithms, we plot the loss of orthogonality given in (5.2) and the relative Cholesky residual (calculated via (5.1) divided by $\|\boldsymbol{\mathcal{X}}\|^2$) of each algorithm versus the condition number of the matrix. To examine the effect of the `IntraOrtho` on each variant, we use Householder QR (`HouseQR`) and Cholesky (`CholQR`), where a variant of Cholesky factorization is used to bypass MATLAB's built-in `chol` function for halting the computation when a matrix loses numerical positive definiteness.

We use the Advanpix Toolbox (Advanpix LLC.) to simulate quadruple precision in our mixed-precision experiments. All numerical tests are run in MATLAB 2022a on a Lenovo ThinkPad E15 Gen 2 with 8GB memory and AMD Ryzen 5 4500U CPU with Radeon Graphics.

Figure 5.1 shows the stability of the reorthogonalized variants in comparison to BCGS-PIP. We see that BCGS-PIP only satisfies a $\mathcal{O}(u)\,\kappa^2(\boldsymbol{\mathcal{X}})$ loss of orthogonality bound for the set of `glued` matrices while the reorthogonalized variants satisfy $\mathcal{O}(u)$ when $\kappa(\boldsymbol{\mathcal{X}}) \leq 10^8$, regardless of the choice of `IntraOrtho`. After this point, we observe breakdowns in both variants that use `HouseQR` and `CholQR`, which was expected due to the theoretical bounds proved in Theorems 2 and 6. Note that missing points in both plots for large condition numbers (after the provable bound) emerged due to `NaN` being computed during Cholesky factorization.

[1]The version of the codes used in this study can be found at `https://github.com/katlund/BlockStab/releases/tag/v2.2024-beta`.

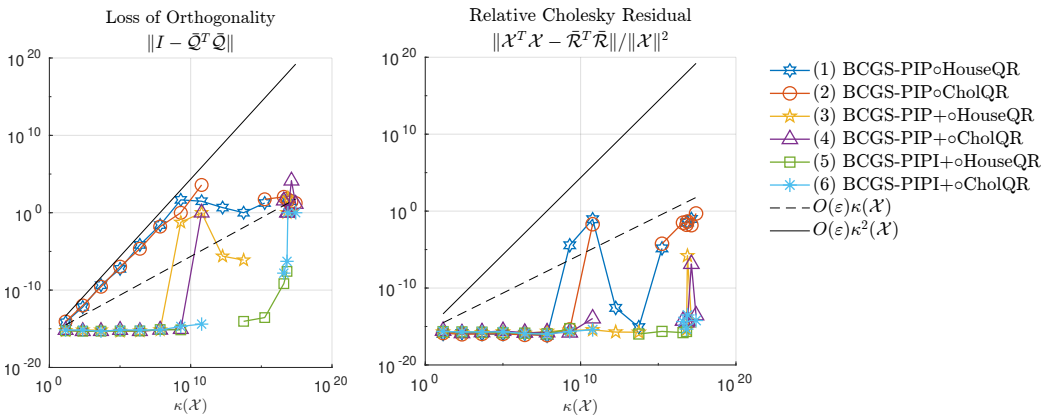

Figure 5.1: Loss of orthogonality and relative Cholesky residual plots for `glued` matrices.

Figure 5.2 shows the effect of the constraint on the loss of orthogonality of `IntraOrtho`. Since there is no assumption on the loss of orthogonality of the `IntraOrtho` in Theorem 2, we observe that Cholesky works well within the given theoretical bound for BCGS-PIP+ while we observe a breakdown before the proved theoretical bound for BCGS-PIPI+ due to the restriction in Theorem 6.

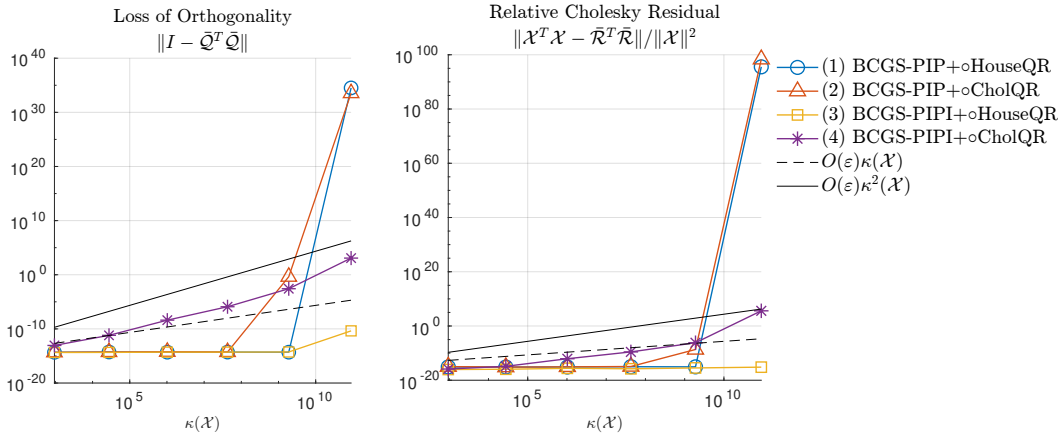

Figure 5.2: Loss of orthogonality and relative Cholesky residual plots for `monomial` matrices.

Figure 5.3 shows the effects of simulated mixed precision for the reorthogonalized variants, denoted by the superscript MP. From the figure, we see that after $\kappa(\boldsymbol{\mathcal{X}}) \approx 10^8$, BCGS-PIP+ has stability problems and the relative Cholesky residuals of BCGS-PIPI+ with Cholesky and BCGS-PIP+ with Householder QR become `NaN`. Using simulated mixed precision overcomes `NaN` for both methods. However, the loss of orthogonality of BCGS-PIPI+$^{\text{MP}}$ remains $\mathcal{O}(u)$ whereas BCGS-PIP+$^{\text{MP}}$ is still not completely below this bound.

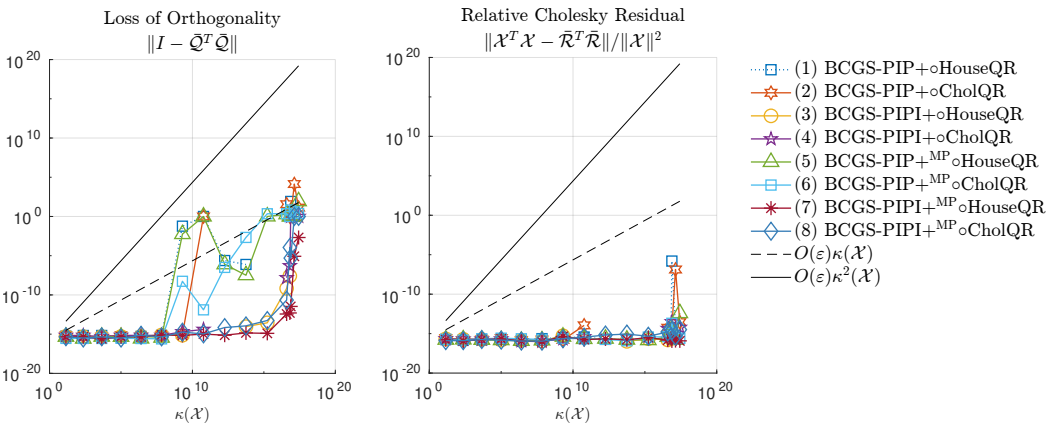

Figure 5.3: Loss of orthogonality and relative Cholesky residual plots for `glued` matrices.

Bibliography

Advanpix LLC. Multiprecision computing toolbox for MATLAB. URL `http://www.advanpix.com/`.

Jesse L Barlow and Alicja Smoktunowicz. Reorthogonalized block classical Gram–Schmidt. *Numerische Mathematik*, 123(3):395–423, 2013.

Erin Carson, Kathryn Lund, and Miroslav Rozložník. The stability of block variants of classical Gram–Schmidt. *SIAM Journal on Matrix Analysis and Applications*, 42(3):1365–1380, 2021. doi: 10.1137/21M1394424.

Stephan Ramon Garcia and Roger A. Horn. A Second Course in Linear Algebra, 2017.

G. H. Golub and C. F. Van Loan. *Matrix Computations*. Johns Hopkins Studies in the Mathematical Sciences. Johns Hopkins University Press, Baltimore, 4 edition, 2013. ISBN 978-1-4214-0794-4.

Nicholas J Higham. *Accuracy and stability of numerical algorithms*. SIAM, 2002.

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(1):111–130, 2012. doi: 10.1007/s13160-011-0053-x.

Eda Oktay and Erin Carson. Using mixed precision in low-synchronization reorthogonalized block classical Gram-Schmidt. *PAMM*, 23(1):e202200060, 2023.

Alicja Smoktunowicz, Jesse Barlow, and Julien Langou. A note on the error analysis of classical Gram-Schmidt. *Numerische Mathematik*, 105, 07 2006. doi: 10.1007/s00211-006-0042-1.

Y. Yamamoto, Y. Nakatsukasa, Y. Yanagisawa, and T. Fukaya. Roundoff error analysis of the Cholesky QR2 algorithm. *Electron. Trans. Numer. Anal.*, 44: 306–326, 2015.

6. Mixed-precision Rayleigh quotient iteration for total least squares problems[1]

The standard least squares (LS) problem involves solving $\min_x \|b - Ax\|_2$, where $A \in \mathbb{R}^{m\times n}$ with $m \geq n$ and $\text{rank}(A) = n$. Using the QR factorization

$$A = Q\bar{R} = [Q_1 \quad Q_2] \begin{bmatrix} R \\ 0 \end{bmatrix},$$

where $Q \in \mathbb{R}^{m\times m}$ is orthogonal and $R \in \mathbb{R}^{n\times n}$ is upper triangular, the solution to the LS problem is given by $x = U^{-1}Q_1^T b$ and the residual by $\|b - Ax\|_2 = \|Q_2^T b\|_2$. The LS problem can also be solved via the normal equations, $A^T Ax = A^T b$, which are equivalent to the augmented system

$$\begin{bmatrix} I & A \\ A^T & 0 \end{bmatrix} \begin{bmatrix} r \\ x \end{bmatrix} = \begin{bmatrix} b \\ 0 \end{bmatrix}.$$

Using mixed precision hardware can provide significant performance benefits in numerical linear algebra computations by reducing both computation and communication costs, which has motivated a flurry of recent work in this area; see, e.g., the surveys Abdelfattah et al. [2021], Higham and Mary [2022]. There are various existing mixed precision approaches to solving least squares problems, namely those based on performing iterative refinement on the augmented system. The augmented system formulation is originally due to Björck (Björck [1967]), who showed that iterative refinement for linear systems could be applied, using extra precision in the residual computation and making use of the QR factorization of A to solve a linear system with the augmented matrix in each iteration. For error bounds and implementation with BLAS operations, see Demmel et al. [2009]. Later, the authors in Carson et al. [2020] showed that GMRES-based iterative refinement can also be used, in which preconditioned GMRES is used as the solver within each refinement step. The QR factors of A, computed in potentially low precision, are used to construct a left preconditioner for the augmented matrix. This method is called GMRES-LSIR.

An alternative approach is to use mixed precision iterative refinement on the normal equations (Higham and Pranesh [2021]). Since $A^T A$ is symmetric positive definite, the Cholesky factor of this matrix, again potentially computed in lower precision (see also Yamazaki et al. [2015]), can be used to construct a preconditioner.

Although the LS problem is commonly encountered in applications, it may not be a realistic approach in some cases. Since LS problems are based on the standard linear model, $Ax = b + r$, with a random error vector r, it is assumed

[1]This chapter is a pre-copyedited, author-produced version of an article accepted for publication in Springer: Numerical Algorithms following peer review. The version of record [Numerical Algorithms, Oktay, E., Carson, E.: Mixed Precision Rayleigh Quotient Iteration for Total Least Squares Problems, (2023)] is available online at `https://link.springer.com/article/10.1007/s11075-023-01665-z`.

that the only error is in the right-hand side, b. However, especially in statistics, the matrix A may also affected by sampling and modeling errors. The errors-in-variables model represents errors in both A and b and is denoted as $(A+E)x = b+r$, where E and r are a random error matrix and vector, respectively.

The first numerically stable algorithm for solving the errors-in-variables model was introduced in Golub and Loan [1980]. The algorithm solves the minimization problem $min_{E,r}\|[E,r]\|_F$ subject to $(A+E)x = b+r$. Once a suitable (E,r) pair is found, any solution x to the system $(A+E)x = b+r$ is then considered to be a solution to the total least squares (TLS) problem.

The exact solution to the TLS problem is given by the singular value decomposition (SVD), $[A,b] = U\Sigma V^T$, where U is $m \times (n+1)$ with orthonormal columns, $\Sigma = \text{diag}(\sigma_1, \cdots, \sigma_{n+1})$, and $V = [v_1, \cdots, v_{n+1}]$ is an $(n+1) \times (n+1)$ orthogonal matrix. Assume that the singular values of the matrix $[A,b]$ are $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_{n+1} \geq 0$ and the singular values of A are $\sigma'_1 \geq \sigma'_2 \geq \cdots \geq \sigma'_n \geq 0$. If $\sigma_{n+1} = 0$, then $[E,r] = 0$. Otherwise, $\min_{\text{rank}(A+E,b+r)<n+1}\|[E,r]\|_F = \sigma_{n+1}$ and the TLS solution is

$$x_{TLS} = -\frac{1}{v_{n+1,n+1}}[v_{1,n+1}, \cdots, v_{n,n+1}]^T.$$

However, computing the SVD explicitly has a high computational cost and is thus not the preferred approach in practice.

In Golub and Loan [1980], the authors show that the approximate condition number for the TLS problem is

$$\kappa_{TLS}(A,b) = \frac{\sigma'_1}{\sigma'_n - \sigma_{n+1}} = \kappa(A)\frac{\sigma'_n}{\sigma'_n - \sigma_{n+1}},$$

where $\kappa(A)$ denotes the 2-norm condition number of A. As stated in Björck et al. [2000], when $1 - \sigma_{n+1}/\sigma'_n \ll 1$, $\kappa_{TLS}(A,b)$ can be much greater than $\kappa(A)$. Thus the TLS problem can be very sensitive to perturbations in the data.

To reduce the computational cost of solving the TLS problem, an iterative approach can be used. There are various iterative approaches to solving the TLS problem, such as inverse iteration. However, inverse iteration convergence depends on the ratio $\|\sigma_{n-p} - \mu\|/\|\sigma_{n-p+1} - \mu\|$, where p is the dimension of the desired right singular subspace of $[A,b]$ and μ is a shift for the iteration. If this ratio is not sufficiently small, the inverse iteration may converge very slowly. To overcome slow convergence, Chebyshev polynomials can be used, and thus, the method becomes (inverse) Chebyshev iteration. For detailed information, see Van Huffel [1991]. TLS problems can also have multiple right-hand sides. In this case, the block Golub-Kahan bidiagonalization procedure proposed in Hnětynková et al. [2013] can also be used to solve the problem.

In this work, we focus on the use of Rayleigh quotient iteration (RQI) to solve the TLS problem, which is the approach advocated by Björck et al. (Björck et al. [2000]) for large-scale problems. We introduce a mixed precision variant of the RQI-PCGTLS algorithm developed in Björck et al. [2000]. Our approach potentially decreases the computational cost of RQI-PCGTLS by using up to three different precisions in the algorithm. Using the ideas in Björck et al. [2000] and Connolly and Higham [2022], we discuss the convergence and accuracy of our algorithm. We derive two theoretical constraints on the precision that can be

used for the construction of the preconditioner for within the inner solver. Our numerical experiments suggest that our approach can attain the same accuracy as RQI-PCGTLS, potentially with a convergence delay. Our performance modeling shows, however, that we can achieve up to 4× speedup in the ideal case. Thus despite a delay in convergence, the mixed precision approach may still provide a faster time-to-solution with less computational cost.

We begin the next section by providing background information related to the existing RQI-based method, RQI-PCGTLS. In Section 6.2, we introduce our mixed precision approach, RQI-PCGTLS-MP, and present our analysis and performance modeling. In Section 6.3, we perform numerical experiments on several small-scale matrices that are widely used in the literature for TLS problems. We finally provide our concluding remarks in Section 6.4.

6.1 Rayleigh quotient iteration with preconditioned conjugate gradient method for TLS problems (RQI-PCGTLS)

Rayleigh quotient iteration is equivalent to inverse iteration with a shift ρ, where

$$\rho(x) = \frac{x^T B x}{x^T x}$$

is called the computed Rayleigh quotient of the matrix $B \in \mathbb{R}^{m \times m}$. The aim of RQI is to use the Rayleigh quotient to estimate the eigenvalue $\rho(x)$ corresponding to a given eigenvector x of B. The computed eigenvalue is iteratively improved within the algorithm to increase the rate of convergence of inverse iteration at each step. The general scheme of RQI is given in Algorithm 13.

Algorithm 13 RQI

Input: Given x_0 with $\|x_0\|_2 = 1$

1: $\rho_0 = (x_0)^T B x_0$
2: **for** k = 1,2,... **do**
3: Solve $(B - \rho_{k-1} I)\omega = x_{k-1}$
4: $x_k = \omega / \|\omega\|_2$
5: $\rho_k = (x_k)^T B x_k$

To deal with large sparse systems, Björck presented an approach using RQI to solve TLS problems in Björck [1997]. This approach solves the eigenvalue problem

$$\begin{bmatrix} A^T A & A^T b \\ b^T A & b^T b \end{bmatrix} \begin{bmatrix} x \\ -1 \end{bmatrix} = \lambda \begin{bmatrix} x \\ -1 \end{bmatrix} \tag{6.1}$$

to find $x = x_{TLS}$, where $\lambda = \sigma_{n+1}^2$. The problem can also be written as

$$\begin{bmatrix} A^T \\ b^T \end{bmatrix} (-r) = \lambda \begin{bmatrix} x \\ -1 \end{bmatrix},$$

where $r = b - Ax$ is the residual. The algorithm uses Newton's method to solve

$$\begin{bmatrix} f(x,\lambda) \\ g(x,\lambda) \end{bmatrix} = \begin{bmatrix} -A^T r - \lambda x \\ -b^T r + \lambda \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}. \tag{6.2}$$

The RQI algorithm for the augmented matrix in (6.1) is given in Algorithm 14. For the augmented matrix, if RQI converges to the TLS solution in n steps, the Rayleigh quotient ρ will be an estimate for σ_{n+1}^2. To ensure the convergence of RQI, the algorithm does one inverse iteration before starting RQI in lines 1-5 (Szyld [1988]). Within iteration loop k, the Rayleigh quotient $\rho_k = \sigma_k^2 = (r_k^T r_k)/(x_k^T x_k + 1)$ is computed in line 8. The next approximation x_{k+1} is computed by solving the system

$$\begin{bmatrix} J_k & A^T b \\ b^T A & \eta_k \end{bmatrix} \begin{bmatrix} x_{k+1} \\ -1 \end{bmatrix} = \beta_k \begin{bmatrix} x_k \\ -1 \end{bmatrix},$$

where $J_k = A^T A - \rho_k I$, $\eta_k = b^T b - \rho_k$, and β_k is the scaling factor at the k-th iteration. According to Björck et al. [2000], if J_k is positive definite, the solution can be obtained by block Gaussian elimination, which leads to

$$\begin{bmatrix} J_k & A^T b \\ 0 & \tau_k \end{bmatrix} \begin{bmatrix} x_{k+1} \\ -1 \end{bmatrix} = \beta_k \begin{bmatrix} x_k \\ -(z_k)^T x_k - 1 \end{bmatrix},$$

where $J_k z_k = A^T b$, and $\tau_k = b^T(b - Az_k) - \rho_k$.

We can also compute τ_k using f_k and g_k in each iteration using the formulas

$$J_k \omega_k = -f_k, \quad z_k = x_k + \omega_k, \quad \tau_k = (z_k)^T f_k - g_k. \tag{6.3}$$

The new way of computing τ_k can be used to express (6.2) with f_k and g_k, i.e.,

$$\begin{bmatrix} f_k \\ g_k \end{bmatrix} = \begin{bmatrix} -A^T r_k - \rho_k x_k \\ -b^T r_k + \rho_k \end{bmatrix},$$

where $r_k = b - Ax_k$ is the residual.

Algorithm 14 RQI for TLS (Björck et al. [2000])

1: $x_0 = x_{LS}$
2: $r_0 = b - Ax_0$
3: $\sigma_0^2 = r_0^T r_0/(x_0^T x_0 + 1)$
4: Solve $A^T A u_0 = x_0$
5: $x_1 = x_0 + \sigma_0^2 u_0$
6: **for** k=1,2,... **do**
7: $\quad r_k = b - Ax_k$
8: $\quad \sigma_k^2 = r_k^T r_k/(1 + x_k^T x_k)$
9: $\quad f_k = -A^T r_k - \sigma_k^2 x$
10: $\quad g_k = -b^T r_k + \sigma_k^2$
11: $\quad$ Solve $(A^T A - \sigma_k^2 I)\omega_k = -f_k$
12: $\quad z_k = x_k + \omega_k$
13: $\quad \beta_k = (z_k^T f_k - g_k)/(z_k^T x_k + 1)$
14: $\quad$ Solve $(A^T A - \sigma_k^2 I)u_k = x_k$
15: $\quad x_{k+1} = z_k + \beta_k u_k$

Performing one Rayleigh quotient iteration requires solving two linear systems (lines 11 and 14). Since the coefficient matrix $A^T A - \sigma^2 I$ is symmetric and positive definite, the conjugate gradient method can be applied to the problem. In Björck

et al. [2000], the RQI algorithm is used with a preconditioned conjugate gradient algorithm called PCGTLS to approximately solve these two systems using a fixed number of steps with the Cholesky factor of A^TA used as the preconditioner. Björck et al. call this approach RQI-PCGTLS (Björck et al. [2000]). Note that we need to form A^TA and compute its Cholesky factor R only once. The PCGTLS algorithm is given in Algorithm 15. The algorithm performs $k+1$ PCG steps in the k-th Rayleigh quotient iteration; the number of PCG steps is discussed further in Björck et al. [2000].

Algorithm 15 l-step PCGTLS for $(A^TA - \sigma^2 I)\omega = f$ (Björck et al. [2000])

Initialize $\omega_0 = 0$, $p_0 = s_0 = R^{-T}f$, $\eta_0 = \|s_0\|_2^2$
for $j = 0, 1, \ldots, l$, while $\delta_j \neq 0$ **do**
    $q_j = R^{-1}p_j$
    $\delta_j = \|p_j\|_2^2 - \sigma^2\|q_j\|_2^2$
    $\alpha_j = \eta_j/\delta_j$
    $\omega_{j+1} = \omega_j + \alpha_j q_j$
    $q_j = R^{-T}q_j$
    $s_{j+1} = s_j - \alpha_j(p_j - \sigma^2 q_j)$
    $\eta_{j+1} = \|s_{j+1}\|_2^2$
    $\beta_j = \eta_{j+1}/\eta_j$
    $p_{j+1} = s_{j+1} + \beta_j p_j$

In Algorithm 15, the Cholesky factor R of A^TA is assumed to be computed exactly and used to explicitly form the preconditioned matrix $\tilde{C} = I - \sigma^2 R^{-T}R^{-1}$. Thus, the matrix A is never used in PCGTLS except in forming the right-hand side vector f, which reduces the computational cost of the algorithm significantly. Therefore, forming the matrix A^TA and computing its Cholesky factor can be considered the most expensive parts of RQI-PCGTLS. This is especially true if A is not skinny, i.e., if n is close to m, since forming A^TA has cost $\mathcal{O}(n^2m)$, and Cholesky factorization has cost $\mathcal{O}(n^3)$.

6.2 Mixed precision RQI-PCGTLS (RQI-PCGTLS-MP)

One can use several techniques to reduce the computational cost of an algorithm. A popular approach, motivated by the recent emergence of commercially-available low-precision hardware, is the development of mixed precision algorithms, in which lower precision is used for the computationally dominant parts. However, one must use lower precision with caution, since this can cause loss of accuracy, convergence delay, and make the algorithm more susceptible to overflow and underflow.

Our goal is thus to use different precisions in different parts of the algorithm to safely balance accuracy, stability, and cost. In Freitag and Spence [2007], the authors state that the inverse iteration need not be performed exactly to be convergent. We therefore expect that mixed precision can be used within the RQI-PCGTLS algorithm. Moreover, assume that RQI solves the eigenvalue problem $(A - \Theta I)\omega = z$ for the eigenpair (Θ, ω) where z is an approximate eigenvector. The

authors in Simoncini and Eldén [2002] proved that this is mathematically equivalent to Newton's method on the unit sphere, i.e., solving $\Pi(A - \Theta I)\Pi d = -r$, where d is a correction with $z^*d = 0$, $\Pi = I - zz^*$, and $r = \Pi Az = Az - \Theta z$, if a Galerkin method is used to solve both systems. Thus, motivated by the mixed precision inexact Newton method introduced in Kelley [2022], we developed a mixed precision variant of RQI-PCGTLS, which we call RQI-PCGTLS-MP. We note that we expect that the analyses of RQI in Freitag and Spence [2007], Simoncini and Eldén [2002] and Newton in Higham and Mary [2022], Kelley [2022], Tisseur [2001] can be extended to give a complete analysis of RQI-PCGTLS-MP; we leave this as future work.

Our mixed precision approach, RQI-PCGTLS-MP, is presented in Algorithm 16. RQI-PCGTLS-MP uses three different precisions (denoted in the algorithm by prec.): u is the working precision to store data and solutions in the RQI-PCGTLS-MP algorithm, u_p is the working precision in the PCGTLS algorithm, and u_q is the precision for computing the QR factors of A. Throughout this work, we assume that $u \geq u_p \geq u_q$.

The authors of Björck et al. [2000] noted that one could use the R-factor from the QR decomposition of A instead of the Cholesky factor of A^TA. Since A is rectangular, the QR decomposition of the matrix can be written

$$A = [Q_1 \quad Q_2] \begin{bmatrix} R \\ 0 \end{bmatrix}.$$

From Björck [1996], we know that if A is of full rank, the R-factor in the QR decomposition of A is equal to the Cholesky factor of A^TA. However, the use of the Cholesky factor of A^TA results in a squaring of $\kappa(A)$, which is undesirable from a numerical perspective, especially if we hope to use low precision for this computation. Thus to enable the use of lower precision for more ill-conditioned systems, we use the R-factor from the Householder QR factorization of A instead of the Cholesky factorization of A^TA within RQI-PCGTLS-MP. We note that we have, however, observed experimentally that in some cases using the Cholesky factor of A^TA over the R-factor of A may be better in terms of RQI convergence rate. Determining the mechanism for these observations is left as future work.

To terminate the Rayleigh quotient iterations in Algorithm 16, we use the termination criteria given in Björck et al. [2000]. The authors in Björck et al. [2000] discuss that the convergence is directly related to the behavior of the normalized residual norm,

$$\psi_k = \left(\frac{\|f_k\|_2^2 + g_k^2}{\|x_k\|_2^2 + 1} \right)^{1/2}. \tag{6.4}$$

The method is converging in exact arithmetic as long as $\psi_{k+1} \leq \psi_k$. Hence, when $\psi_{k+1} > \psi_k$, we terminate the iterations.

Assuming exact arithmetic, there are several necessary conditions for the RQI-PCGTLS-MP algorithm (and thus also the RQI-PCGTLS algorithm) to converge. The first condition guarantees that the TLS problem has a unique solution, x_{TLS}, under exact arithmetic. If $\sigma'_n > \sigma_{n+1}$, then x_{TLS} is unique (Björck [1996]). The second condition is based on the assumptions required for the PCGTLS algorithm. If $A^TA - \sigma_k^2 I$ is symmetric and positive definite for all σ_k, then PCGTLS converges to an approximate solution in exact arithmetic (Björck et al. [2000]).

Algorithm 16 RQI-PCGTLS-MP

1: $x_0 = x_{LS}$
2: $r_0 = b - Ax_0$ prec. u
($2mn + m$ *ops*)
3: $\sigma_0^2 = r_0^T r_0/(1 + x_0^T x_0)$ prec. u
($2m + 2n - 2$ *ops*)
4: Compute $A = Q[R \quad 0]^T$ prec. u_q
($2mn^2 - 2n^3/3$ *ops*)
5: Solve $A^T A u_0 = x_0$ prec. u
($2n^2$ *ops*)
6: $x_1 = x_0 + \sigma_0^2 u_0$ prec. u
($2n$ *ops*)
7: **for** k = 1,2,... **do**
8: $r_k = b - Ax_k$ prec. u
($2mn + m$ *ops*)
9: $\sigma_k^2 = r_k^T r_k/(1 + x_k^T x_k)$ prec. u
($2m + 2n - 2$ *ops*)
10: $f_k = -A^T r_k - \sigma_k^2 x_k$ prec. u
($2mn + 2n$ *ops*)
11: $g_k = -b^T r_k + \sigma_k^2$ prec. u
($2m - 1$ *ops*)
12: Solve $(A^T A - \sigma_k^2 I)\omega_k = -f_k$ via PCGTLS ($\leq k + 1$ iterations) prec. u_p
($2n^2 + 14n - 3$ *ops per PCGTLS iteration* $+n^2 + 2n - 1$ *ops*)
13: $z_k = x_k + \omega_k$ prec. u
(n *ops*)
14: $\beta_k = (z_k^T f_k - g_k)/(z_k^T x_k + 1)$ prec. u
($4n - 2$ *ops*)
15: Solve $(A^T A - \sigma_k^2 I)u_k = x_k$ via PCGTLS ($\leq k + 1$ iterations) prec. u_p
($2n^2 + 14n - 3$ *ops per PCGTLS iteration* $+n^2 + 2n - 1$ *ops*)
16: $x_{k+1} = z_k + \beta_k u_k$ prec. u
($2n$ *ops*)

6.2.1 Constraints on factorization precision

In the case of very large A, the QR factorization (or Cholesky factorization) to construct the preconditioner is expected to be the most expensive part of the computation. Thus, we want to use as low a precision as possible for u_q. There are two potential sources from which a constraint on u_q arises.

First, we want the computed $\hat{R}$ factor to be nonsingular, since we will use it as a preconditioner. Note that here and in the remainder of the paper we use hats to denote quantities computed in finite precision. The precision required to guarantee the nonsingularity of $\hat{R}$ can be found in, e.g., [Higham, 2002, Theorem 19.4]. Suppose the Householder QR factorization of A is computed in precision u_q so that $A + \Delta A = \tilde{Q}_1 \hat{R}$, where $\hat{R} \in \mathbb{R}^{n\times n}$ and $\tilde{Q}_1 \in \mathbb{R}^{m\times n}$ has orthonormal columns. If

$$u_q < \frac{1}{cmn^{3/2}\kappa(A)},$$

for a small integer constant c, then $\hat{R}$ is guaranteed to be non-singular. We note that a probabilistic approach to rounding error analysis can loosen this bound by reducing the dimensional factors by their square roots. In the probabilistic model, the rounding errors $\gamma_1, \gamma_2, \ldots$ satisfying $fl(x \ \text{op} \ y) = (x \ \text{op} \ y)(1+\delta)$ with $|\delta| \leq u$ and $\text{op} = +, -, *, /$ are mean independent random variables of mean zero. The analysis in [Connolly and Higham, 2022, Theorem 4.4] proves that under the probabilistic model of rounding errors, applying the matrix concentration equality given in [Connolly and Higham, 2022, Theorem 3.2] results in a bound on the error of Householder QR in which dimensional factors may be replaced by their square roots, thus giving the constraint

$$u_q < \frac{1}{cm^{1/2}n^{3/4}\kappa(A)}.$$

As a rough heuristic, this means that we must have

$$u_q \lesssim \kappa(A)^{-1}. \tag{6.5}$$

A similar constraint can be obtained for the case where the Cholesky factorization of $fl(A^TA)$ is used to construct the preconditioner. From Wilkinson's result on Cholesky factorization in Wilkinson [1971], we say the Cholesky factorization of $fl(A^TA)$ succeeds in precision u_q if

$$u_q \leq \frac{1}{20n^{3/2}\kappa(A)^2},$$

or as a rough heuristic,

$$u_q \lesssim \kappa(A)^{-2}.$$

Further, using the results in Demmel [1989] and Higham and Pranesh [2021], if we introduce the scaled matrix H given by

$$H = D^{-1} fl(A^TA) D^{-1},$$

where D is a diagonal matrix with the square root of the diagonal elements of A^TA on the diagonal, we can say that if

$$u_q < \frac{\lambda_{min}(H)}{(2\lambda_{min}(H) + n)(n+1)},$$

where $\lambda_{min}(H)$ is the minimum eigenvalue of H, then the Cholesky factorization of $fl(A^TA)$ with two-sided diagonal scaling, run on $fl(A^TA)$ computed in precision u_q, will succeed.

Note that this result assumes that we can apply this scaling without incurring additional rounding error. For a dynamic approach to scaling and shifting for low-precision Cholesky factorization along with a complete analysis, see Higham and Pranesh [2021]. In our numerical experiment involving Cholesky factorization in Section 6.3, we will apply a two-sided scaling for equilibration before Cholesky factorization to avoid overflow and underflow, which may result from rounding higher precision numbers to a lower precision.

Our second constraint on u_q will come from the requirement that the preconditioned system constructed using the computed $\hat{R}$ must remain positive definite. Assume we are computing R via Householder QR factorization. Then we have

$$A+\Delta A = Q_1\hat{R} \quad \text{with} \quad \|\Delta A\|_F \leq \tilde{\gamma}^q_{mn}\|A\|_F, \quad \text{where} \quad \tilde{\gamma}^q_{mn} = \frac{cmnu_q}{1-cmnu_q} \tag{6.6}$$

for some constant c, where the subscript F denotes the Frobenius norm. Note that if $A = Q_1R$, we have

$$\|\tilde{Q}_1 - Q_1\|_F \leq \sqrt{m}\tilde{\gamma}^q_{mn}; \tag{6.7}$$

see, e.g., [Carson et al., 2020, Equation (2.3)].

When we take the inexactness of $\hat{R}$ due to computation in precision u_q into account in determining the preconditioned coefficient matrix $\hat{C}$, we have

$$\begin{aligned}
\hat{C} &= \hat{R}^{-T}(A^TA - \sigma^2 I)\hat{R}^{-1} \\
&= \hat{R}^{-T}A^TA\hat{R}^{-1} - \sigma^2\hat{R}^{-T}\hat{R}^{-1} \\
&= (\tilde{Q}_1^T - \hat{R}^{-T}\Delta A^T)(\tilde{Q}_1 - \Delta A\hat{R}^{-1}) - \sigma^2\hat{R}^{-T}\hat{R}^{-1} \\
&\approx I - \tilde{Q}_1^T\Delta A\hat{R}^{-1} - \hat{R}^{-T}\Delta A^T\tilde{Q}_1 - \sigma^2\hat{R}^{-T}\hat{R}^{-1}.
\end{aligned} \tag{6.8}$$

We must now investigate the last term on the right-hand side above. We can write (using a first order approximation)

$$\begin{aligned}
\sigma^2\hat{R}^{-T}\hat{R}^{-1} &= \sigma^2\left((\hat{R}^T - R^T) + R^T\right)^{-1}\left((\hat{R}-R)+R\right)^{-1} \\
&= \sigma^2\left(R^T\left(I - R^{-T}(R^T - \hat{R}^T)\right)\right)^{-1}\left(R\left(I - R^{-1}(R-\hat{R})\right)\right)^{-1} \\
&\approx \sigma^2\left(R^{-T} + R^{-T}(R^T - \hat{R}^T)R^{-T}\right)\left(R^{-1} + R^{-1}(R-\hat{R})R^{-1}\right) \\
&\approx \sigma^2\left(R^{-T}R^{-1} + R^{-T}R^{-1}(R-\hat{R})R^{-1} + R^{-T}(R^T - \hat{R}^T)R^{-T}R^{-1}\right).
\end{aligned}$$

Using that

$$\hat{R} - R = (\tilde{Q}_1 - Q_1)^TA + \tilde{Q}_1^T\Delta A,$$

the above becomes

$$\begin{aligned}
\sigma^2\hat{R}^{-T}\hat{R}^{-1} \approx \sigma^2\Big(&R^{-T}R^{-1} + R^{-T}R^{-1}\big((\tilde{Q}_1 - Q_1)^TA + \tilde{Q}_1^T\Delta A\big)R^{-1} \\
&+ R^{-T}\big(A^T(\tilde{Q}_1 - Q_1) + \Delta A^T\tilde{Q}_1\big)R^{-T}R^{-1}\Big).
\end{aligned}$$

Substituting this into (6.8), we have

$$\begin{aligned}\hat{C} &\approx I - \sigma^2 R^{-T}R^{-1} - \tilde{Q}_1^T \Delta A \hat{R}^{-1} - \hat{R}^{-T}\Delta A^T \tilde{Q}_1 \\ &\quad + \sigma^2 R^{-T}R^{-1}\big((\tilde{Q}_1 - Q_1)^T A + \tilde{Q}_1^T \Delta A\big)R^{-1} \\ &\quad + \sigma^2 R^{-T}\big(A^T(\tilde{Q}_1 - Q_1) + \Delta A^T \tilde{Q}_1\big)R^{-T}R^{-1} \\ &\equiv \tilde{C} + \underline{\Delta},\end{aligned}$$

where $\tilde{C}$ is the preconditioned system with exact R, i.e., $\tilde{C} = I - \sigma^2 R^{-T}R^{-1}$.

We now want to bound the norm of $\underline{\Delta}$. Using the definition of $\underline{\Delta}$ above, along with (6.6) and (6.7), and ignoring terms of order $O(u_q^2)$, we have

$$\begin{aligned}\|\underline{\Delta}\|_2 &\le 2\|\Delta A\|_2 \|\hat{R}^{-1}\|_2 + 2\sigma^2 \|\hat{R}^{-1}\|_2^3 \left(\|\tilde{Q}_1 - Q_1\|_2 \|A\|_2 + \|\Delta A\|_2\right) \\ &\le \tilde{\gamma}_{mn}^q \kappa_F(A) + \sigma^2 \|A^{-1}\|_2^3 \cdot \sqrt{m}\tilde{\gamma}_{mn}^q \|A\|_2 \\ &\le \left(1 + \sigma^2 \|A^{-1}\|_2^2\right)\sqrt{m}\tilde{\gamma}_{mn}^q \kappa_F(A),\end{aligned}$$

where $\kappa_F(A)$ denotes the Frobenius norm condition number. Assuming that $\sigma = \sigma_{n+1}$, we have $\sigma \le 1/\|A^{-1}\|_2$, and thus

$$\|\underline{\Delta}\|_2 \le \sqrt{m}\tilde{\gamma}_{mn}^q \kappa_F(A).$$

Using eigenvalue perturbation theory, we can say how far the eigenvalues of $\hat{C}$ are from $\tilde{C}$ for the case $\sigma = \sigma_{n+1}$. From (4.4) in Björck et al. [2000], we know that in this case the eigenvalues of $\tilde{C}$ lie in the interval

$$\left[1 - \frac{\sigma_{n+1}^2}{(\sigma_n')^2}, 1\right],$$

and so we expect the exact preconditioner to work very well unless $\sigma_{n+1}^2 \approx (\sigma_n')^2$, in which case the condition number may be large and thus fast convergence of CG is not guaranteed.

For $\hat{C}$ with *inexact* $\hat{R}$, we know the eigenvalues must lie in the interval

$$\left[1 - \frac{\sigma_{n+1}^2}{(\sigma_n')^2} - \|\underline{\Delta}\|_2, 1 + \|\underline{\Delta}\|_2\right].$$

This tells us two things. First, for the computed $\hat{R}$, we expect the preconditioner to work well unless

$$\sigma_{n+1}^2 \approx (1 + \|\underline{\Delta}\|_2)(\sigma_n')^2.$$

Second, we must have that

$$1 - \frac{\sigma_{n+1}^2}{(\sigma_n')^2} - \|\underline{\Delta}\|_2 > 0,$$

which implies

$$\|\underline{\Delta}\|_2 < 1 - \frac{\sigma_{n+1}^2}{(\sigma_n')^2},$$

or else we cannot guarantee the positive definiteness of the preconditioned system.

As a rough heuristic, this means that we want

$$u_q \kappa_F(A) \lesssim 1 - \frac{\sigma_{n+1}^2}{(\sigma_n')^2}.$$

In other words, we must have

$$u_q \lesssim \kappa_F(A)^{-1} \left(1 - \frac{\sigma_{n+1}^2}{(\sigma_n')^2}\right). \tag{6.9}$$

Note that this means that for TLS problems, in contrast to standard least squares problems, our choice of precision should depend not only on A but also on the right-hand side b.

To compare the bounds given in (6.5) and (6.9), we need to look at $\sigma_{n+1}^2/(\sigma_n')^2$. When σ_{n+1} is close to zero, i.e., b is almost in the column space of A, $\sigma_{n+1}^2/(\sigma_n')^2$ is very small, and thus the two constraints become very close to each other. Otherwise, if $\sigma_{n+1}^2/(\sigma_n')^2$ is close to 1, then the constraint on u_q in (6.9) is much tighter. In this case, we need to use a higher precision for u_q.

6.2.2 Performance modeling

To evaluate to what extent the computational cost can be reduced by using the mixed precision variant with Householder QR factorization, we construct a performance model which takes as input the size (m, n) of A (with $m \geq n$) and the number r of the Rayleigh quotient iterations performed by RQI-PCGTLS and RQI-PCGTLS-MP. The model assumes that PCGTLS performs exactly $k + 1$ iterations in the k-th Rayleigh quotient iteration. We also note that our performance model is based on computational complexity and thus essentially assumes an ideal model in which data movement is not dominant; this may not be realistic in practice.

Using the operation counts in Table 6.1, we calculate the cost of each line in Algorithm 16. Then we multiply them by a constant depending on the chosen precision. For unit roundoff of different precisions, see Table 1. Since the default working precision is fp64, we use 1 for double, 0.5 for single, and 0.25 as the constant for half precision. For instance, line 2 in Algorithm 16, $r = b - Ax$, performs one matrix-vector product and a vector update. Therefore this line requires $2mn - m + 2m = 2mn + m$ operations. Since we use double precision for u in our experiments, the cost of this line becomes $1 \times (2mn + m) = 2mn + m$. In the end, the total operation count for the RQI-PCGTLS-MP algorithm depends on the size of A, the precision setting, and the number r of Rayleigh quotient iterations performed and can be written as

$$\begin{aligned} cost(m, n, r, c, c_p, c_q) = {} & c(2mn + 3m + 4n + 2n^2 - 2 + r(4mn + 5m + 11n - 5)) \\ & + c_p(2r(n^2 + 2n - 1) + (r^2 + 3r)(2n^2 + 14n - 3)) \\ & + c_q \left(2mn^2 - \frac{2n^3}{3}\right), \end{aligned}$$

where (c, c_p, c_q) are the constants used for (u, u_p, u_q), respectively. For the RQI-PCGTLS algorithm, we use the same *cost* function with $(c, c_p, c_q) = (1, 1, 1)$.

Table 6.1: Cost and places of each operation used in RQI-PCGTLS-MP. In the table, we use the abbreviations QR decomp (QR decomposition), Subs (Substitutions), MatVec (Matrix-Vector multiplication Ab), VecMul (Inner product $b^T c$), VecUpdate (Vector update $\lambda b + c$).

operation	cost	lines in Algorithm 16	lines in Algorithm 15
QR decomp of $A^{m\times n}$	$2mn^2 - 2n^3/3$	4	-
Forward/Backward Subs with $R^{n\times n}$	n^2	5	1, 3, 7
MatVec with $A^{m\times n}, b^{n\times 1}$	$2mn - m$	2, 8, 10	-
VecMul with $b^{n\times 1}, c^{n\times 1}$	$2n - 1$	3, 9, 11, 14	1, 4, 9
VecUpdate with $b^{n\times 1}, c^{n\times 1}, \lambda \in \mathbb{R}$	$2n$	2, 6, 8, 10, 13, 16	6, 8, 11

Then we calculate the speedup of RQI-PCGTLS-MP as

$$speedup = \frac{cost \text{ of RQI-PCGTLS-MP}}{cost \text{ of RQI-PCGTLS}}.$$

Figure 6.1 illustrates the speedups one can obtain using the $(u, u_p, u_q) =$ (double, single, half) setting with several values of r. Using this mixed precision setting can provide up to $\times 4$ speedup in the best-case scenario, which comes from the use of half precision (fp16) for the dominant part of the computation. From the figure, we observe that even for a tall-skinny matrix, our mixed precision approach can perform more than 2 times faster than the RQI-PCGTLS algorithm. We also see that we get a greater speedup the larger n grows. The best-case scenario is obtained when both m and n are large. In these cases, we can obtain $4\times$ modeled speedup even with 2000 Rayleigh quotient iterations. One should note that r depends on the matrix. Thus for a large-scale matrix, r may be larger than 2000. However, the speedup will never drop below 1 as long as u is not higher than the precision RQI-PCGTLS uses.

6.3 Numerical experiments

In this section, we present the numerical results comparing RQI-PCGTLS and RQI-PCGTLS-MP for solving TLS problems that are commonly used in the literature. To illustrate the comparison of the methods in terms of relative error, we perform numerical experiments in MATLAB with various matrices. The experiments are performed on a computer with AMD Ryzen 5 4500U having 6 CPUs and 8 GB RAM with OS system Ubuntu 22.04 LTS. Our RQI-PCGTLS-MP algorithm and associated functions are available at `https://github.com/edoktay/rqipc-gtlsmp`, which includes scripts for generating the data and plots in this work.

Recall that in the k-th Rayleigh quotient iteration, both algorithms perform at most $k+1$ PCGTLS iterations in each of lines 12 and 15 of Algorithm 16. Recall also that we terminate both RQI-PCGTLS and RQI-PCGTLS-MP algorithms when the normalized residual norm increases, i.e., $\psi_{k+1} > \psi_k$, where ψ_k is defined in (6.4). For a detailed explanation, see Section 6.2.

Each experiment contains a random error matrix and vector. For reproducibility of random arrays for the experiments (except Example 1 in Section 6.3.1), we use the MATLAB command `rng(1)` each time we run the algorithm.

Each plot shows the relative error versus the number of Rayleigh quotient iterations performed. Dashed lines represent the relative error `rerrx` in the ap-

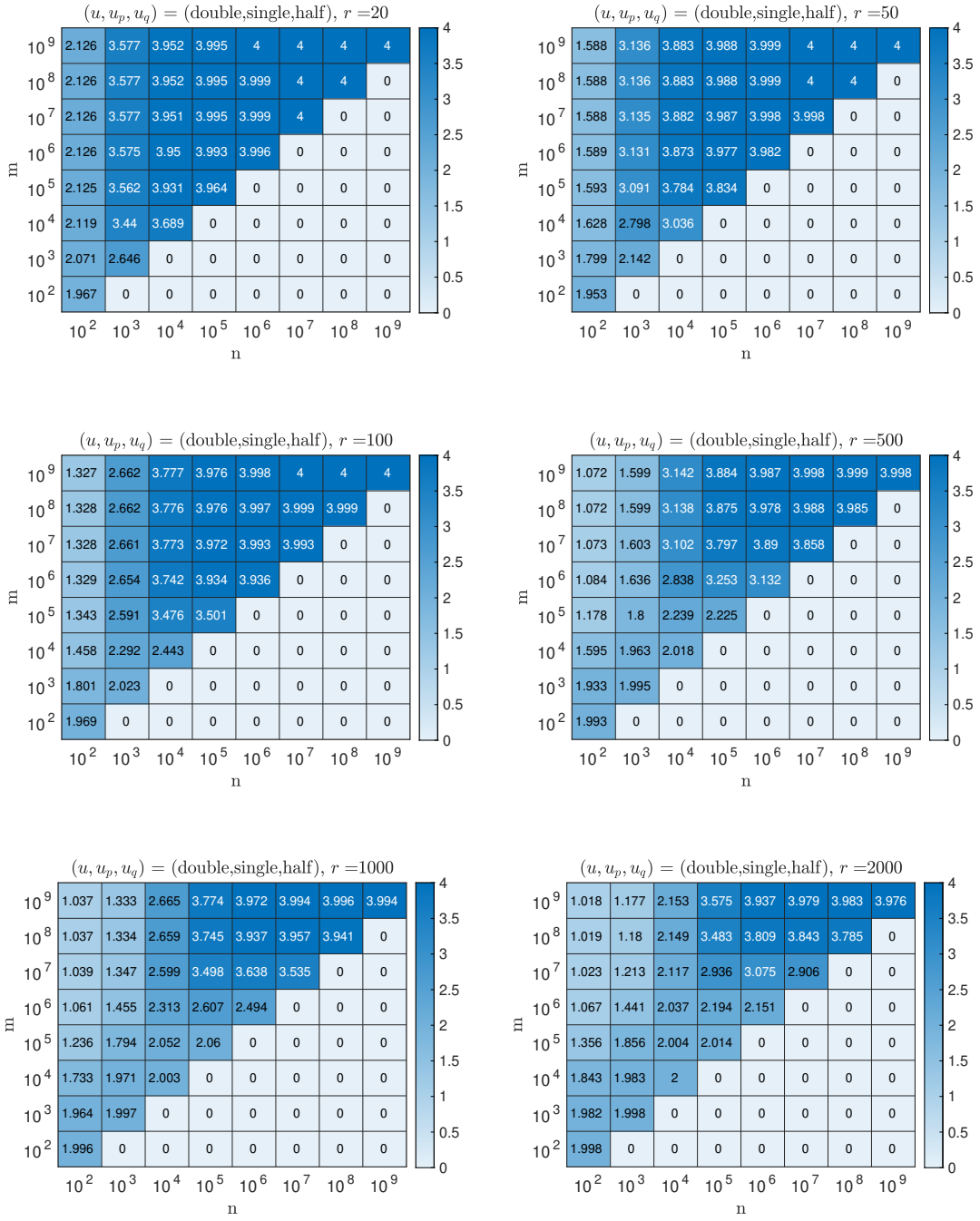

Figure 6.1: Modeled speedup for $A \in \mathbb{R}^{m \times n}$ (for $m \geq n$) with $(u, u_p, u_q) =$ (double, single, half) and various numbers r of Rayleigh quotient iterations performed by RQI-PCGTLS and RQI-PCGTLS-MP.

proximate solution, i.e., $\|x_k - x\|_2/\|x\|_2$, whereas solid lines represent the relative error `rerrs` in the Rayleigh quotient after k Rayleigh quotient iterations, i.e., $|\sigma_{n+1}^{(k)} - \sigma_{n+1}|/|\sigma_{n+1}|$. The quantity `rerrs` is calculated with σ_{n+1} since the Rayleigh quotient iteratively approaches σ_{n+1}^2. Thus, `rerrs` calculates the convergence of $\sigma_{n+1}^{(k)}$ to σ_{n+1}. Red lines show errors in RQI-PCGTLS-MP and blue lines in RQI-PCGTLS. We used the Advanpix toolbox (Advanpix LLC.) to simulate quadruple precision for the computation of the plotted quantities, which is not included in the performance analysis.

In our numerical experiments, we used double precision in MATLAB for the RQI working precision u to be able to compare with RQI-PCGTLS (performed in double precision). For each problem, we evaluate the constraints on u_q given in (6.5) and (6.9), and show that half precision satisfies both constraints for each of the tested problems. To simulate half precision, we use the `chop` library and associated functions from Higham and Pranesh [2019], available at `https://github.com/higham/chop` and `https://github.com/SrikaraPranesh/LowPrecision_Simulation`.

Although we do not provide a constraint on u_p, we found that using a precision less than fp32 in RQI-PCGTLS-MP when u is fp64 gives `rerrx` and `rerrs` much greater than the fp64 unit round-off. Thus, RQI-PCGTLS-MP fails to converge to x and σ_{n+1}^2. Although single precision for u_p was sufficient for all included examples, we give an example where lower than single precision for u_p does not work well in Section 6.3.3. We also note that, although not shown for each experiment, we have also tested the use of Cholesky factorization instead of Householder QR to compute the preconditioner for PCG. In many cases, the use of Cholesky resulted in a greater number of Rayleigh quotient iterations required for convergence for RQI-PCGTLS and/or RQI-PCGTLS-MP. An exception to this is discussed in Section 6.3.5.

6.3.1 Example 1: Random matrix

We first test our algorithm on a random matrix A of dimension $(m, n) = (100, 60)$ having uniformly distributed random elements. We generated the matrices using the MATLAB command `rand(m,n)`. A vector of ones is used for the right-hand side vector b. The error matrix E and vector e are added to the left and right-hand sides, respectively. The final system can be written as $(A + E)x = b + e$, where $E = \epsilon\cdot$`rand(m,n)` and $e = \epsilon\cdot$`rand(m,1)`. For this example, we use the MATLAB command `rng(0)` for reproducibility and ϵ is chosen as 10^{-6}.

For the random matrix, using constraints (6.5) and (6.9), we obtain the bounds $u_q < 2 \times 10^{-2}$ and $u_q < 4 \times 10^{-3}$, respectively. Thus, half precision is guaranteed to work. The left plot in Figure 6.2 shows the relative error behavior of RQI-PCGTLS and RQI-PCGTLS-MP. We see that RQI-PCGTLS performs 8 Rayleigh quotient iterations, whereas RQI-PCGTLS-MP performs 13 before the stopping criterion is satisfied. The mixed precision variant performing more iterations than the uniform precision variant is expected, since we may have a worse preconditioner and thus a worse solution produced by the inner PCG solve for a fixed number of iterations.

It can be more illustrative to look at the point at which the rate of improvement of the algorithm slows down significantly. For `rerrs`, we see that

RQI-PCGTLS improvement starts to slow down at the fourth iteration, and for the RQI-PCGTLS-MP algorithm, it takes six iterations. Importantly, note that the limiting accuracy for both `rerrs` and `rerrx` is similar for both RQI-PCGTLS and RQI-PCGTLS-MP.

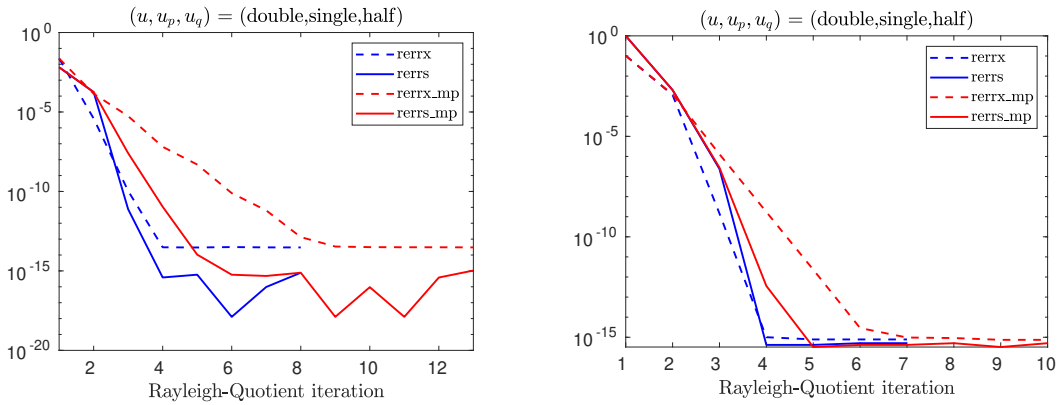

Figure 6.2: Relative errors versus Rayleigh quotient iterations for the random (left) and δ (right) matrices.

6.3.2 Example 2: The δ matrix

We now test our algorithm on the system $(A + E)x = b + e$ where A is a matrix used in Diao and Sun [2018], which we call the δ matrix. The system $Ax = b$ is constructed as

$$\begin{bmatrix} \delta & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & \delta & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ . \\ . \\ . \\ x_9 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ . \\ . \\ . \\ 1 \end{bmatrix}.$$

The error matrix $E = \epsilon \bar{E} \cdot A$, and error vector $e = \epsilon \bar{e} \cdot b$ are constructed with $\epsilon = 10^{-1}$, $\delta = 10^{-2}$, and a uniformly distributed random matrix and vector with entries in the interval $(-1, 1)$, $\bar{E}$ and $\bar{e}$, respectively. For this example, since $\psi_{k+1} \approx \psi_k$ from the tenth iteration onward, we slightly modify the stopping criterion and use $\psi_{k+1} \geq \psi_k$ for this test case.

For the δ matrix, using constraints (6.5) and (6.9), we obtain the bounds $u_q < 9 \times 10^{-3}$ and $u_q < 1 \times 10^{-3}$, respectively. Thus, half precision is guaranteed to work in this case as well.

The right plot in Figure 6.2 shows the relative error behavior of RQI-PCGTLS and RQI-PCGTLS-MP for the δ matrix. Here, RQI-PCGTLS performs 7 Ray-

leigh quotient iterations before the stopping criterion is met, whereas RQI-PCGTLS-MP performs 10. When we look at the point where the rate of improvement in `rerrs` sufficiently slows down, we see that RQI-PCGTLS-MP requires only one iteration more to have approximately the same error as RQI-PCGTLS. Again, another important observation from the plot is that although the mixed precision variant performs more iterations, both RQI-PCGTLS variants reached a similar limiting accuracy. This again shows that using mixed precision does not cause a loss of accuracy for this problem.

6.3.3 Example 3: The Björck matrix

Next, we use the matrix A that is used to test the convergence properties of RQI-PCGTLS in Björck et al. [2000], which we call the Björck matrix. The TLS problem is defined as $(A + E)x = b + e$, where

$$A = Y \begin{bmatrix} D \\ 0 \end{bmatrix} Z^T \in \mathbb{R}^{m \times n},$$

$b = Ax$ with $(m, n) = (30, 15)$, $x = (1, 1/2, \ldots, 1/n)$, $D = \mathrm{diag}(1, 2^{-1}, \ldots, 2^{-n+1})$, and Y, Z are random orthogonal matrices generated using the MATLAB command `RandOrthMat()`. The error matrix $E = \epsilon$·`rand(m,n)` and vector $e = \epsilon$·`rand(m,1)` are generated with $\epsilon = 0.05$.

For the Björck matrix, using constraints (6.5) and (6.9), we obtain the bounds $u_q < 4 \times 10^{-2}$ and $u_q < 5 \times 10^{-3}$, respectively. Thus, half precision is guaranteed to work.

In the left plot in Figure 6.3, which uses the usual precision settings, we see similar behavior as in the previous examples. Therefore, we again conclude that while RQI-PCGTLS-MP can result in a convergence delay, both algorithms eventually converge to a similar level. The right plot shows the same experiment, but with u_p set to half instead of single. In this case, while the relative error in the approximate solution given in the dashed line is around 10^{-14} for RQI-PCGTLS, it is only around 10^{-4} with RQI-PCGTLS-MP.

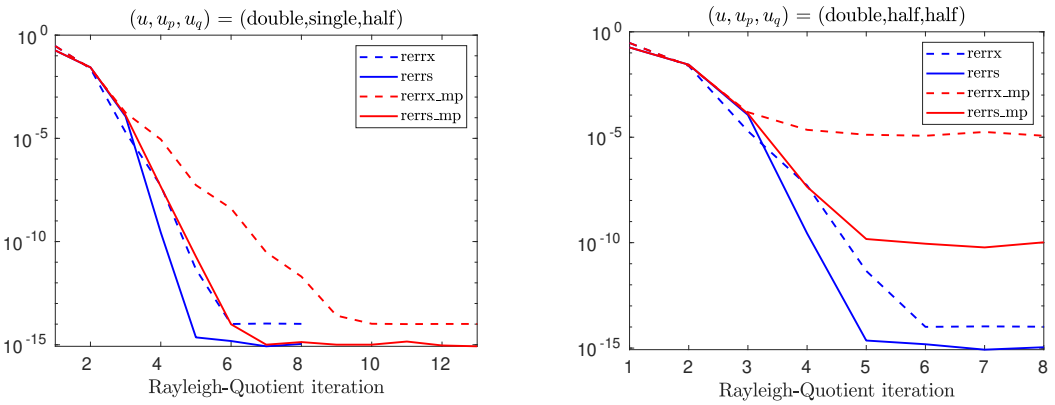

Figure 6.3: Relative errors versus Rayleigh quotient iterations for the Björck matrix using single precision (left) and half precision (right) for u_p.

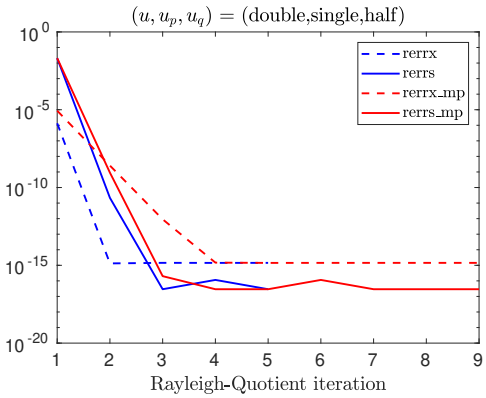

Figure 6.4: Relative errors versus Rayleigh quotient iterations for the Toeplitz matrix.

6.3.4 Example 4: The Toeplitz matrix

The matrix A is constructed as $A = \bar{T} + E$, where E is a random Toeplitz matrix and $\bar{T} \in \mathbb{R}^{n\times(n-2\omega)}$ is the lower Toeplitz matrix having elements

$$t_{i,1} = \begin{cases} \frac{1}{\sqrt{2\pi\alpha^2}} e^{[\frac{-(\omega-i+1)^2}{2\alpha^2}]} & i = 1, 2, \ldots, 2\omega+1, \\ 0 & \text{otherwise}, \end{cases}$$

in the first row and

$$t_{1,j} = \begin{cases} t_{1,1} & \text{if } j = 1, \\ 0 & \text{otherwise} \end{cases}$$

in the first column. This matrix comes from an application in signal restoration (Trussell [1983]). The matrices $\bar{T}$ and E are constructed using the built-in MATLAB function `toeplitz()` with the input of the first column of $\bar{T}$. The right-hand side vector is constructed as $b = \bar{b} + e$, where $\bar{b}$ is a vector of ones, and e is a random vector. For this test, we use $n = 100$, $\omega = 2$, and $\alpha = 1.25$. We scale E and e with a scaling factor of 0.001.

For the Toeplitz matrix, using constraints (6.5) and (6.9), we obtain the bounds $u_q \lesssim 1$ and $u_q \lesssim 10^{-2}$, respectively. Thus, half precision is guaranteed to work here.

Even though Figure 6.4 shows that `rerrs` oscillates after the third iteration for both uniform and mixed precision variants, the oscillations are negligible since they happened after the error reaching the level indicated by the working precision, $u \approx 10^{-16}$. We also see that, as in previous examples, both variants converge to a similar level although mixed precision may result in slightly delayed convergence.

6.3.5 Example 5: The Van Huffel matrix

Finally, we test the matrix with $n = 100$ from Van Huffel and Vandewalle [1991], which is used to illustrate the difference between LS and TLS solutions. We

define the TLS problem by $(A+E)x = b+e$, where $Ax = b$ is given as

$$\underbrace{\begin{bmatrix} n-1 & -1 & \cdots & -1 \\ -1 & n-1 & \cdots & -1 \\ . & & & \\ . & & & \\ . & & & \\ -1 & -1 & \cdots & n-1 \\ -1 & -1 & \cdots & -1 \\ -1 & -1 & \cdots & -1 \end{bmatrix}}_{n \times n-2} \begin{bmatrix} x_1 \\ . \\ . \\ . \\ x_{n-3} \\ x_{n-2} \end{bmatrix} \approx \begin{bmatrix} -1 \\ -1 \\ . \\ . \\ . \\ -1 \\ n-1 \\ -1 \end{bmatrix},$$

and $E = \epsilon \cdot$`rand(n,n-2)`, $e = \epsilon \cdot$`rand(n,1)` are the error matrix and vector generated with $\epsilon = 10^{-6}$, respectively.

For the Van Huffel matrix, using constraints (6.5) and (6.9), we obtain the bounds $u_q < 2 \times 10^{-1}$ and $u_q < 5 \times 10^{-3}$, respectively. Thus, half precision is guaranteed to work.

We see exceptional behavior on the RQI-PCGTLS-MP algorithm for this example in the left plot of Figure 6.5. Here, we observe that using mixed-precision causes the algorithm to perform over $5\times$ more Rayleigh quotient iterations than RQI-PCGTLS if the QR factorization of A is used as a preconditioner in PCGTLS (although at the end, they again both converge to a similar level). For the comparison of RQI-PCGTLS and our mixed precision variant with this matrix, we also tried using the Cholesky factorization as a preconditioner within both variants. For low-precision Cholesky factorization with two-sided diagonal scaling, we use the `cgir3.m` function available in `https://github.com/SrikaraPranesh/Multi_precision_NLA_kernels`. In our experiments, we see that for this example, using the Cholesky factor of A^TA is ultimately better than using the QR factorization within RQI-PCGTLS-MP; from the plot on the right, the mixed precision variant with the Cholesky factorization performs only 3 and 6 more iterations to converge to a similar level as RQI-PCGTLS for `rerrs` and `rerrx`, respectively. Thus it is clear that for some matrices, Cholesky factorization makes a better preconditioner than QR factorization within RQI-PCGTLS-MP. A full explanation for this behavior is left as future work.

6.4 Conclusion

In this paper, we present a mixed precision variant of RQI-PCGTLS called RQI-PCGTLS-MP. Our algorithm allows the use of up to three different precisions in different parts of the algorithm. To allow the use of lower precision for more ill-conditioned systems, we advocate for the use of the Householder QR factorization of A instead of the Cholesky factorization of A^TA to construct a low-precision preconditioner for PCGTLS. We derive constraints on the precision with which the QR factorization can be computed. Interestingly, in contrast with the standard least squares case, the factorization precision we can use depends not only on A but on the right-hand side b in the total least squares setting.

The numerical experiments we performed show that using mixed precision increases the number of Rayleigh quotient iterations performed; however, since we

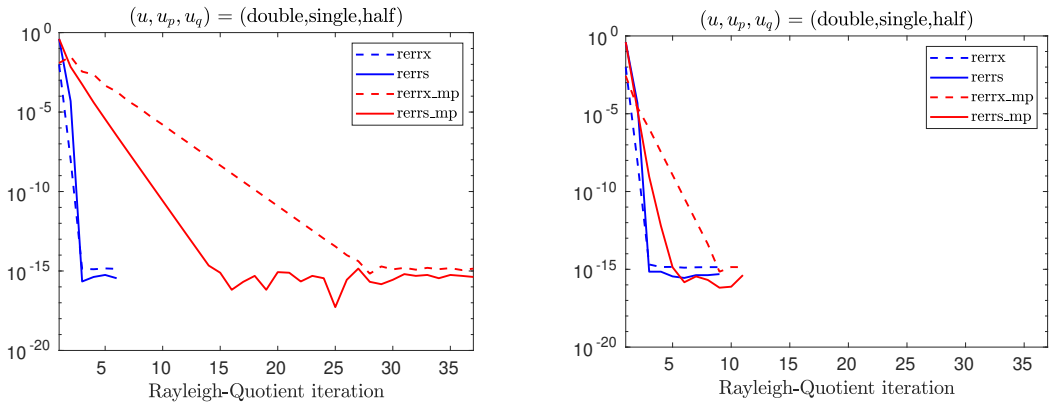

Figure 6.5: Relative errors versus Rayleigh quotient iterations for the Van Huffel matrix with QR (left) and Cholesky factorization with two-sided diagonal scaling (right).

are using half precision in the QR factorization and single precision in PCGTLS, the computation cost is reduced per iteration. Thus, from the performance model we constructed, we see that we can compensate for the additional Rayleigh quotient iterations up to 4× depending on the size of the matrix and how many Rayleigh quotient iterations are performed. Furthermore, our experiments indicate that the use of low precision does not impact the final attainable accuracy of the TLS solution.

Our future work involves a complete mathematical analysis, which we suspect can be done by adapting the results on inexact Newton in Higham and Mary [2022], Tisseur [2001], Kelley [2022] and inexact RQI in Freitag and Spence [2007], Simoncini and Eldén [2002]. With the complete analysis, we will be able to prove our observations on the attainable accuracy and provide a theoretical explanation of different convergence delays on different systems. Furthermore, it would be of great interest to compare the practical run-time of both algorithms on large-scale matrices in a high-performance setting with hardware that supports half precision computation.

Bibliography

Ahmad Abdelfattah, Hartwig Anzt, Erik Boman, Erin Carson, Terry Cojean, Jack Dongarra, Alyson Fox, Mark Gates, Nicholas Higham, Xiaoye Li, Jennifer Loe, Piotr Luszczek, Srikara Pranesh, Siva Rajamanickam, Tobias Ribizel, Barry Smith, Kasia Swirydowicz, Stephen Thomas, Stanimire Tomov, and Ulrike Yang. A survey of numerical linear algebra methods utilizing mixed-precision arithmetic. *The International Journal of High Performance Computing Applications*, page 109434202110033, 03 2021. doi: 10.1177/10943420211003313.

Advanpix LLC. Multiprecision computing toolbox for MATLAB. URL `http://www.advanpix.com/`.

Åke Björck. Iterative refinement of linear least squares solutions i. *BIT Numerical Mathematics*, 7(4):257–278, 1967.

Åke Björck. Newton and Rayleigh quotient methods for total least squares problems. In *Recent Advances in Total Least Squares Techniques and Errors–in–Variables Modeling: Proceedings of the Second International Workshop on Total Least Squares and Errors–in–Variables Modeling*, pages 149–160, 1997.

Åke Björck, Pinar Heggernes, and Pontus Matstoms. Methods for large scale total least squares problems. *SIAM journal on matrix analysis and applications*, 22 (2):413–429, 2000.

Åke Björck. *Numerical Methods for Least Squares Problems*. Society for Industrial and Applied Mathematics, 1996. doi: 10.1137/1.9781611971484.

Erin Carson, Nicholas J. Higham, and Srikara Pranesh. Three-precision GMRES-based iterative refinement for least squares problems. *SIAM Journal on Scientific Computing*, 42(6):A4063–A4083, 2020. doi: 10.1137/20M1316822.

Michael P. Connolly and Nicholas J. Higham. Probabilistic rounding error analysis of Householder QR factorization. Technical Report 2022.5, February 2022. URL `http://eprints.maths.manchester.ac.uk/2865/`. Revised August 2022.

James Demmel. *On floating point errors in Cholesky*. University of Tennessee. Computer Science Department, 1989.

James Demmel, Yozo Hida, E Jason Riedy, and Xiaoye S Li. Extra-precise iterative refinement for overdetermined least squares problems. *ACM Transactions on Mathematical Software (TOMS)*, 35(4):1–32, 2009.

Huai-An Diao and Yang Sun. Mixed and componentwise condition numbers for a linear function of the solution of the total least squares problem. *Linear Algebra and its Applications*, 544:1–29, 2018. ISSN 0024-3795. doi: https://doi.org/10.1016/j.laa.2018.01.008.

Melina A Freitag and Alastair Spence. Convergence theory for inexact inverse iteration applied to the generalised nonsymmetric eigenproblem. *Electronic Transactions on Numerical Analysis*, 28:40–64, 2007.

Gene H. Golub and Charles F. Van Loan. An analysis of the total least squares problem. *SIAM Journal on Numerical Analysis*, 17(6):883–893, 1980. ISSN 00361429.

Nicholas J Higham. *Accuracy and stability of numerical algorithms*. SIAM, 2002.

Nicholas J Higham and Theo Mary. Mixed precision algorithms in numerical linear algebra. *Acta Numerica*, 31:347–414, 2022.

Nicholas J Higham and Srikara Pranesh. Simulating low precision floating-point arithmetic, mims eprint 2019.4. *Manchester Institute for Mathematical Sciences, The University of Manchester, UK*, 2019.

Nicholas J. Higham and Srikara Pranesh. Exploiting lower precision arithmetic in solving symmetric positive definite linear systems and least squares problems. *SIAM Journal on Scientific Computing*, 43(1):A258–A277, 2021. doi: 10.1137/19M1298263.

Iveta Hnětynková, Martin Plesinger, and Zdenek Strakoš. The core problem within a linear approximation problem $AX \approx B$ with multiple right-hand sides. *SIAM Journal on Matrix Analysis and Applications*, 34(3):917–931, 2013.

CT Kelley. Newton's method in mixed precision. *SIAM Review*, 64(1):191–211, 2022.

Valeria Simoncini and Lars Eldén. Inexact Rayleigh quotient-type methods for eigenvalue computations. *BIT Numerical Mathematics*, 42(1):159–182, 2002.

Daniel B Szyld. Criteria for combining inverse and Rayleigh quotient iteration. *SIAM Journal on Numerical Analysis*, 25(6):1369–1375, 1988.

Françoise Tisseur. Newton's method in floating point arithmetic and iterative refinement of generalized eigenvalue problems. *SIAM Journal on Matrix Analysis and Applications*, 22(4):1038–1057, 2001.

H Trussell. Convergence criteria for iterative restoration methods. *IEEE Transactions on acoustics, speech, and signal processing*, 31(1):129–136, 1983.

Sabine Van Huffel. Iterative algorithms for computing the singular subspace of a matrix associated with its smallest singular values. *Linear algebra and its applications*, 154:675–709, 1991.

Sabine Van Huffel and Joos Vandewalle. *The total least squares problem: computational aspects and analysis*. SIAM, 1991.

J. H. Wilkinson. Modern error analysis. *SIAM Review*, 13(4):548–568, 1971. ISSN 00361445.

Ichitaro Yamazaki, Stanimire Tomov, and Jack Dongarra. Mixed-precision Cholesky QR factorization and its case studies on multicore CPU with multiple GPUs. *SIAM Journal on Scientific Computing*, 37(3):C307–C330, 2015.

7. Mixed-precision GMRES-based iterative refinement with recycling[1]

Iterative refinement (IR) a commonly-used approach for solving nonsingular linear systems $Ax = b$, where $A \in \mathbb{R}^{n\times n}, x, b \in \mathbb{R}^n$ (Wilkinson [1963]). In each refinement step i, the approximate solution $x_{i+1} \in \mathbb{R}^n$ is updated by adding a correction term obtained from the previously computed approximate solution x_i. The initial approximate solution x_0 is usually computed by Gaussian elimination with partial pivoting (GEPP). The computed L and U factors of A are then reused to solve the correction equation $Ad_{i+1} = r_i$, where $r_i = b - Ax_i$ is the residual. Finally, the original approximate solution is refined by the correction term, $x_{i+1} = x_i + d_{i+1}$, and the process continues iteratively until the desired accuracy is achieved.

Recently, mixed precision hardware has become commonplace in supercomputing architecture. This has inspired the development of a new mixed precision benchmark for supercomputers, called HPL-AI, which is based on mixed precision iterative refinement and on which today's top supercomputers exceed exascale performance; see, e.g., HPL-MxP, Kudo et al. [2020], TOP500. Algorithm 1 shows a general IR scheme in a mixed precision setting. The authors in Carson and Higham [2018] used three hardware precisions in the algorithm: u_f is used for LU factorization, u_r for residual calculation, and u (the working precision) for the remaining computations. The authors also introduce a fourth quantity, u_s, which is the effectively precision of the solve (taking on the values of u or u_r depending on the particular solver used in line 4). We assume throughout this work that $u_f \geq u \geq u_r$.

For a given combination of precisions and choice of solver, it is well-understood under which conditions Algorithm 1 will converge and what the limiting accuracy will be. The constraint for convergence is usually stated via a constraint on the infinity-norm condition number of the matrix A; see, e.g., Carson and Higham [2018]. In the case that the computed LU factors are used to solve for the correction in line 4, often referred to as "standard IR" (SIR), then $\kappa_\infty(A) = \|A\|_\infty \|A^{-1}\|_\infty$ must be less than u_f^{-1} in order for convergence to be guaranteed.

To relax this constraint on condition number, the authors of Carson and Higham [2017] and Carson and Higham [2018] devised a mixed precision GMRES-based iterative refinement scheme (GMRES-IR), given in Algorithm 17. In GMRES-IR, the correction equation is solved via left-preconditioned GMRES, where the computed LU factors of A are used as preconditioners. This results in a

[1]This chapter is a pre-copyedited, author-produced version of an article accepted for publication in Institute of Mathematics of the Czech Academy of Sciences: Programs and Algorithms of Numerical Mathematics, Programs and Algorithms of Numerical Mathematics. Proceedings of Seminar following peer review. The version of record [Institute of Mathematics of the Czech Academy of Sciences: Programs and Algorithms of Numerical Mathematics. Proceedings of Seminar, Oktay, E., Carson, E.: Mixed precision GMRES-based iterative refinement with recycling] is available online at `https://dml.cz/handle/10338.dmlcz/703196`.

looser constraint on condition number in order to guarantee the convergence of forward and backward errors; in the case that the preconditioned matrix is applied to a vector in each iteration of GMRES in double the working precision, we require $\kappa_\infty(A) \leq u^{-1/2}u_f^{-1}$, and in the case that a uniform precision is used within GMRES, we require $\kappa_\infty(A) \leq u^{-1/3}u_f^{-2/3}$; see Amestoy et al. [2021]. If these constraints are met, we are guaranteed that the preconditioned GMRES method will converge to a backward stable solution after n iterations and that the iterative refinement scheme will converge to its limiting accuracy. We note that in order to guarantee backward stability, all existing analyses (e.g., Carson and Higham [2017, 2018], Amestoy et al. [2021]) assume that unrestarted GMRES is used within GMRES-IR.

Algorithm 17 GMRES-IR (Carson and Higham [2017])

Input: matrix $A^{n\times n}$; right-hand side b^n; maximum number of refinement steps i_{max}; GMRES convergence tolerance τ.

Output: Approximate solution $\hat{x}$ to $Ax = b$.

1: Compute LU factorization $A = LU$ in precision u_f
2: Solve $Ax_0 = b$ by substitution in precision u_f, store in precision u
3: **for** $i = 0$: $i_{max} - 1$ **do**
4: Compute $r_i = b - Ax_i$ in precision u_r, store in precision u
5: Scale $r_i = r_i/\|r_i\|_\infty$
6: Solve $U^{-1}L^{-1}Ad_{i+1} = U^{-1}L^{-1}r_i$ by GMRES in working precision u, with matrix-vector products with $\tilde{A} = U^{-1}L^{-1}A$ computed at precision u^2; store in precision u
7: Compute $x_{i+1} = x_i + d_{i+1}$ in precision u
8: **if** converged **then** return x_{i+1}. **end if**

We also note that existing analyses are unable to say anything about how fast GMRES will convergence in each refinement step, only that it will do so within n iterations. However, if indeed n iterations are required to converge in each GMRES solve, this can make GMRES-IR more expensive than simply computing the LU factorization in higher precision and using SIR. Unfortunately, GMRES convergence is incredibly difficult to predict. In fact, for any set of prescribed eigenvalues, one can construct a linear system for which GMRES will stagnate entirely until the nth iteration (Greenbaum et al. [1996]). The situation is better understood at least in the case of normal matrices, see, e.g., Liesen and Tichý [2004]. The worst-case scenario in the case of normal A is when eigenvalues are clustered near the origin, which can cause complete stagnation of GMRES (Liesen and Tichý [2004]). After the preconditioning step in GMRES-IR, all eigenvalues of the preconditioned matrix ($U^{-1}L^{-1}A$) ideally become 1 in the absence of finite precision error in computing LU and within GMRES. However, in practice, since we have inexact LU factors, if A has a cluster of eigenvalues near the origin, this imperfect preconditioner may fail to shift some of them away from the origin, which can cause GMRES to stagnate. For instance, when random dense matrices having geometrically distributed singular values are used in the multistage iterative refinement algorithm devised in Oktay and Carson [2022], the authors showed that for relatively large condition numbers relative to precision u_f, GMRES tends to perform n iterations in each refinement step.

Figure 7.1 shows the eigenvalue distribution of a double-precision 100×100 random dense matrix having geometrically distributed singular values with condition number $\kappa_2(A) = 10^{12}$, generated in MATLAB via the command `gallery('randsvd',100,1e12,3)`. In the unpreconditioned case (upper left), it is seen that the eigenvalues are clustered around the origin, which is a known difficult case for GMRES. When double-precision LU factors are used for preconditioning (upper right), it is observed that the eigenvalues of the preconditioned system are now clustered around 1. On the other hand, using half-precision LU factors as preconditioners (lower plot) causes a cluster of eigenvalues to remain near the origin, indicating that GMRES convergence will likely be slow (we note that these are nonnormal matrices and so the theory of Liesen and Tichý [2004] does not apply, but our experimental evidence indicates that this is the case).

It is thus clear even in the case that low-precision LU factors can theoretically be used within GMRES-IR, they may not be the best choice from a performance perspective. In this scenario, we are left with two options: either increase the precision in which the LU factors are computed, or seek to improve the convergence behavior of GMRES through other means. It is the latter approach that we take in this work, in particular, we investigate the use of Krylov subspace recycling.

In Section 7.1, we give a background on the use of recycling in Krylov subspace methods. In Section 7.2 we detail our implementation and experimental setup. Extensive numerical experiments that demonstrate the potential benefit of recycling within GMRES-based iterative refinement are presented in Section 7.3. We outline future work and open problems in Section 7.4.

7.1 Krylov subspace recycling

One way to speed up the convergence of the GMRES solver is using recycling (Parks et al. [2006], Soodhalter et al. [2020]). The idea of recycling is that if we have a sequence of linear systems ($Ax_1 = b_1, Ax_2 = b_2, \ldots$) to solve and they all use the same (or a similar) coefficient matrix A, then we can reuse the Krylov subspace information generated in solving $Ax_1 = b_1$ to speed up converge of the method in solving $Ax_2 = b_2$, etc. This is exactly the situation we have in GMRES-IR: within the iterative refinement loop, we call GMRES on the matrix A many times, and only the right-hand side changes between refinement steps. Thus we can use Krylov subspace recycling within GMRES across iterative refinement steps, and theoretically the convergence of GMRES should improve as the refinement proceeds.

GMRES-DR (Morgan [2002]) is a truncated and restarted solver developed for solving single nonsymmetric linear systems. The method deflates small eigenvalues for the new subspace to improve the convergence of restarted GMRES. At the end of each cycle of GMRES-DR, the desired number k of harmonic Ritz values are computed, and an approximate invariant subspace associated with these values is recycled. One of the disadvantages of GMRES-DR is that it is not an adaptive method.

Another truncated solver used for recycling is called GCROT (De Sturler [1999]). The difference between GCROT and GMRES-DR is that GCROT is a truncated minimum residual method. In other words, it recycles a subspace that minimizes the loss of orthogonality with Krylov subspace from the previ-

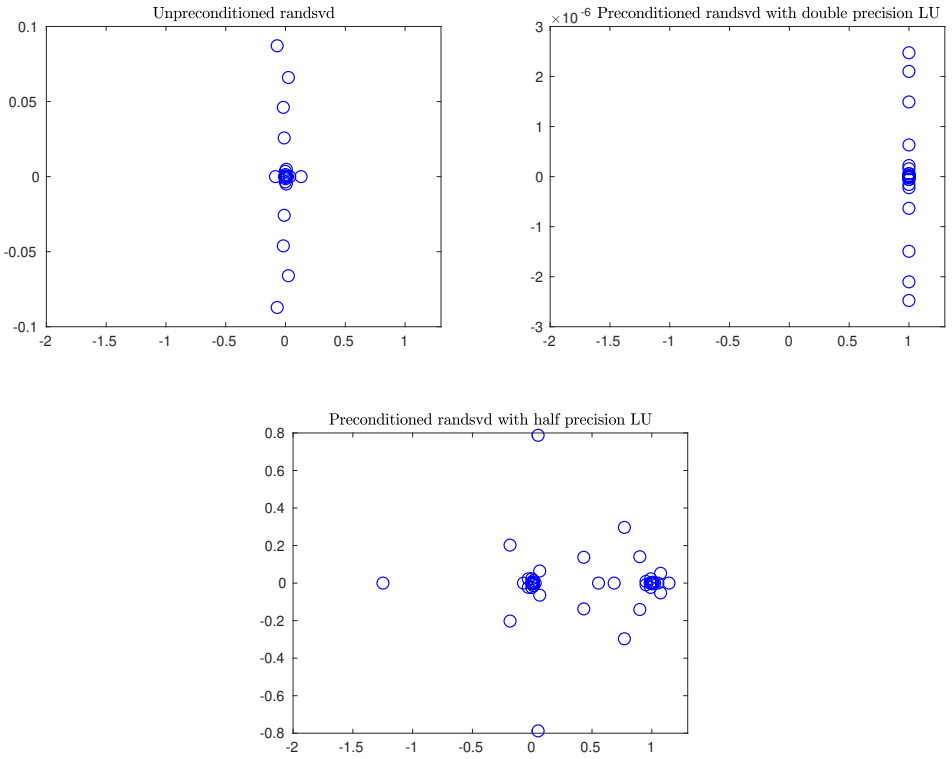

Figure 7.1: Eigenvalue distribution of a double-precision random dense matrix with $\kappa_2(A) = 10^{12}$ without preconditioner (top left), with a double-precision LU preconditioner (top right), and with a half-precision LU preconditioner (bottom).

ous system. Like GMRES-DR, this method is also not able to be adapted for recycling.

Non-adaptive methods compute the recycling space periodically and replace it by starting from the beginning. If A changes continuously, this is a problem since non-adaptive methods cannot adapt to gradual changes in matrices. To use truncation with adaptive recycling, the authors in Parks et al. [2006] modified GCROT and combined it with GMRES-DR to obtain a cheaper and more effective adaptive truncated recycling method called GCRO-DR. In this method, the residual minimization and orthogonalization are performed over the recycled subspace. This allows the algorithm to obtain sufficiently good approximations by continuously updating the approximation to the invariant subspace, and thus it requires fewer iterations per system.

The GCRO-DR algorithm uses the deflated restarting idea in GMRES-DR in the same manner as GCROT. First, using the k harmonic Ritz vectors $\widetilde{Y}_k$ corresponding to the k smallest harmonic Ritz values, matrices $U_k, C_k \in \mathbb{C}^{m\times k}$ are constructed such that $A^{(i+1)}U_k = C_k$ and $C_k^H C_k = I_k$ hold. After finding the optimal solution over $range(U_k)$ and computing the residual, GCRO-DR constructs the Arnoldi relation by generating a Krylov subspace of dimension $m-k+1$ with $(I - C_k C_k^H)A$. After completing the Arnoldi process, the algorithm solves a minimization problem at the end of each cycle, which reduces to an $(m+1)\times m$ least-squares problem. After solving the least-squares problem and computing the residual, a generalized eigenvalue problem is solved, and harmonic Ritz vec-

tors are recovered. Since harmonic Ritz vectors are constructed differently than in GMRES-DR, GCRO-DR is suitable for solving individual linear systems and sequences of them. The GCRO-DR algorithm is given in Algorithm 18.

Algorithm 18 GCRO-DR (Parks et al. [2006])

Input: matrix $A^{n\times n}$; right-hand side b^n; the maximum size m of the subspace; the desired number k of harmonic Ritz vectors; initial guess x_0; GMRES convergence tolerance τ.

Output: Approximate solution x_i to $Ax = b$.

1: **if** $\widetilde{Y}_k$ is known **then**
2: Compute the reduced QR factorization $A\widetilde{Y_k} = QR$.
3: Compute $C_k = Q$ and $U_k = \widetilde{Y}_k R^{-1}$.
4: Compute $x_1 = x_0 + U_k C_k^H r_0$ and $r_1 = r_0 - C_k C_k^H r_0$.
5: **else**
6: Compute $v_1 = r_0/\|r_0\|_2$ and $c = \|r_0\|_2 e_1$.
7: Perform m steps of GMRES, solving $\min\|c - \underline{H}_m y\|_2$ for y and generating V_{m+1} and $\underline{H}_m$.
8: Compute $x_1 = x_0 + V_m y$ and $r_1 = V_{m+1}(c - \underline{H}_m y)$.
9: Compute the k eigenvectors $\widetilde{z}_j$ of $(H_m + h_{m+1,m}^2 H_m^{-H} e_m e_m^H)\widetilde{z}_j = \widetilde{\theta}_j \widetilde{z}_j$ associated with the smallest magnitude eigenvalues $\widetilde{\theta}_j$ and store in P_k.
10: Compute $\widetilde{Y}_k = V_m P_k$.
11: Compute the reduced QR factorization $\underline{H}_m P_k = QR$.
12: Compute $C_k = V_{m+1} Q$ and $U_k = \widetilde{Y}_k R^{-1}$.
13: **while** $\|r_i\|_2 > \tau$ **do**
14: Perform $m-k$ Arnoldi steps with $(I - C_k C_k^H)A$, letting $v_1 = r_{i-1}/\|r_{i-1}\|^2$ and generating $V_{m-k+1}, \underline{H}_{m-k}$, and B_{m-k}.
15: Compute $\widetilde{U}_k = U_k D_k$.
16: Define $\widehat{V}_m = [\widetilde{U}_k \quad V_{m-k}]$, $\widehat{W}_{m+1} = [C_k \quad V_{m-k+1}]$, and $\underline{G}_m = \begin{bmatrix} D_k & B_{m-k} \\ 0 & \underline{H}_{m-k} \end{bmatrix}$.
17: Solve $\min\|\widehat{W}_{m+1}^H r_{i-1} - \underline{G}_m y\|_2$ for y.
18: Compute $x_i = x_{i-1} + \widehat{V}_m y$ and $r_i = r_{i-1} - \widehat{W}_{m+1}\underline{G}_m y$.
19: Compute the k eigenvectors $\widetilde{z}_i$ of $\underline{G}_m^H \underline{G}_m \widetilde{z}_i = \widetilde{\theta}_i \underline{G}_m^H \widehat{W}_{m+1}^H \widehat{V}_m \widetilde{z}_i$ associated with the smallest magnitude eigenvalues $\widetilde{\theta}_i$ and store in P_k.
20: Compute $\widetilde{Y}_k = v_m P_k$.
21: Compute the reduced QR factorization $\underline{G}_m P_k = QR$.
22: Compute $C_k = W_{m+1} Q$ and $U_k = \widetilde{Y}_k R^{-1}$.
23: Update $\widetilde{Y}_k = U_k$ for the next system.

The use of recycling may also be favorable from a performance perspective. In line 14 of Algorithm 18, GCRO-DR performs only $m - k$ Arnoldi steps implying that it performs $m - k$ matrix-vector multiplications per cycle, whereas GMRES(m) performs m matrix-vector multiplications. It is also mentioned in Parks et al. [2006] that since GCRO-DR stores U_k and C_k, it performs $2kn(1+k)$ fewer operations during the Arnoldi process. On the other hand, since we are using k eigenvectors, GCRO-DR(m, k) requires storing k more vectors than GMRES(m). Moreover, after the first cycle, in line 2, GCRO-DR requires computing the QR factorization of $A\widetilde{Y}_k$ before the GCROT step. Although choosing

the number k can be considered a trade-off between memory and performance, performing $2kn(1+k)$ fewer operations in the Arnoldi stage can provide a performance improvement.

7.2 Implementation and experimental setup

In an effort to reduce the overall computational cost of the GMRES solves within GMRES-IR, we develop a recycled GMRES-based iterative refinement algorithm, called RGMRES-IR, presented in Algorithm 19. The algorithm starts with computing the LU factors of A and computing the initial approximate solution in the same manner as GMRES-IR. Instead of preconditioned GMRES however, the algorithm uses preconditioned GCRO-DR(m,k) to solve the correction equation. Similar to GMRES-IR, our presentation of RGMRES-IR in Algorithm 19 also uses an extra u^2 precision in preconditioning, although in practice one could use a uniform precision within GCRO-DR(m,k).

Algorithm 19 GMRES-based Iterative Refinement with Recycling (RGMRES-IR)

Input: matrix $A^{n\times n}$; right-hand side b^n; maximum number of refinement steps i_{max}; GMRES convergence tolerance τ; the maximum size m of the subspace; the desired number k of harmonic Ritz vectors.

Output: Approximate solution x_{i+1} to $Ax=b$.

1: Compute LU factorization $A \approx LU$ in precision u_f
2: Solve $Ax_0 = b$ by substitution in precision u_f, store in precision u
3: **for** $i = 0 : i_{max} - 1$ **do**
4: Compute $r_i = b - Ax_i$ in precision u_r, store in precision u
5: Scale $r_i = r_i/\|r_i\|_\infty$
6: Solve $U^{-1}L^{-1}Ad_{i+1} = U^{-1}L^{-1}r_i$ by GCRO-DR(m,k) with tolerance τ in precision u, with matrix-vector products with $\widetilde{A} = U^{-1}L^{-1}A$ computed in precision u^2, store in precision u.
7: Compute $x_{i+1} = x_i + \|r_i\|_\infty d_{i+1}$ in precision u
8: **if** converged **then** return x_{i+1}. **end if**

In the RGMRES-IR algorithm, as in GMRES-IR, we use three precisions: u_f is the factorization precision in which the factorization of A is computed, u is the working precision in which the input data A and b and the solution x are stored, and u_r is the precision used to compute the residuals. Again we note that we assume $u_r \leq u \leq u_f$. For the implementation, we adapted the MATLAB implementations of the GMRES-IR method developed in Carson and Higham [2018], and the GCRO-DR method developed in Parks et al. [2006]. To simulate half-precision floating-point arithmetic, we use the `chop` library and associated functions from Higham and Pranesh [2019], available at `https://github.com/higham/chop` and `https://github.com/SrikaraPranesh/LowPrecision_Simulation`. For single and double precision, we use the MATLAB built-in data types and to simulate quadruple precision we use the Advanpix multiprecision computing toolbox Advanpix LLC..

Using different precision settings results in different constraints on the condition number to guarantee convergence of the forward and backward errors. Al-

though our experiments here use three different precisions, two precisions (only computing residuals in higher precision) or fixed (uniform) precision can also be used in the RGMRES-IR algorithm. We also restrict ourselves to IEEE precisions (see Table 1), although we note that one could also use formats like bfloat16 (Intel Corporation [2018]). Table 7.1 summarizes the error bounds for GMRES-IR with various choices of precisions. For convergence of GMRES-IR, $\kappa_\infty(A)$ should be less than the values shown in the fourth column. For detailed information about GMRES-IR, see Carson and Higham [2018].

Table 7.1: Choices of IEEE standard precisions for three-precision GMRES-IR presented in Carson and Higham [2018], and their convergence conditions.

u_f	u	u_r	$\kappa_\infty(A)$	Backward error		
				Normwise	Componentwise	Forward error
half	half	single	10^4	half	half	half
half	single	double	10^8	single	single	single
half	double	quad	10^{12}	double	double	double
single	single	double	10^8	single	single	single
single	double	quad	10^{16}	double	double	double

We experiment with three categories of test matrices. We will first test our algorithm on random dense matrices of dimension $n = 100$ having geometrically distributed singular values. We generated the matrices using the MATLAB command `gallery('randsvd',n,kappa(i),3)`, where `kappa` is the array of the desired 2-norm condition numbers $\kappa_2(A) = \{10^4, 10^5, 10^6, 10^7, 10^8, 10^9, 10^{10}, 10^{11}, 10^{12}, 10^{13}\}$, and `3` stands for the mode for singular value distribution of the matrices. Mode 3 corresponds to the matrix having geometrically distributed singular values. For reproducibility, we use the MATLAB command `rng(1)` each time we run the algorithm. We will present the numerical results in Section 7.3.3.

As shown in Figure 7.1, these matrices have eigenvalues clustered around the origin, which can be a difficult case for GMRES convergence. This class of artificial problems thus represents a good use case for the RGMRES-IR algorithm. We note however that the recycling approach can improve the convergence of GMRES even when the eigenvalues are not clustered around the origin. To illustrate this, we also test our algorithm on real matrices. All matrices shown in Table 7.2 are taken from the SuiteSparse Collection (Davis and Hu [2011]). The numerical results are shown in Section 7.3.2.

Besides the matrices mentioned above, we also test our algorithm on prolate (symmetric, ill-conditioned Toeplitz matrices whose eigenvalues are distinct, lie in the interval $(0, 1)$, and tend to cluster around 0 and 1) matrices (Varah [1993]) of dimension $n = 100$. We generated the matrices using the MATLAB command

Table 7.2: Matrices from Davis and Hu [2011] used for numerical experiments and their properties.

Name	Size	nnz	$\kappa_\infty(A)$	Group	Kind
orsirr_1	1030	1030	9.96E+04	HB	Computational Fluid Dynamics Problem
comsol	1500	1500	3.42E+06	Langemyr	Structural Problem
circuit204	1020	1020	9.03E+09	Yzhou	Circuit Simulation Problem

Table 7.3: Prolate matrices used for numerical experiments and their properties.

α	$\kappa_\infty(A)$	$\kappa_2(A)$
0.475	1.21E+06	3.60E+05
0.47	2.63E+07	7.60E+06
0.467	1.68E+08	4.79E+07
0.455	2.91E+11	8.04E+10
0.45	6.64E+12	1.82E+12
0.4468	4.98E+13	1.35E+13
0.44	3.41E+15	9.07E+14
0.434	3.44E+16	8.84E+15

`gallery('prolate',n,alpha)`, where `alpha` is the array of the desired parameters $\alpha = \{0.475, 0.47, 0.467, 0.455, 0.45, 0.4468, 0.44, 0.434\}$. When $\alpha < 0.5$ is chosen to be small, it becomes difficult for GMRES-IR to solve the system since the eigenvalues skewed more towards zero. The properties of prolate matrices used in this study are shown in Table 7.3. The numerical results are shown in Section 7.3.1.

For a fair comparison between GMRES-IR and RGMRES-IR, GMRES-IR is used with restart value m, which is the maximum size of the recycled space used in RGMRES-IR. Since the first refinement step of RGMRES-IR does not have a recycled subspace, it is the same as the first step of GMRES-IR. We thus expect a decrease in the number of GMRES iterations per refinement step starting from the second refinement step.

The experiments are performed on a computer with Intel Core i7-9750H having 12 CPUs and 16 GB RAM with OS system Ubuntu 20.04.1. Our RGMRES-IR algorithm and associated functions are available through the repository `https://github.com/edoktay/rgmresir`, which includes scripts for generating the data and plots in this work.

7.3 Numerical experiments

In this section, we present the numerical results comparing GMRES-IR and RGMRES-IR for solving $Ax = b$. To ensure that we fully exhibit the behavior of the methods, we set $i_{max} = 10000$, which is large enough to allow all approaches that eventually converge sufficient time to do so. The GMRES convergence tolerance τ, which appears both in Algorithms 17 and 19, dictates the stopping criterion for the inner GMRES iterations. The algorithm is considered converged if the relative (preconditioned) residual norm drops below τ. In tests here with single working precision, we use $\tau = 10^{-4}$. For double working precision, we use $\tau = 10^{-8}$. The results are compared in two different metrics: The number of GMRES iterations per refinement step and the total number of GMRES iterations. The number of steps and iterations are shown in tables.

In this study, $m \leq n$ is chosen such that the number of GMRES iterations in the first step of the iterative refinement method is smaller than m. To reduce the total number of GMRES steps in RGMRES-IR even further than the results presented in this study, one can also test RGMRES-IR with various $m \leq n$ values. However, one should note that there may be a trade-off

between the total number of GMRES iterations and the number of refinement steps. For RGMRES-IR, the optimal number $k < m$ of harmonic Ritz vectors is chosen for each group of matrices with the desired precision settings after several experiments on various (m, k) scenarios. The optimum k differs for each matrix. The least total number of GMRES iterations is obtained for $k =$ (the number of GMRES iterations in the first refinement step) $- 1$ since, in this case, we are recycling the whole generated subspace, which is expensive. That is why one should choose a k value as small as possible to reduce computational cost while benefiting from recycling. Figure 7.2 shows the change in the total GMRES iterations according to the given k values for two matrices. From the plots, one can easily *find the knee*, i.e., find the optimum k value that uses the smallest number of GMRES iterations.

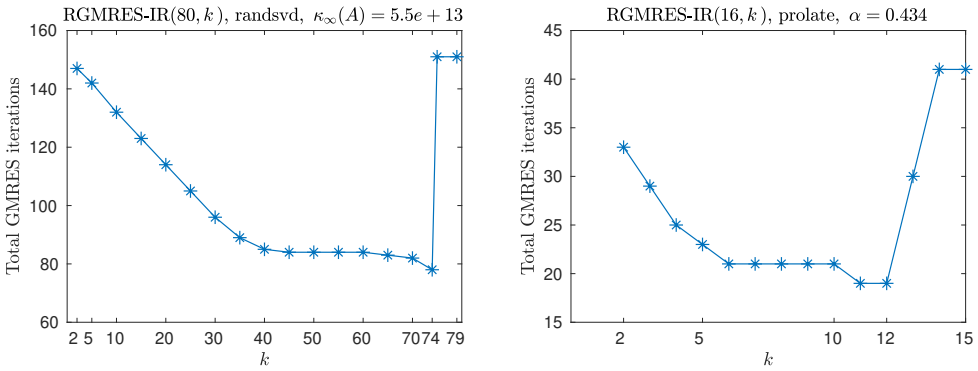

Figure 7.2: Total GMRES iterations for various k for a randsvd matrix with $\kappa_2(A) = 10^{13}$ (left) and a prolate matrix with $\alpha = 0.434$ (right) for $(u_f, u, u_r) =$ (single, double, quad).

For simplicity, b is chosen to be the vector of ones for all matrices, and the precisions are chosen such that $u \leq u_f^2$, and $u_r \leq u^2$. For each random dense matrix and each matrix in Table 7.2 and Table 7.3, $Ax = b$ is solved by using precisions $(u_f, u, u_r) =$ (single, double, quad), $(u_f, u, u_r) =$ (half, single, double), and $(u_f, u, u_r) =$ (half, double, quad).

In the tables we will present, the first number shows the total number of GMRES iterations. The numbers in the parentheses indicate the number of GMRES iterations performed in each refinement step. For instance, 5 (2,3) implies that at the first step of GMRES-IR, 2 GMRES iterations are performed, respectively, while for the second step, 3 GMRES iterations are performed, which gives a total of 5 GMRES iterations.

Since all of the numerical experiments in this study were performed in MATLAB, it does not include performance analysis. Performance results for mixed precision GMRES-based iterative refinement with restarting were recently presented in Lindquist et al. [2022].

7.3.1 Prolate matrices

For the prolate matrices generated as described in Section 7.2 with `alpha` parameters $\alpha =$ {0.475, 0.47, 0.467, 0.455, 0.45, 0.4468, 0.44, 0.434}, RGMRES-IR is used

with precisions (u_f, u, u_r) = (single, double, quad) and (u_f, u, u_r) = (half, single, double). Tables 7.4 and 7.5 show the number of GMRES iterations performed by GMRES-IR and RGMRES-IR with different precision settings.

Table 7.4 shows the data for experiments with (u_f, u, u_r) = (single, double, quad). In the table, we can see that for $\alpha > 0.46$, recycling does not affect the performance of GMRES for $(m, k) = (16, 4)$ since only a small number of GMRES iterations are performed in each step. As α decreases, however, the convergence of GMRES slows down; hence recycling starts to more significantly decrease the total number of GMRES iterations.

Table 7.4: Number of GMRES-IR and RGMRES-IR refinement steps and the number of GMRES iterations for each refinement step for prolate matrices with various α values, using precisions (u_f, u, u_r) = (single, double, quad) and (m, k) = (16,4).

α	GMRES-IR (16)	RGMRES-IR (16,4)
0.475	5 (2,3)	5 (2,3)
0.47	5 (2,3)	5 (2,3)
0.467	7 (3,4)	7 (3,4)
0.455	13 (6,7)	8 (6,2)
0.45	15 (7,8)	11 (7,4)
0.4468	25 (7,9,9)	15 (7,4,4)
0.44	34 (10,12,12)	19 (10,5,4)
0.434	41 (13,14,14)	25 (13,6,6)

In the case of (u_f, u, u_r) = (half, single, double), presented in Table 7.5, we can see that GMRES-IR diverges for $\alpha < 0.45$ with and without recycling. However, when $\alpha = 0.45$, we see that RGMRES-IR diverges although GMRES-IR converges. This is because of the multiple periods of stagnation in the second refinement step due to recycling. GMRES cannot converge in the first $16-5 = 11$ iterations in the second step, causing an infinite restart which results in divergence. However, for cases where both GMRES-IR and RGMRES-IR convergence, RGMRES-IR always requires fewer total GMRES iterations. For $\alpha = 0.455$, GMRES restarts in the second refinement step of GMRES-IR, while, because of recycling, RGMRES-IR converges without restarting, which decreases the computational cost.

7.3.2 SuiteSparse matrices

For matrices in Table 7.2, Tables 7.6 and 7.7 compare the performance of GMRES-IR and RGMRES-IR for precisions (u_f, u, u_r) = (half, single, double) and (u_f, u, u_r) = (half, double, quad), respectively. In the (u_f, u, u_r) = (half, single, double) scenario, for the orsirr_1 matrix, GMRES already converges quite quickly, and so recycling does not have an effect. For the matrices comsol and circuit204, however, RGMRES-IR reduces the total number of GMRES iterations required by 35% and 28%, respectively. RGMRES-IR has a benefit for all matrices when using (u_f, u, u_r) = (half, double, quad).

Table 7.5: Number of GMRES-IR and RGMRES-IR refinement steps with the number of GMRES iterations for each refinement step for prolate matrices with various α values, using precisions (u_f, u, u_r) = (half,single,double) and (m, k) = (16,5).

α	GMRES-IR (16)	RGMRES-IR (16,5)
0.475	12 (6,6)	8 (6,2)
0.47	16 (8,8)	10 (8,2)
0.467	19 (9,10)	11 (9,2)
0.455	50 (15,25,10)	19 (15,4)
0.45	89 (14,43,32)	-
0.4468	-	-
0.44	-	-
0.434	-	-

Table 7.6: Number of GMRES-IR and RGMRES-IR refinement steps with the number of GMRES iterations for each refinement step for real matrices, using precisions (u_f, u, u_r) = (half,single,double) and (m, k) = (40,6).

Matrix	GMRES-IR (40)	RGMRES-IR(40,6)
orsirr_1	12 (6,6)	12 (6,6)
comsol	46 (22,24)	30 (22,8)
circuit204	40 (12,14,14)	29 (12,9,8)

Table 7.7: Number of GMRES-IR and RGMRES-IR refinement steps with the number of GMRES iterations for each refinement step for real matrices, using precisions (u_f, u, u_r) = (half, double, quad) and (m, k) = (40,10).

Matrix	GMRES-IR (40)	RGMRES-IR(40,10)
orsirr_1	22 (11,11)	20 (11,9)
comsol	52 (25,27)	34 (25,9)
circuit204	59 (18,20,21)	47 (18,14,15)

7.3.3 Random dense matrices

For the random dense matrices with geometrically distributed singular values described in Section 7.2, we compared methods using precisions (u_f, u, u_r) = (single, double, quad), (u_f, u, u_r) = (half, single, double), and (u_f, u, u_r) = (half, double, quad).

SGMRES-IR versus RSGMRES-IR

In practice, it is common that implementations use a uniform precision within GMRES (i.e., applying the preconditioned matrix to a vector in precision u rather than u^2. This is beneficial from a performance perspective (in particular if precision u^2 must be implemented in software). The cost is that the constraint on condition numbers for which the refinement scheme is guaranteed to converge becomes tighter. To illustrate the benefit of recycling in this scenario, we first compare what we call SGMRES-IR (which is GMRES-IR but with a uniform precision within GMRES) to the recycled version, RSGMRES-IR. For a fair

comparison, restarted SGMRES-IR (SGMRES-IR(m)) is compared with recycled SGMRES-IR (RSGMRES-IR(m, k)).

Table 7.8 shows the number of GMRES iterations performed by SGMRES-IR and RSGMRES-IR in the (u_f, u, u_r) = (single, double, quad) setting. From the table, we observe that recycling reduces the number of GMRES iterations in this case as well. The reason why SGMRES-IR does not converge for $\kappa_2(A) \geq 10^{14}$ is that in the first refinement step, restarted SGMRES does not converge (restarting an infinite number of times). For RSGMRES-IR, in the first GCRO-DR call, the recycling after the first restart cycle helps, allowing GCRO-DR to converge. We note that this is another benefit of the recycling approach, as it can improve the reliability of restarted GMRES, which is almost always used in practice.

Table 7.8: Number of SGMRES-IR and RSGMRES-IR refinement steps with the number of GMRES iterations for each refinement step for precisions (u_f, u, u_r) = (single, double, quad) and (m, k) = (80,18). For $\kappa_2(A) = 10^{15}$, RSGMRES-IR required 2093 (139, 60, 59, 60, 59, 59, 60, 60, 60, 60, 60, 60, 60, 60, 60, 60, 60, 48, 46, 46, 51, 60, 48, 48, 48, 51, 48, 48, 48, 58, 48, 48, 48, 48, 48, 48, 61) iterations.

$\kappa_\infty(A)$	$\kappa_2(A)$	SGMRES-IR (80)	RSGMRES-IR (80,18)
$6 \cdot 10^9$	10^9	64 (19,23,22)	34 (19,8,7)
$6 \cdot 10^{10}$	10^{10}	120 (39,40,41)	65 (39,13,13)
$6 \cdot 10^{11}$	10^{11}	160 (52,54,54)	94 (52,21,21)
$6 \cdot 10^{12}$	10^{12}	196 (65,65,66)	163 (65,32,32,34)
$5 \cdot 10^{13}$	10^{13}	301 (75,75,75,76)	199 (75,41,41,42)
$5 \cdot 10^{14}$	10^{14}	-	493 (131,51,51,52,52,52,52,52)
$5 \cdot 10^{15}$	10^{15}	-	2093*

GMRES-IR versus RGMRES-IR

We now return to our usual setting and compare GMRES-IR and RGMRES-IR for random dense matrices with condition numbers $\kappa_2(A) = \{10^4, 10^5, 10^6, 10^7, 10^8, 10^9, 10^{10}, 10^{11}, 10^{12}, 10^{13}, 10^{14}, 10^{14}\}$. Results using precisions (u_f, u, u_r) = (single, double, quad), (u_f, u, u_r) = (half, single, double), and (u_f, u, u_r) = (half, double, quad) are displayed in Tables 7.9-7.11, respectively.

Table 7.9 shows the numerical experiments with (u_f, u, u_r) = (single, double, quad). In the table, we can see that for relatively well-conditioned matrices ($\kappa_\infty(A) < 10^8$) since the total number of GMRES iterations is small, using recycling does not change the total number of GMRES iterations. When the condition number increases, however, the number of GMRES iterations drops significantly. For $\kappa_2(A) \geq 10^{14}$, GMRES-IR does not converge for the same reason SGMRES-IR did not as described above; namely, that GMRES with the restart parameter 80 does not converge. As before, the recycling between restart cycles fixed this, and RGMRES-IR is still able to converge in this case.

In Table 7.10 we present the experiments for (u_f, u, u_r) = (half, single, double). For $\kappa_\infty(A) > 10^5$, recycling reduces the total number of GMRES iterations. This is also the case in Table 7.11, which shows results for precisions (u_f, u, u_r) = (half, double, quad). This is a known difficult case for GMRES, and thus is a clear case where we can see significant benefit of recycling. We see the most significant

Table 7.9: Number of GMRES-IR and RGMRES-IR refinement steps with the number of GMRES iterations for each refinement step for random dense matrices having geometrically distributed singular values (mode 3) with various condition numbers, using precisions (u_f, u, u_r) = (single, double, quad) and (m, k) = (80,18).

$\kappa_\infty(A)$	$\kappa_2(A)$	GMRES-IR (80)	RGMRES-IR (80, 18)
$9 \cdot 10^4$	10^4	4 (2,2)	4 (2,2)
$8 \cdot 10^5$	10^5	6 (3,3)	6 (3,3)
$7 \cdot 10^6$	10^6	8 (4,4)	8 (4,4)
$7 \cdot 10^7$	10^7	11 (5,6)	11 (5,6)
$7 \cdot 10^8$	10^8	22 (10,12)	22 (10,12)
$6 \cdot 10^9$	10^9	67 (19,24,24)	36 (19,8,9)
$6 \cdot 10^{10}$	10^{10}	80 (39,41)	53 (39,14)
$6 \cdot 10^{11}$	10^{11}	107 (52,55)	75 (52,23)
$6 \cdot 10^{12}$	10^{12}	132 (65,67)	99 (65,34)
$5 \cdot 10^{13}$	10^{13}	151 (75,76)	117 (75,42)
$5 \cdot 10^{14}$	10^{14}	-	184 (131,53)
$5 \cdot 10^{15}$	10^{15}	-	325 (139,62,124)

improvement for the matrix with $\kappa_\infty(A) = 10^{13}$, in which RGMRES-IR requires over 16× fewer GMRES iterations than GMRES-IR. We note that GMRES-IR is only guaranteed to converge up to $\kappa_2(A) < 10^{12}$ for this combination of precisions; see Table 7.1.

Table 7.10: Number of GMRES-IR and RGMRES-IR refinement steps with the number of GMRES iterations for each refinement step for random dense matrices having geometrically distributed singular values (mode 3) with various condition numbers, using precisions (u_f, u, u_r) = (half, single, double) and (m, k) = (90,30).

$\kappa_\infty(A)$	$\kappa_2(A)$	GMRES-IR (90)	RGMRES-IR (90, 30)
$9 \cdot 10^4$	10^4	20 (9,11)	20 (9,11)
$8 \cdot 10^5$	10^5	70 (32,38)	44 (32,12)
$7 \cdot 10^6$	10^6	129 (64,65)	77 (64,13)
$7 \cdot 10^7$	10^7	164 (82,82)	107 (82,25)

The reason that RGMRES-IR outperforms GMRES-IR is different than in the previous cases (caused by stagnation caused by restarting), and is almost accidental in this case. We investigate this more closely in Figure 7.3. In the left plot, we see the convergence trajectory of GMRES(100). In the first restart cycle, the residual decreases from 10^6 to 10^3 after 100 GMRES iterations. GMRES restarts and performs two more iterations, at which point it converges to a relative residual of 10^{-8} (absolute residual of around 10^{-2}. Hence, the first refinement step of GMRES-IR does $100 + 2 = 102$ iterations. The right plot shows the residual trajectory for GCRO-DR. The first restart cycle is the same as in GMRES, however, once the method restarts, the residual stagnates just above the level required to declare convergence. After $m - k = 70$ more iterations, GCRO-DR restarts again, and this time, the residual drops significantly lower. So while GCRO-DR requires more iterations (172) to converge to the specified tolerance, when it does

Table 7.11: Number of GMRES-IR and RGMRES-IR refinement steps with the number of GMRES iterations for each refinement step for random dense matrices having geometrically distributed singular values (mode 3) with various condition numbers, using precisions (u_f, u, u_r) = (half, double, quad) and (m, k) = (100,30). For 10^{13}, GMRES-IR did 3954 (102, 105, 98, 98, 88, 81, 85, 84, 81, 84, 83, 84, 89, 90, 87, 88, 89, 91, 83, 83, 87, 88, 91, 84, 89, 88, 83, 82, 89, 92, 88, 97, 83, 84, 90, 83, 84, 83, 90, 83, 94, 83, 82, 85, 99) iterations.

$\kappa_\infty(A)$	$\kappa_2(A)$	GMRES-IR (100)	RGMRES-IR (100,30)
$9 \cdot 10^4$	10^4	33 (16,17)	33 (16,17)
$8 \cdot 10^5$	10^5	85 (41,44)	71 (41,15,15)
$7 \cdot 10^6$	10^6	134 (66,68)	85 (66,19)
$7 \cdot 10^7$	10^7	167 (83,84)	113 (83,30)
$7 \cdot 10^8$	10^8	193 (96,97)	138 (96,42)
$6 \cdot 10^9$	10^9	200 (100,100)	151 (100,51)
$6 \cdot 10^{10}$	10^{10}	200 (100,100)	158 (100,58)
$6 \cdot 10^{11}$	10^{11}	200 (100,100)	165 (100,65)
$6 \cdot 10^{12}$	10^{12}	200 (100,100)	170 (100,70)
$5 \cdot 10^{13}$	10^{13}	3954*	241 (171,70)

converge, it converges to a solution with a smaller residual. This phenomenon can in turn reduce the total number of refinement steps required. It is possible that we could reduce the overall number of GMRES iterations within GMRES-IR (and also RGMRES-IR) by making the GMRES convergence tolerance τ smaller. We did not experiment with changing the GMRES tolerance within GMRES-IR or RGMRES-IR, but this tradeoff would be interesting to explore in the future.

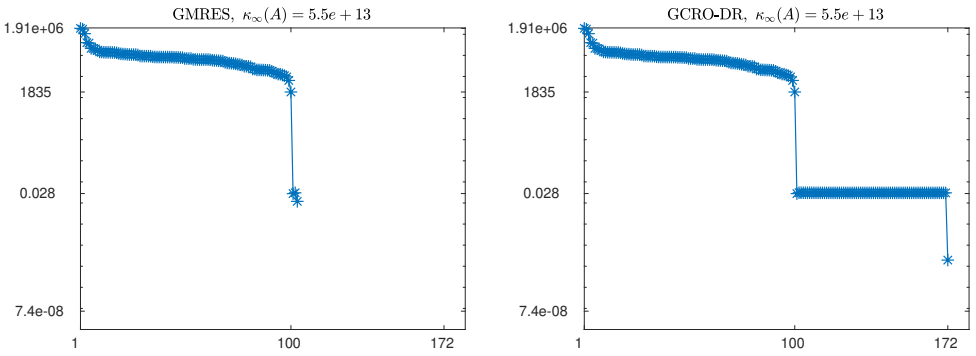

Figure 7.3: Residual trajectory of GMRES (left) and GCRO-DR (right), used within GMRES-IR and RGMRES-IR, respectively, for a randsvd matrix with $\kappa_2(A) = 10^{13}$ and precisions (u_f, u, u_r) = (half, double, quad).

We stress that the convergence guarantees for GMRES-IR for various precisions stated in Carson and Higham [2017, 2018], Amestoy et al. [2021] hold only for the case of unrestarted GMRES, i.e., $m = n$. When $m < n$, there is no guarantee that GMRES converges to a backward stable solution and thus no guarantee that GMRES-IR will converge. Choosing a restart parameter m that allows for convergence is a difficult problem, and a full theory regarding

Table 7.12: Number of GMRES-IR and RGMRES-IR refinement steps with the number of GMRES iterations for each refinement step for random dense matrices having geometrically distributed singular values (mode 3) with various condition numbers, using precisions (u_f, u, u_r) = (half, double, quad) and (m, k) = (90,40).

$\kappa_\infty(A)$	$\kappa_2(A)$	GMRES-IR (90)	RGMRES-IR (90,40)
$9 \cdot 10^4$	10^4	33 (16,17)	33 (16,17)
$8 \cdot 10^5$	10^5	85 (41,44)	50 (41,9)
$7 \cdot 10^6$	10^6	134 (66,68)	81 (66,15)
$7 \cdot 10^7$	10^7	167 (83,84)	100 (83,17)
$7 \cdot 10^8$	10^8	-	149 (119,30)
$6 \cdot 10^9$	10^9	-	179 (134,45)
$6 \cdot 10^{10}$	10^{10}	-	470 (388,41,41)
$6 \cdot 10^{11}$	10^{11}	-	-
$6 \cdot 10^{12}$	10^{12}	-	-
$5 \cdot 10^{13}$	10^{13}	-	-

the behavior of restarted GMRES is lacking. The behavior of restarted GMRES is often unintuitive; whereas one would think that a larger restart parameter is likely to be better than a smaller one as it is closer to unrestarted GMRES, this is not always the case. In Embree [2003], the author gave examples where a larger restart parameter causes complete stagnation, whereas a smaller one results in fast convergence.

Whereas in Table 7.12 we now illustrates the behavior for the same problems and same precisions (u_f, u, u_r) = (half, double, quad) for the case $m < n$. It is seen from the table that both methods converge for $\kappa_\infty(A) < 10^8$. After this point, GMRES-IR does not converge, whereas RGMRES-IR does. This serves as an example where the convergence guarantees given in Carson and Higham [2017, 2018] do not hold for GMRES-IR with restarted GMRES; for unrestarted GMRES, convergence is guaranteed up to $\kappa_\infty(A) \leq 10^{12}$ for this precision setting. Here, GMRES-IR does not converge because of the stagnation caused by restarting in the first refinement step. Aided by the recycling between restart cycles, RGMRES-IR does converge up to $\kappa_2(A) = 10^{11}$, although the large number of GMRES iterations required in the first refinement step makes this approach impractical.

7.4 Conclusion and future work

With the emergence of mixed precision hardware, mixed precision iterative refinement algorithms are the focus of significant renewed interest. A promising approach is the class of GMRES-based refinement schemes, which can enable the accurate solution of extremely ill-conditioned matrices. However, for some matrices, GMRES convergence can be very slow, even when (low-precision) preconditioners are applied. This makes the GMRES-based approaches unattractive from a performance perspective. In this work, incorporate Krylov subspace recycling into the mixed precision GMRES-based iterative refinement algorithm in order to reduce the total number of GMRES iterations required. We call our algorithm RGMRES-IR. Instead of preconditioned GMRES, RGMRES-IR uses

a preconditioned GCRO-DR algorithm to solve for the approximate solution update in each refinement step. Our detailed numerical experiments on random dense matrices, prolate matrices, and matrices from SuiteSparse (Davis and Hu [2011]) show the potential benefit of the recycling approach. Even in cases where the number of GMRES iterations does not preclude the use of GMRES-based iterative refinement, recycling can have a benefit. In particular, it can improve the reliability of restarted GMRES, which is used in most practical scenarios.

One major caveat for GMRES-based iterative refinement schemes is that the analysis and convergence criteria discussed in the literature all rely on the use of unrestarted GMRES. When restarted GMRES is used, we can not give such concrete guarantees, as restarted GMRES may not converge even in infinite precision. A greater understanding of the theoretical behavior of restarted GMRES (and GCRO-DR) both in infinite and finite precision would be of great interest.

Another potential future direction is the exploration of the potential for the use of mixed precision within GCRO-DR. In this work, within GCRO-DR we only used extra precision in applying the preconditioned matrix to a vector, as is done in GMRES-IR. There may however, be further potential for the use of low precision within GCRO-DR, for example, in computation of harmonic Ritz pairs.

Bibliography

Advanpix LLC. Multiprecision computing toolbox for MATLAB. URL `http://www.advanpix.com/`.

Patrick Amestoy, Alfredo Buttari, Nicholas J. Higham, Jean-Yves L'Excellent, Theo Mary, and Bastien Vieublé. Five-precision GMRES-based iterative refinement. Technical Report 2021.5, April 2021. URL `http://eprints.maths.manchester.ac.uk/2807/`.

Erin Carson and Nicholas J. Higham. A new analysis of iterative refinement and its application to accurate solution of ill-conditioned sparse linear systems. *SIAM Journal on Scientific Computing*, 39(6):A2834–A2856, 2017. doi: 10.1137/17M1122918.

Erin Carson and Nicholas J. Higham. Accelerating the solution of linear systems by iterative refinement in three precisions. *SIAM Journal on Scientific Computing*, 40(2):A817–A847, 2018. doi: 10.1137/17M1140819.

Timothy A. Davis and Yifan Hu. The University of Florida Sparse Matrix Collection. *ACM Transactions on Mathematical Software*, 38(1), 2011. doi: 10.1145/2049662.2049663.

Eric De Sturler. Truncation strategies for optimal Krylov subspace methods. *SIAM Journal on Numerical Analysis*, 36(3):864–889, 1999.

Mark Embree. The tortoise and the hare restart GMRES. *SIAM Review*, 45(2): 259–266, 2003.

Anne Greenbaum, Vlastimil Pták, and Zdeněk Strakoš. Any nonincreasing convergence curve is possible for GMRES. *SIAM journal on matrix analysis and applications*, 17(3):465–469, 1996.

Nicholas J Higham and Srikara Pranesh. Simulating low precision floating-point arithmetic, mims eprint 2019.4. *Manchester Institute for Mathematical Sciences, The University of Manchester, UK*, 2019.

HPL-MxP. HPL-MxP mixed-precision benchmark. `https://icl.bitbucket.io/hpl-ai/`, November 2019.

Intel Corporation. Bfloat16 – hardware numerics definition. Technical Report 338302-001US, Revision 1.0, Intel, November 2018.

Shuhei Kudo, Keigo Nitadori, Takuya Ina, and Toshiyuki Imamura. Prompt report on exa-scale HPL-AI benchmark. In *2020 IEEE Int. Conf. Cluster Comput. (CLUSTER)*, pages 418–419. IEEE, 2020.

Jörg Liesen and Petr Tichý. The worst-case GMRES for normal matrices. *BIT Numerical mathematics*, 44:79–98, 2004.

Neil Lindquist, Piotr Luszczek, and Jack Dongarra. Accelerating restarted GMRES with mixed precision arithmetic. *IEEE Transactions on Parallel and Distributed Systems*, 33(4):1027–1037, 2022. doi: 10.1109/TPDS.2021.3090757.

Ronald B Morgan. GMRES with deflated restarting. *SIAM Journal on Scientific Computing*, 24(1):20–37, 2002.

Eda Oktay and Erin Carson. Multistage mixed precision iterative refinement. *Numerical Linear Algebra with Applications*, 29(4):e2434, 2022.

Michael L. Parks, Eric de Sturler, Greg Mackey, Duane D. Johnson, and Spandan Maiti. Recycling Krylov subspaces for sequences of linear systems. *SIAM Journal on Scientific Computing*, 28(5):1651–1674, 2006. doi: 10.1137/040607277.

Kirk M. Soodhalter, Eric de Sturler, and Misha Kilmer. A survey of subspace recycling iterative methods, 2020.

TOP500. TOP500. Online, June 2021. URL `https://www.top500.org/`.

J.M. Varah. The prolate matrix. *Linear Algebra and its Applications*, 187:269–278, 1993. ISSN 0024-3795. doi: 10.1016/0024-3795(93)90142-B.

James Hardy Wilkinson. *Rounding errors in algebraic processes*. Prentice-Hall, 1963.

8. (F)GMRES-IR for (W)LSP

Iterative refinement can also be used to improve the accuracy of solutions to the least squares (LS) problems $min_x\|b - Ax\|_2$, where $A \in \mathbb{R}^{m\times n}$. If $m > n$, the system is called overdetermined; if $m < n$, it is underdetermined.

If $m \geq n$ and $\text{rank}(A) = n$, then the system can be solved by QR factorization of A or via the normal equations $A^TAx = A^Tb$ (Björck [1996]). Details of various IR schemes for LS problems are discussed in Björck [1996], Higham [2002]. To use the SIR method in LS problems, one performs iterative refinement on the $(m + n) \times (m + n)$ augmented system

$$\underbrace{\begin{bmatrix} I^{m\times m} & A \\ A^T & 0 \end{bmatrix}}_{\tilde{A}} \underbrace{\begin{bmatrix} r \\ x \end{bmatrix}}_{\tilde{x}} = \underbrace{\begin{bmatrix} b \\ 0 \end{bmatrix}}_{\tilde{b}} \quad \text{or} \quad \tilde{A}\tilde{x} = \tilde{b}$$

via QR factorization. We call this approach LS-IR (Björck [1967])

The LS-IR algorithm of Björck does not form the augmented system explicitly, but instead performs solves using the QR factors of A. However, as stated above, QR factorization is very expensive and dominates the computation cost of the LS-IR algorithm. Thus, we can use multiple precisions in LS-IR to further reduce the computation cost. Namely, we can perform the QR factorization in some potentially lower precision u_f.

Analogous to SIR for linear systems, we can see that as long as $\kappa_\infty(\tilde{A}) \lesssim u_f^{-1}$, where u_f is the unit round-off of the precision used for the QR factorization, the backward error of LS-IR is $\mathcal{O}(u)$ and the forward error is

$$\frac{\|\tilde{x} - \hat{x}\|_\infty}{\|\tilde{x}\|_\infty} \approx u_r cond(\tilde{A}, \tilde{x}) + u,$$

where u_r and u are the unit round-offs of the residual precision and working precision used in LS-IR to store data and solutions, respectively. Based on this relation, we observe that if u_f is chosen to be fp16, LS-IR converges only for $\kappa_\infty(\tilde{A}) \lesssim 4.88 \cdot 10^4$, which restricts the range of problems that can be solved.

To use low precision for worse-conditioned systems, the authors in Carson et al. [2020] devised a GMRES-based iterative refinement algorithm analogous to the GMRES-IR approach in Carson and Higham [2017], called GMRES-LSIR, to solve least squares problems. GMRES-LSIR solves the augmented system (8.2) using GMRES preconditioned by a preconditioner M computed using the QR factors of A:

$$M = \begin{bmatrix} \alpha I & Q_1U \\ U^TQ_1^T & 0 \end{bmatrix},$$

where $A = Q_1U$ is the thin QR factorization of A. As long as $\kappa_\infty(A) \leq u^{-1/2}u_f^{-1}$, and assuming $u_r = u^2$, GMRES-LSIR provides $\mathcal{O}(u)$ backward and forward error. Furthermore, using the left preconditioner M, the conditioning of the preconditioned augmented matrix can be bounded by

$$\kappa_\infty(M^{-1}\tilde{A}) \lesssim (1 + 2m\sqrt{n}\tilde{\gamma}_{mn}^f\kappa_\infty(A))^2,$$

where

$$\tilde{\gamma}_{mn}^{f} = \frac{cmn}{1 - mnu_f},$$

and c is a small constant. This bound shows that even when $\kappa_\infty(A) \gg u_f^{-1}$, M will reduce the condition number of the augmented system when constructed in precision u_f.

In some scientific applications, such as electrical networks and interior point methods for constrained optimization, the standard least squares problem becomes insufficient since it cannot fully model a problem having a perturbed coefficient matrix. In such cases, a "weight" matrix D can be used to perturb the system and model it using the weighted least squares (WLS) problem

$$\min_x \|D^{1/2}(Ax - b)\|_2, \tag{8.1}$$

where $A \in \mathbb{R}^{m\times n}$ with $m \geq n$ and $D^{1/2}$ is a diagonal matrix of weights. In some applications, the weights can vary significantly and result in stiff problems, which makes it impossible to transform into a stable standard least squares problem ([Björck, 1996, Sec. 4.1.1]).

WLS problems can be solved via the normal equations

$$A^T DAx = A^T Db.$$

or the corresponding augmented system

$$\begin{bmatrix} \alpha D^{-1} & A \\ A^T & 0 \end{bmatrix} \begin{bmatrix} \alpha^{-1} y \\ x \end{bmatrix} = \begin{bmatrix} b \\ 0 \end{bmatrix}, \tag{8.2}$$

where $y = D(b - Ax)$, α is the scaling factor for stability.

An exact solution to WLS problems can be found using Householder QR factorization with row and column pivoting. For more details and error analysis, see Morell [1969] and Cox and Higham [1998], respectively. However, QR factorization can be expensive, especially when the matrix is large.

In Section 8.1, we introduce a GMRES-LSIR variant using flexible GMRES (FGMRES) instead of the GMRES algorithm for solving weighted least squares problems. For more detailed information on the algorithm, see Carson and Oktay [2024].

8.1 FGMRES-WLSIR

Motivated by its success, we want to solve WLS problems using GMRES-LSIR. For WLS problems, we need to construct an effective and inexpensive preconditioner M for the weighted augmented system so that $\kappa_\infty(M^{-1}\tilde{A})$ is small enough for GMRES to be backward stable in the given precision. Furthermore, we need to have a forward error of less than one for GMRES-LSIR to converge.

Using this roadmap, we focus on two preconditioners for GMRES-LSIR to solve WLS problems: A left preconditioner

$$M_l = \begin{bmatrix} \alpha D^{-1} & Q\hat{R} \\ \hat{R}^T Q^T & 0 \end{bmatrix}, \tag{8.3}$$

which is the direct extension of the M in Carson et al. [2020], except with a D^{-1} instead of I in the $(1,1)$ block and a block diagonal split preconditioner in Rozložník [2018]:

$$M_b = \begin{bmatrix} \alpha D^{-1} & 0 \\ 0 & \hat{C} \end{bmatrix}, \tag{8.4}$$

where $\hat{C} \approx \alpha^{-1} A^T D A$ is a symmetric positive definite approximation to the Schur complement.

Using the analysis in Carson et al. [2020], we can derive a bound for the conditioning of the preconditioned augmented system with M_l. For the bound, let

$$\widetilde{E} = \widetilde{A} - M_l = \begin{bmatrix} 0 & -E \\ -E^T & 0 \end{bmatrix},$$

where the error term E is defined as

$$A + E = Q\widehat{R}, \tag{8.5}$$

produced from the finite precision computation of the QR factorization in precision u_f ([Higham, 2002, Theorem 19.4]). Then we can write

$$\begin{aligned} M_l^{-1}\widetilde{A} &= M_l^{-1}(M_l + \widetilde{E}) = I + M_l^{-1}\widetilde{E} \quad \text{and} \\ \widetilde{A}^{-1} M_l &= (M_l + \widetilde{E})^{-1} M_l \approx I - M_l^{-1}\widetilde{E}, \end{aligned}$$

where the inverse of the preconditioner is

$$M_l^{-1} = \begin{bmatrix} \frac{1}{\alpha}\left(D - DQ(Q^TDQ)^{-1}Q^TD\right) & DQ(Q^TDQ)^{-1}\widehat{R}^{-T} \\ \widehat{R}^{-1}(Q^TDQ)^{-1}Q^TD & -\alpha\widehat{R}^{-1}(Q^TDQ)^{-1}\widehat{R}^{-T} \end{bmatrix}.$$

Using the equation above, we can find bound the condition number of the preconditioned matrix as

$$\kappa_\infty(M_l^{-1}\widetilde{A}) = \|M_l^{-1}\widetilde{A}\|_\infty \|\widetilde{A}^{-1}M_l\|_\infty \lesssim (1 + \|M_l^{-1}\widetilde{E}\|_\infty)^2. \tag{8.6}$$

To bound $\|M_l^{-1}\widetilde{E}\|_\infty$, we write

$$M_l^{-1}\widetilde{E} = \begin{bmatrix} -DQ(Q^TDQ)^{-1}\widehat{R}^{-T}E^T & -\frac{1}{\alpha}\left(D - DQ(Q^TDQ)^{-1}Q^TD\right)E \\ \alpha\widehat{R}^{-1}(Q^TDQ)^{-1}\widehat{R}^{-T}E^T & -\widehat{R}^{-1}(Q^TDQ)^{-1}Q^TDE \end{bmatrix},$$

where $(QDQ)^{-1}Q^TD$ is called the "scaled" or "weighted" pseudoinverse (Stewart [1989]). To give an (over)estimate on this bound, let $Q_D = D^{1/2}Q$. Then we have

$$\begin{aligned} \|(Q^TDQ)^{-1}Q^TD\|_2 &= \|(Q_D^TQ_D)^{-1}Q_D^TD^{1/2}\|_2 \\ &\le \|Q_D^\dagger\|_2\|D^{1/2}\|_2 \\ &\le \|Q^\dagger\|_2\|D^{-1/2}\|_2\|D^{1/2}\|_2 \\ &= \kappa_2^{1/2}(D). \end{aligned} \tag{8.7}$$

As an alternative, we can let

$$\|(Q^TDQ)^{-1}Q^TD\|_2 \equiv \rho,$$

and then argue that this ρ is hopefully of reasonable size since it is independent of D, using the results of Stewart [1989], in which the author shows that this quantity is bounded and is independent of D. In this study we will use this relation for simplicity.

Now, using Carson et al. [2020], we obtain

$$\begin{aligned}\|M_l^{-1}\widetilde{E}\|_\infty &\le \max(\|(M_l^{-1}\widetilde{E})(1,1)\|_\infty + \|(M_l^{-1}\widetilde{E})(1,2)\|_\infty, \\ &\qquad , \|(M_l^{-1}\widetilde{E})(2,1)\|_\infty + \|(M_l^{-1}\widetilde{E})(2,2)\|_\infty) \\ &\le \sqrt{m}\max\left(\|(M_l^{-1}\widetilde{E})(1,1)\|_F, \|(M_l^{-1}\widetilde{E})(2,1)\|_F\right) \\ &\quad + \sqrt{n}\max\left(\|(M_l^{-1}\widetilde{E})(1,2)\|_F, \|(M_l^{-1}\widetilde{E})(2,2)\|_F\right), \end{aligned} \tag{8.8}$$

where $(M_l^{-1}\widetilde{E})(i,j)$ denotes the (i,j)-block of $M_l^{-1}\widetilde{E}$.

Using $|\alpha| \approx \|A^\dagger\|_2^{-1}$ and ignoring terms of order u_f^2,

$$\begin{aligned}\|(M_l^{-1}\widetilde{E})(1,1)\|_F &= \|DQ(Q^TDQ)^{-1}\widehat{R}^{-T}E^T\|_F \le \rho\|A^\dagger\|_2\|E^T\|_F \\ \|(M_l^{-1}\widetilde{E})(1,2)\|_F &= \left\|\frac{1}{\alpha}D\left(I - Q(Q^TDQ)^{-1}Q^TD\right)E\right\|_F \le \rho\|D\|_2\|A^\dagger\|_2\|E\|_F \\ \|(M_l^{-1}\widetilde{E})(2,2)\|_F &= \|\widehat{R}^{-1}(Q^TDQ)^{-1}Q^TDE\|_F \le \rho\|A^\dagger\|_2\|E\|_F \\ \|(M_l^{-1}\widetilde{E})(2,1)\|_F &= \|\alpha\widehat{R}^{-1}(Q^TDQ)^{-1}\widehat{R}^{-T}E^T\|_F \\ &= \|\alpha\widehat{R}^{-1}(Q^TDQ)^{-1}Q^TD(Q^TD)^\dagger\widehat{R}^{-T}E^T\|_F \\ &\le \rho\|D^{-1}\|_2\|A^\dagger\|_2\|E^T\|_F.\end{aligned}$$

Plugging them into (8.8), and letting $\theta \equiv \max(1, \|D\|_2, \|D^{-1}\|_2)$, we get

$$\|M_l^{-1}\widetilde{E}\|_\infty \le (\sqrt{m}+\sqrt{n})\rho\theta\|A^\dagger\|_2\|E\|_F.$$

Assuming that we have a standard QR factorization, $\|E\|_F \le \tilde{\gamma}_{mn}^f\|A\|_F$, and thus

$$\|M_l^{-1}\widetilde{E}\|_\infty \le (\sqrt{m}+\sqrt{n})\rho\theta\|A^\dagger\|_2\|E\|_F \tag{8.9}$$

$$\le 2m\sqrt{n}\rho\theta\tilde{\gamma}_{mn}^f\kappa_\infty(A). \tag{8.10}$$

Note that in the case that we use a standard QR factorization and $D = I$ (and thus $\rho = 1$), we recover the same bound as in [Carson et al., 2020, Section 3.1]. Finally, plugging into (8.6), we obtain

$$\kappa_\infty(M_l^{-1}\widetilde{A}) \lesssim \left(1 + 2m\sqrt{n}\rho\theta\tilde{\gamma}_{mn}^f\kappa_\infty(A)\right)^2. \tag{8.11}$$

The D dependence in the bound limits the applicability of the algorithm since, in real applications, D can be very ill-conditioned. Using the split preconditioner M_b and its analysis in Rozložník [2018], we can derive a similar bound as follows. The preconditioned system with split preconditioner can be written as

$$\begin{aligned}M_b^{-1/2}\tilde{A}M_b^{-1/2} &= \begin{bmatrix}(\alpha D^{-1})^{-1/2} & 0 \\ 0 & \hat{C}^{-1/2}\end{bmatrix}\begin{bmatrix}\alpha D^{-1} & A \\ A^T & 0\end{bmatrix}\begin{bmatrix}(\alpha D^{-1})^{-1/2} & 0 \\ 0 & \hat{C}^{-1/2}\end{bmatrix} \\ &= \begin{bmatrix}(\alpha D^{-1})^{-1/2}(\alpha D^{-1})(\alpha D^{-1})^{-1/2} & (\alpha D^{-1})^{-1/2}A^T\hat{C}^{-1/2} \\ \hat{C}^{-1/2}A^T(\alpha D^{-1})^{-1/2} & 0\end{bmatrix} \\ &= \begin{bmatrix}I & \hat{A} \\ \hat{A}^T & 0\end{bmatrix},\end{aligned}$$

where

$$\hat{A} = \frac{1}{\sqrt{\alpha}} D^{1/2} A^T \hat{C}^{-1/2}.$$

There is no applicable analysis of the forward and backward errors in split preconditioned GMRES in the literature. However, there is an existing analysis of these quantities for a general split preconditioned flexible GMRES method (FGMRES). Thus, to be able to use split preconditioners and discuss their effects on the stability of this general approach, we use FGMRES instead of GMRES in GMRES-LSIR. Our new variant to solve WLSP is, therefore, called FGMRES-WLSIR.

As explained above, since $\hat{C}$ is spectrally equivalent to the matrix $\alpha^{-1}A^TDA$, there exist positive constants $0 < \hat{\gamma} \leq \hat{\delta}$ such that

$$\hat{\gamma}(\hat{C}y, y) \leq (B^T(\alpha D^{-1})^{-1}Ay, y) \leq \hat{\delta}(\hat{C}y, y) \quad \forall y \in \mathbb{R}^n.$$

Using [Rozložník, 2018, Proposition 3.4], the spectrum of the preconditioned matrix can be written as

$$sp(M_b^{-1/2}\tilde{A}M_b^{-1/2}) = \frac{1}{2}\left(1 \pm \sqrt{1 + 4\sigma_k^2(\hat{A})}\right) \quad for \quad k = 1, \ldots, n-r,$$

where $\text{rank}(\hat{A}) = n - r$ and $0 < \sigma_{n-r}(\hat{A}) \leq \cdots \leq \sigma_1(\hat{A})$ are the singular values of $\hat{A}$.

We assume that the Schur complement is computed exactly and we consider the left-preconditioned matrix

$$M_b^{-1}\tilde{A} = \begin{bmatrix} I & \alpha^{-1}A^TD \\ \alpha^{-1}A(A^TDA)^{-1} & 0 \end{bmatrix},$$

which is a nonsymmetric diagonalizable matrix with three distinct eigenvalues $\left\{1, \frac{1}{2}(1 \pm \sqrt{5})\right\}$. This makes the block diagonal preconditioner a suitable choice for FGMRES since the method can converge in a small number of iterations. However, our experiments show that $\kappa_\infty(M_b^{-1}\tilde{A})$ can be very large, often larger than $\kappa_\infty(\tilde{A})$, when A is ill-conditioned. An ill-conditioned preconditioned matrix makes the preconditioner unsuitable for proving the backward stability of FGMRES although we show that in practice, the split preconditioner improves the condition number of the preconditioned matrix.

Using the analysis in Rozložník [2018], we can obtain the bound

$$\kappa_\infty(M_b^{-1/2}\tilde{A}M_b^{-1/2}) \leq \frac{|1 + \sqrt{1 + 4\sigma_1^2(\hat{A})}|}{|1 - \sqrt{1 + 4\sigma_{n-r}^2(\hat{A})}|}(n+m), \tag{8.12}$$

where $0 < \sigma_{n-r}(\hat{A}) \leq \ldots \leq \sigma_1(\hat{A})$, are the singular values of $\hat{A}$.

Although the bound on M_b is also dependent on D, we can reduce its impact numerically via computing $D^{1/2}A$ in high precision and using R factor (from lower precision QR factorization) of it as $\hat{C}$, which makes the preconditioner advantageous numerically. However, the forward error of FGMRES also depends on the conditioning of the right preconditioner $M_b^{-1/2}$

$$\frac{\|x - \bar{x}_k\|}{\|x\|} \lesssim \kappa_\infty(M_b^{-1/2}\tilde{A}M_b^{-1/2})\kappa_\infty(M_b^{-1/2})\mathcal{O}(u),$$

Table 8.1: Properties of matrices from the SuiteSparse collection.

Name	m	n	$\kappa_2(A)$	#nnz
`ash958`	958	292	3.2014	1916
`robot24c1_mat5`	404	302	3.33×10^{11}	15118

whereas in the case of any left preconditioner M_l, FGMRES has a forward error

$$\frac{\|x - \bar{x}_k\|}{\|x\|} \lesssim \kappa_\infty(M_l^{-1}\tilde{A})\mathcal{O}(u).$$

These bounds show that even though $M_b^{-1/2}\tilde{A}M_b^{-1/2}$ is well-conditioned, $M_b^{-1/2}$ can be still ill-conditioned. We thus need to analyze $\kappa_\infty(M_b^{-1/2})$ as well for the block split preconditioned FGMRES-WLSIR.

To observe the effect of preconditioning, we perform numerical experiments in MATLAB using sparse matrices from the SuiteSparse collection in Davis and Hu [2011] and random sparse matrices generated in MATLAB. The properties of the two sparse matrices from the collection are given in Table 8.1. Code for our FGMRES-WLSIR and associated functions can be found in the repository `https://github.com/edoktay/fgmreswlsir`.

For our experiments, we set the weight matrix D such that each row i of the matrix A has $\max|A(i,j)| = 1$. The scaling factor $\alpha = 2^{-1/2}\sigma_n$, where σ_n is the smallest singular value of A. We compute QR factorizations in half, single, and double precision and construct the corresponding preconditioners M. We then measure the infinity-norm condition number of the preconditioned system $M^{-1}\tilde{A}$.

In each figure, the condition number of the preconditioned systems is represented as colored lines. Each color shows half (red), single (green), and double (blue) precisions used for computing QR factorizations for the preconditioners. The dashed black line gives the condition number of the unpreconditioned augmented system and the dotted black line gives the inverse of the unit round-off for the FGMRES-WLSIR working precision, u. The figures show that the convergence of FGMRES-WLSIR is guaranteed only when the colored solid lines remain below the dashed line. Numerically, this shows the cases when $\kappa_\infty(M^{-1}\tilde{A}) \leq u^{-1}$. Only in this case can we guarantee that the forward error of FGMRES is less than 1 and thus FGMRES-WLSIR converges.

To examine the effect of preconditioners on the conditioning of the augmented system, we also construct Table 8.2 using random dense matrices A' with a randomly generated solution vector b. We prescale the rows of A' so that they have drastically different sizes. We use 7 different scalings $A = SA'$ where S is a diagonal matrix created by `diag(logspace(1, j, 100)` where $j \in [4, 6, 8, 10, 12, 14, 16]$ to create different matrices with different condition numbers. We then set the solution vector b using `randn(m,1)`, where $A \in \mathbb{R}^{m\times n}$. From Table 8.2, we see that for the chosen matrices, M_l is not able to decrease the condition number sufficiently, whereas M_b works well. We finally observe from the last column that even though $M_b^{1/2}$ is ill-conditioned since the preconditioned system is very well conditioned, FGMRES-WLSIR still converges due to the forward error constraint in (8).

With these numerical experiments, we conclude that since the conditioning of the weight matrix is highly problem-dependent, we cannot generalize which

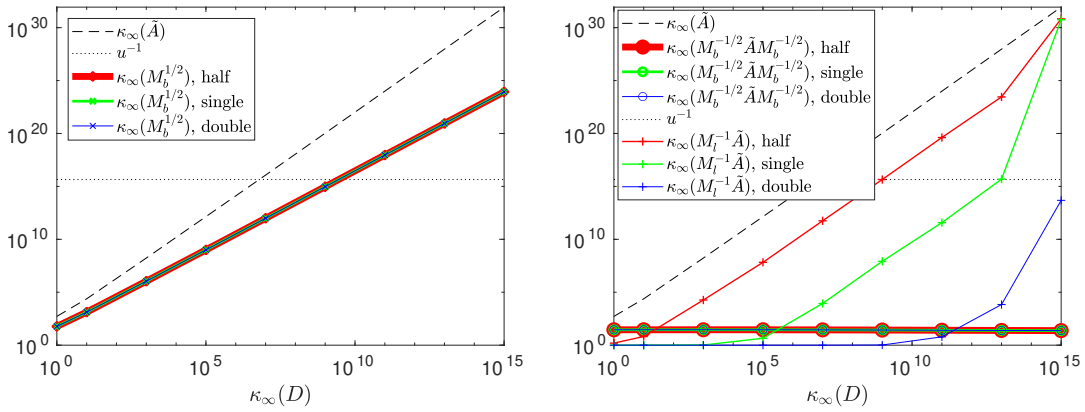

Figure 8.1: Measured condition number of the preconditioners (left) and preconditioned systems (right) using *ash958* matrix as A where M_l and M_b are constructed using QR factorizations in various precisions, versus the condition number of the weight matrix D. The working precision for FGMRES-WLSIR is assumed to be double precision.

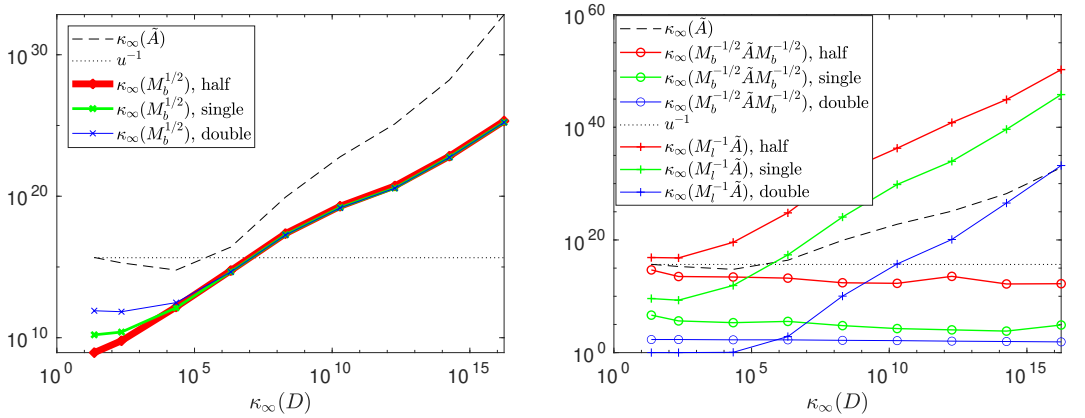

Figure 8.2: Measured condition number of the preconditioners (left) and preconditioned systems (right) using *robot24c1_mat5* matrix as A, where M_l and M_b are constructed using QR factorizations in various precisions, versus the condition number of the weight matrix D. The working precision for FGMRES-WLSIR is assumed to be double precision.

preconditioner is more useful for FGMRES-WLSIR. Although our numerical experiments show that using the block split preconditioner in half-precision may be used in more ill-conditioned systems, in real applications, D might be worse-conditioned.

Because of the D dependence of both preconditioners and right preconditioner dependence of split preconditioned FGMRES, this work can be studied further. Further studies can focus either on the choice of a preconditioner or another iterative approach other than FGMRES. In any case, the optimal algorithm needs to have the error bound of the preconditioner being independent from the weight matrix.

Table 8.2: Condition numbers of $\tilde{A}$, right preconditioner, and preconditioned augmented matrices.

$\kappa_2(A)$	$\kappa_\infty(\tilde{A})$	$\kappa_\infty(M_l^{-1}\tilde{A})$	$\kappa_\infty(M_b^{-1/2}\tilde{A}M_b^{-1/2})$	$\kappa_\infty(M_b^{1/2})$
1.47e+02	1.73e+05	1.16e+02	5.79e+01	3.60e+02
1.91e+02	8.04e+08	1.14e+05	4.03e+01	3.08e+04
2.47e+02	5.06e+12	1.50e+08	3.02e+01	2.48e+06
3.21e+02	3.81e+16	4.36e+11	2.62e+01	2.16e+08
4.22e+02	2.97e+20	5.24e+14	2.30e+01	1.87e+10
5.61e+02	2.26e+24	1.12e+18	2.05e+01	1.63e+12
7.59e+02	1.67e+28	1.16e+22	1.86e+01	1.36e+14

Bibliography

Åke Björck. Iterative refinement of linear least squares solutions i. *BIT Numerical Mathematics*, 7(4):257–278, 1967.

Åke Björck. *Numerical Methods for Least Squares Problems*. Society for Industrial and Applied Mathematics, 1996. doi: 10.1137/1.9781611971484.

Erin Carson and Nicholas J. Higham. A new analysis of iterative refinement and its application to accurate solution of ill-conditioned sparse linear systems. *SIAM Journal on Scientific Computing*, 39(6):A2834–A2856, 2017. doi: 10.1137/17M1122918.

Erin Carson and Eda Oktay. Mixed precision FGMRES-based iterative refinement for weighted least squares. *arXiv preprint arXiv:2401.03755*, 2024.

Erin Carson, Nicholas J. Higham, and Srikara Pranesh. Three-precision GMRES-based iterative refinement for least squares problems. *SIAM Journal on Scientific Computing*, 42(6):A4063–A4083, 2020. doi: 10.1137/20M1316822.

Anthony J. Cox and Nicholas J. Higham. Stability of Householder QR factorization for weighted least squares problems. In D. F. Griffiths, D. J. Higham, and G. A. Watson, editors, *Numerical Analysis* 1997*, Proceedings of the* 17*th Dundee Biennial Conference*, volume 380 of *Pitman Research Notes in Mathematics*, pages 57–73. Addison Wesley Longman, Harlow, Essex, UK, 1998.

Timothy A. Davis and Yifan Hu. The University of Florida sparse matrix collection. *ACM Trans. Math. Softw.*, 38(1), dec 2011. ISSN 0098-3500. doi: 10.1145/2049662.2049663.

Nicholas J Higham. *Accuracy and stability of numerical algorithms*. SIAM, 2002.

A.J.M. Morell, editor. *On applying Householder's method to linear least squares problems*, 1969. North-Holland, Amsterdam. pp. 122-126.

Miroslav Rozložník. *Saddle-point problems and their iterative solution*. Springer, 2018.

Gilbert W Stewart. On scaled projections and pseudoinverses. *Linear Algebra and its Applications*, 112:189–193, 1989.

9. Multistage mixed-precision iterative refinement[1]

Iterative refinement (IR) is frequently used in solving linear systems $Ax = b$, where $A \in \mathbb{R}^{n\times n}, x, b \in \mathbb{R}^n$, to improve the accuracy of a computed approximate solution $\hat{x} \in \mathbb{R}^n$. Typically, one computes an initial approximate solution $\hat{x}_0 \in \mathbb{R}^n$ using Gaussian elimination with partial pivoting (GEPP), saving the approximate factorization $A \approx \hat{L}\hat{U}$, where $\hat{L} \in \mathbb{R}^{n\times n}$ is a lower triangular matrix with unit diagonal, and $\hat{U} \in \mathbb{R}^{n\times n}$ is an upper triangular matrix. After computing the residual $\hat{r} = b - A\hat{x} \in \mathbb{R}^n$ (potentially in higher precision), one reuses $\hat{L}$ and $\hat{U}$ to solve the system $A\hat{d} = \hat{r}$, where $\hat{d} \in \mathbb{R}^n$. The original approximate solution is subsequently refined by adding the corrective term, $\hat{x} = \hat{x}+\hat{d}$. This process can be repeated until either the desired accuracy is reached or the iterative refinement process converges to an approximate solution. Iterative refinement algorithms that exploit multiple different precisions have seen renewed attention recently due to the emergence of mixed precision hardware, and form the basis for the recently developed HPL-AI benchmark, on which today's top supercomputers exceed exascale performance; see e.g., HPL-MxP, Kudo et al. [2020], TOP500.

Algorithm 1 shows a general mixed precision iterative refinement scheme. Following the work of Carson and Higham [2018], there are four different precisions specified: u_f denotes the factorization precision, u denotes the working precision, u_r denotes the precision for the residual computation, and u_s denotes the effective precision with which the correction equation is solved. This final precision, u_s, is not a hardware precision but will rather depend on the particular solver used in line 4 and the particular precision(s) used within the solver (which may be different from u, u_f, and u_r). It is assumed that $u_f \geq u \geq u_r$.

The choice of precisions u, u_f, and u_r, and the choice of a solver to use in line 4 defines different variants of iterative refinement. We refer to any variant that solves the correction equation in line 4 using triangular solves with the computed LU factors as "standard iterative refinement" (SIR). For SIR, the effective precision of the correction solve is limited by the precision with which the factorization is computed, and thus we have $u_s = u_f$. The most commonly-used SIR variant is what we call "traditional" iterative refinement, in which $u_f = u$ and $u_r = u^2$. For instance, if the working precision is single, i.e., $u = 5.96 \cdot 10^{-8}$, then for $u_r = u^2 \approx 10^{-16}$, double precision is used. Traditional iterative refinement was used already by Wilkinson in 1948 (Wilkinson [1948]), and was analyzed in fixed point arithmetic by Wilkinson in 1963 (Wilkinson [1963]) and in floating point arithmetic by Moler in 1967 (Moler [1967]). There have also been analyses of fixed precision iterative refinement, in which $u_f = u = u_r$ (Jankowski and Woźniakowski [1977]), as well as low precision factorization iterative refinement, in which $u_f^2 = u = u_r$ (Langou et al. [2006]). Motivated by the trend of low

[1]This chapter is a pre-copyedited, author-produced version of an article accepted for publication in Wiley, Numerical Linear Algebra with Applications following peer review. The version of record [Numerical Linear Algebra with Applications, Oktay, E., Carson, E.: Multistage mixed-precision iterative refinement] is available online at `https://onlinelibrary.wiley.com/doi/pdf/10.1002/nla.2434`.

and mixed precision capabilities in hardware, the authors in Carson and Higham [2018] developed and analyzed a three-precision iterative refinement scheme, in which u_f, u, and u_r may differ, which generalizes (and in some cases, improves the bounds for) many existing variants. For references to analyses of variants of iterative refinement, see [Carson and Higham, 2018, Table 1.1].

If A is very ill conditioned or badly scaled, SIR can fail, i.e., the error is not eventually bounded by a small multiple of machine precision. In extreme cases of ill conditioning, the error can grow with each refinement step. In Carson and Higham [2017] and Carson and Higham [2018], the authors developed a mixed precision GMRES-based iterative refinement scheme (GMRES-IR), shown in Algorithm 17. The only difference with SIR is the way in which the correction equation is solved. The idea is that instead of using the LU factors to solve for the correction in each step, one can instead use these factors as (left) preconditioners for a preconditioned GMRES method which solves for the correction. In this way, the effective solve precision becomes $u_s = u$, and more ill-conditioned problems can be handled relative to SIR. The GMRES-based refinement approaches have seen much success in practice. Current GMRES-based iterative refinement variants are implemented in the MAGMA library (2.5.0) and in the NVIDIA cuSOLVER library. GMRES-IR also forms the basis for the new HPL-AI benchmark, used to rank computers in the TOP500 list (HPL-MxP). Experiments detailing the performance benefits of three-precision GMRES-IR and SIR can be found in Haidar et al. [2018].

Although GMRES-IR can succeed where SIR fails, each step of GMRES-IR is potentially more expensive than each step of SIR (albeit still potentially less expensive than running the entire process in double the working precision). Whereas each SIR refinement step involves only two triangular solves in precision u, each GMRES-IR step involves two triangular solves in precision u^2 in *each* GMRES iteration (in addition to a matrix-vector product in precision u^2 as well as multiple vector operations in precision u). The convergence behavior of each GMRES solve thus plays a large role, and it is important for the performance of GMRES-IR that each call to GMRES converges relatively quickly. It is additionally a performance concern that each GMRES iteration requires computations in higher precision; in order to obtain the needed bound on backward error of the GMRES solve, the authors in Carson and Higham [2017, 2018] required that the preconditioned system $U^{-1}L^{-1}A$ (not formed explicitly) is applied to a vector in precision u^2.

The authors of Carson and Higham [2017, 2018] suggested that this required use of extra precision was likely to be overly strict in many practical scenarios. Indeed, most practical implementations of GMRES-based iterative refinement do not use this extra precision (Amestoy et al. [2021]). This motivated Amestoy et al. (Amestoy et al. [2021]) to develop an extension of the GMRES-IR algorithm, called GMRES-IR5. The authors in Amestoy et al. [2021] revisit the proof of backward stability for GMRES from Paige et al. [2006] and develop a bound on the backward error for a mixed precision GMRES algorithm, where $u_g \geq u$ is the working precision used within GMRES and u_p is the precision in which the preconditioned matrix is applied to a vector. This enables a five-precision GMRES-IR scheme, where u, u_f, u_r, u_g, and u_p may take on different values. A particular improvement over the GMRES-IR scheme of Carson and Higham [2017,

Table 9.1: Asymptotic computational complexity of operations in each refinement step for SIR, SGMRES-IR, and GMRES-IR.

Once per IR solve (all variants)	$O(n^3)$	in precision u_f	(LU fact.)
SIR step	$O(n^2)$	in precision u_f	(tri. solves)
SGMRES-IR step (k GMRES iterations)	$O(nk^2)$	in precision u	(orthog.)
	$O(nnz \cdot k)$	in precision u	(SpMV)
	$O(n^2k)$	in precision u	(precond.)
GMRES-IR step (k GMRES iterations)	$O(nk^2)$	in precision u	(orthog.)
	$O(nnz \cdot k)$	in precision u^2	(SpMV)
	$O(n^2k)$	in precision u^2	(precond.)
Once per refinement step (all variants)	$O(nnz)$	in precision u_r	(residual comp.)
	$O(n)$	in precision u	(sol. update)

2018] is that the analysis provides bounds on forward and backward errors for a variant of GMRES-IR in which the entire GMRES iteration is carried out in a uniform precision, i.e., the analysis does not require that extra precision is used in applying the preconditioned matrix to a vector. This particular variant of the algorithm is thus less expensive than the former GMRES-IR in terms of both time and memory. The drawback is that it is only theoretically applicable to a smaller set of problems due to a tighter constraint on condition number. We call the particular instance of GMRES-IR5 where $u = u_g = u_p$ "SGMRES-IR" (for "simpler"), shown in Algorithm 20. To make more precise the relative costs of each step of SIR, SGMRES-IR, and GMRES-IR, we list their costs in terms of asymptotic computational complexity in Table 9.1. We discuss the constraints on condition number under which each variant converges later in Section 9.1.2.

Algorithm 20 SGMRES-IR (a particular variant of GMRES-IR5 (Amestoy et al. [2021]))

Input: matrix $A^{n\times n}$; right-hand side b^n; maximum number of refinement steps i_{max}; GMRES convergence tolerance τ.

Output: Approximate solution x_{i+1} to $Ax = b$.

1: Compute LU factorization $A = LU$ in precision u_f.
2: Solve $Ax_0 = b$ by substitution in precision u_f, store in precision u.
3: **for** $i = 0$: $i_{max} - 1$ **do**
4: Compute $r_i = b - Ax_i$ in precision u_r, store in precision u.
5: Solve $U^{-1}L^{-1}Ad_{i+1} = U^{-1}L^{-1}r_i$ by GMRES in working precision u, with matrix-vector products with $\tilde{A} = U^{-1}L^{-1}A$ computed at precision u; store in precision u.
6: Compute $x_{i+1} = x_i + d_{i+1}$ in precision u.
7: **if** converged **then** return x_{i+1}. **end if**

In terms of both cost and range of condition numbers to which the algorithm can be applied, we expect SGMRES-IR to be, in general, somewhere between SIR and GMRES-IR. For example, if we use single precision for u_f, double precision for u, and quadruple precision for u_r, SIR is guaranteed to converge to the level of double precision in both forward and backward errors as long as the infinity-norm

condition number of the matrix A, $\kappa_\infty(A) = \|A\|_\infty \|A^{-1}\|_\infty$, is less than $2 \cdot 10^7$. For SGMRES-IR, this constraint becomes $\kappa_\infty(A) \leq 10^{10}$. GMRES-IR, on the other hand, only requires $\kappa_\infty(A) \leq 2 \cdot 10^{15}$. Thus going from SIR to SGMRES-IR to GMRES-IR, we expect that these algorithms will be increasingly expensive but also expect that they will converge for increasingly ill-conditioned matrices. We note this may not always be the case. For some precision combinations, SGMRES-IR has a tighter constraint on condition number than SIR (see Amestoy et al. [2021]), although this may be an artifact of the analysis. Also, the metric of relative "cost" is difficult to determine a priori, in particular between SGMRES-IR and GMRES-IR, since it depends on the number of GMRES iterations required in each refinement step.

It is thus difficult to choose a priori which particular variant of iterative refinement is the most appropriate for a particular problem. Even if the matrix condition number meets the constraint for convergence for the chosen iterative refinement algorithm, the convergence rate may be unacceptably slow, or each step may require so many GMRES iterations that it becomes impractical. Further, as our experiments show, the condition number constraints in the literature are can be too tight, meaning that for some problems a given refinement scheme converges even when the analysis indicates that it may not. This could lead users to select a more expensive iterative refinement variant than is actually needed in practice.

In this work, we aim to solve this problem through the development of a multistage, three-precision iterative refinement scheme, which we abbreviate MSIR. Our approach automatically switches between solvers and precisions if slow convergence (of the refinement scheme itself or of the inner GMRES solves) is detected using stopping criteria adapted from the work in Demmel et al. [2006]. Two novel aspects of our approach are (1) we attempt to use "stronger" solvers before resorting to increasing the precision of the factorization, and (2) when executing a GMRES-based refinement algorithm, we modify the stopping criteria to also restrict the number of GMRES iterations per refinement step.

Table 9.1 shows why first switching the solver may be more favorable from a performance perspective; whereas increasing the precision and recomputing the factorization will cost $O(n^3)$ flops in precision u_f^2, where u_f is the current factorization precision, using the existing factorization and performing k total GMRES iterations may be faster, requiring $O(n^2 k)$ flops in precision u. This motivates point (2) above. If the number of GMRES iterations k is too large, then one (S)GMRES-IR refinement step at least as expensive as recomputing the factorization in a higher precision (u for SGMRES-IR or u^2 for GMRES-IR).

Our approach may serve to improve existing multistage iterative refinement implementations. For example, the MAGMA library (Tomov et al. [2010]) currently uses a variant of SGMRES-IR and if convergence is not detected after the specified maximum number of iterations, the factorization is recomputed in a higher precision and the refinement restarts. Our numerical experiments confirm that it may be beneficial to first try a different solver before resorting to recomputing the factorization.

In Section 9.1, we present the MSIR algorithm and give a motivating numerical example. We then give details of the stopping criteria used and summarize the analysis for each algorithm variant from Carson and Higham [2018] and Amestoy

et al. [2021]. In Section 9.2 we present more thorough numerical experiments on both random dense matrices and matrices from the SuiteSparse collection (Davis and Hu [2011]). We conclude and discuss future extensions in Section 9.3.

9.1 The MSIR algorithm

In order to balance reliability and cost, we develop a multistage iterative refinement algorithm (MSIR), presented in Algorithm 21. The algorithm starts with three-precision SIR (as in Carson and Higham [2018]) and, using the stopping criteria developed in Demmel et al. [2006], switches to SGMRES-IR if the algorithm is not converging at an acceptable rate (or not converging at all). Then, using the same stopping criteria along with an additional constraint on the number of GMRES iterations per refinement step, the algorithm may choose to switch a second time to the GMRES-IR algorithm (which uses higher precision in applying the preconditioned matrix to a vector within GMRES). If convergence is still not achieved or is too slow, then as a fail-safe, we increase the factorization precision u_f (and the other precisions if necessary to satisfy $u_f \geq u$, $u_r \leq u^2$), recompute the LU factorization, and begin the process again with SIR. We enforce $u_r \leq u^2$ in order to guarantee the convergence of the forward error to the level of the working precision, but this strategy for increasing precisions could be modified in practice. For instance, if the initial setting is $(u_f, u, u_r) =$ (half, single, double), we would double only u_f and use $(u_f, u, u_r) =$ (single, single, double), and then if we need to increase precisions again, we would use $(u_f, u, u_r) =$ (double, double, quad).

Before explaining the details of the algorithm, we begin with a brief motivating example illustrating how MSIR works. We restrict ourselves to IEEE precisions and use initial precisions $(u_f, u, u_r) =$ (single, double, quad) and test random dense matrices of size 100×100 generated using the MATLAB command `gallery('randsvd',n,kappa(i),2)`, where `kappa` is the array of the desired 2-norm condition numbers. Here we test 2-norm condition numbers $10^1, 10^5$, and 10^{16}. We set b to be a vector of normally distributed numbers generated by the MATLAB command `randn`. For reproducibility, we use the MATLAB command `rng(1)` to seed the random number generator before generating each linear system. The GMRES convergence tolerance $\tau \in \mathbb{R}^+$, which also appears in Algorithms 17 and 20, dictates the stopping criterion for the inner GMRES iterations. The algorithm is considered to be converged if the relative (preconditioned) residual norm drops below τ. In all tests here we use $\tau = 10^{-6}$. The particular criteria by which we switch between solvers is discussed in detail in Section 9.1.1.

Figures 9.1, 9.2, and 9.3 show results for condition numbers $10^1, 10^5$, and 10^{16}, respectively. The red, blue, and green lines in the figures show the behavior of the forward error `ferr` (red), normwise relative backward error `nbe` (blue), and componentwise relative backward error `cbe` (green). The dotted black line shows the value of the initial working precision u. If there is a switch in MSIR, the step at which it happened is marked with a star. For instance, if MSIR uses both SGMRES-IR and GMRES-IR, then there are two stars: the first one marks the switch from SIR to SGMRES-IR and the second one marks the switch from SGMRES-IR to GMRES-IR.

Algorithm 21 Multistage Iterative Refinement (MSIR)

Input: $n \times n$ matrix A; right-hand-side b; maximum number of refinement steps of each type i_{max}; GMRES convergence tolerance τ; stopping criteria parameter ρ_{thresh}; maximum GMRES iterations $k_{max} \in \mathbb{N}^+$; initial factorization precision u_f; initial working precision u; initial residual precision u_r.

Output: Approximate solution x_{i+1} to $Ax = b$, boolean `cged`.

Compute LU factorization $A = LU$ in precision u_f.
Solve $Ax_0 = b$ by substitution in precision u_f, store in precision u.
Initialize: $d_0 = \infty$; `alg` = `SIR`; `iter` = 0; $i = 0$; `cged` = 0; $\rho_{max} = 0$.
while not `cged` **do**
    Compute $r_i = b - Ax_i$ in precision u_r, store in precision u.
    Scale $r_i = r_i / \|r_i\|_\infty$.
    if `alg` = `SIR` **then**
        `iter` = `iter` +1
        Compute $d_{i+1} = U^{-1}(L^{-1}r_i)$ in precision u_f; store in precision u.
        if d_{i+1} contains `Inf` or `NaN` **then** `alg` = `SGMRES-IR`; `iter` = 0; break. **end if**
        Compute $x_{i+1} = x_i + \|r_i\|_\infty d_{i+1}$ in precision u.
        $z = \|d_{i+1}\|_\infty / \|x_i\|_\infty$; $v = \|d_{i+1}\|_\infty / \|d_i\|_\infty$; $\rho_{max} = \max(\rho_{max}, v)$; $\phi_i = z/(1-\rho_{max})$
        if $z \le u$ or $v \ge \rho_{thresh}$ or `iter` $> i_{max}$ or $\phi_i \le \sqrt{n}u$ **then**
            if not converged **then**
                `alg` = `SGMRES-IR`; `iter` = 0.
                if $\phi_i > \phi_0$ **then** $x_{i+1} = x_0$ **end if**
            else
                `cged` = 1
    else if `alg` = `SGMRES-IR` **then**
        `iter` = `iter` +1
        Solve $U^{-1}L^{-1}Ad_{i+1} = U^{-1}L^{-1}r_i$ by GMRES in precision u with matrix-vector products with $\tilde{A} = U^{-1}L^{-1}A$ computed at precision u, store in precision u.
        Compute $x_{i+1} = x_i + \|r_i\|_\infty d_{i+1}$ in precision u.
        $z = \|d_{i+1}\|_\infty / \|x_i\|_\infty$; $v = \|d_{i+1}\|_\infty / \|d_i\|_\infty$; $\rho_{max} = \max(\rho_{max}, v)$; $\phi_i = z/(1-\rho_{max})$
        if $z \le u$ or $v \ge \rho_{thresh}$ or `iter` $> i_{max}$ or $k_{GMRES} > k_{max}$ or $\phi_i \le \sqrt{n}u$ **then**
            if not converged **then**
                `alg` = `GMRES-IR`; `iter` = 0.
                if $\phi_i > \phi_0$ **then** $x_{i+1} = x_0$ **end if**
            else
                `cged` = 1
    else if `alg` = `GMRES-IR` **then**
        `iter` = `iter` +1
        Solve $U^{-1}L^{-1}Ad_{i+1} = U^{-1}L^{-1}r_i$ by GMRES in precision u with matrix-vector products with $\tilde{A} = U^{-1}L^{-1}A$ computed at precision u^2, store d_{i+1} in precision u.
        Compute $x_{i+1} = x_i + \|r_i\|_\infty d_{i+1}$ in precision u.
        $z = \|d_{i+1}\|_\infty / \|x_i\|_\infty$; $v = \|d_{i+1}\|_\infty / \|d_i\|_\infty$; $\rho_{max} = \max(\rho_{max}, v)$; $\phi_i = z/(1-\rho_{max})$
        if $z \le u$ or $v \ge \rho_{thresh}$ or `iter` $> i_{max}$ or $k_{GMRES} > k_{max}$ or $\phi_i \le \sqrt{n}u$ **then**
            if not converged **then**
                $u_f = u_f^2$; `alg` = `SIR`; `iter` = 0.
                Compute LU factorization $A = LU$ in precision u_f.
                if $u_f < u$ **then** $u = u_f$ **end if**
                if $u_r > u^2$ **then** $u_r = u^2$ **end if**
                if $\phi_i > \phi_0$ **then** $x_{i+1} = x_0$ **end if**
            else
                `cged` = 1
    $i = i + 1$

For figures in this study, forward error is calculated as $\|\hat{x} - x\|_\infty / \|x\|_\infty$. The normwise relative backward error for $\hat{x}$ is calculated as

$$\frac{\|b - A\hat{x}\|}{\|A\|\|\hat{x}\| + \|b\|},$$

whereas for the computation of the componentwise relative backward error,

$$\max_i \frac{|b - A\hat{x}|_i}{(|A||\hat{x}| + |b|)_i}$$

is used.

The number of refinement steps performed by SIR, SGMRES-IR, GMRES-IR, and MSIR for these matrices are presented in Table 9.2. For SIR, the number in each row gives the number of refinement steps. For SGMRES-IR and GMRES-IR, the number of parenthetical elements gives the total number of refinement steps, and element i in the list gives the number of GMRES iterations performed in refinement step i. For example, $(3, 4)$ indicates that 2 refinement steps were performed, the first of which took 3 GMRES iterations and the second of which took 4. For the MSIR column, the data for SIR, SGMRES-IR, and GMRES-IR is comma-separated. For example, $2, (2)$ indicates that 2 SIR steps were performed, the algorithm then switched to SGMRES-IR, and performed 1 SGMRES-IR step which required 2 GMRES iterations. Since there is no second set of parentheses, this indicates that there was no switch to GMRES-IR. In all columns, a dash denotes that the algorithm diverged or made no progress; to enable a fair comparison, in our experiments we set i_{max} to be a very high value (here $i_{max} = 2000$) in order to allow all approaches that eventually converge sufficient time to do so.

Table 9.2: Number of SIR, SGMRES-IR, GMRES-IR, and MSIR steps with the number of GMRES iterations for each SGMRES-IR and GMRES-IR step for initial precisions $(u_f, u, u_r) =$ (single, double, quad).

$\kappa_\infty(A)$	$\kappa_2(A)$	SIR	SGMRES-IR	GMRES-IR	MSIR
$2 \cdot 10^2$	10^1	2	(2)	(2)	2
$2 \cdot 10^6$	10^5	5	(2,3)	(2)	2, (2)
$2 \cdot 10^{17}$	10^{16}	-	(3,3,3,4,4,4,4,8,100,100,50,12,12,5,4,5,4,4,4,4)	(3,4)	2, (3,3), (3,4)

From Figure 9.1, we can see that for well-conditioned matrices, SIR quickly converges to the solution. When the condition number increases closer to $u_f^{-1} = (5.96 \cdot 10^8)^{-1} \approx 1.7 \cdot 10^7$ as in Figure 9.2, however, SIR convergence begins to slow down, and MSIR switches to SGMRES-IR, which then converges in one step. When the matrix becomes very ill conditioned as in Figure 9.3, then SIR diverges, SGMRES-IR converges very slowly, and thus MSIR makes the second switch to GMRES-IR.

This illustrates the benefit of the multistage approach. In the case that the problem is well conditioned enough that SIR suffices, MSIR will only use SIR. For the $\kappa_2(A) = 10^1$ case, MSIR behaves exactly the same as SIR, and is thus less expensive than SGMRES-IR and GMRES-IR since a single GMRES iteration is more expensive than an SIR step. In the extremely ill-conditioned case, both SIR and SGMRES-IR fail to converge or convergence is too slow. MSIR however does converge quickly, although at roughly double the cost of GMRES-IR. This is the

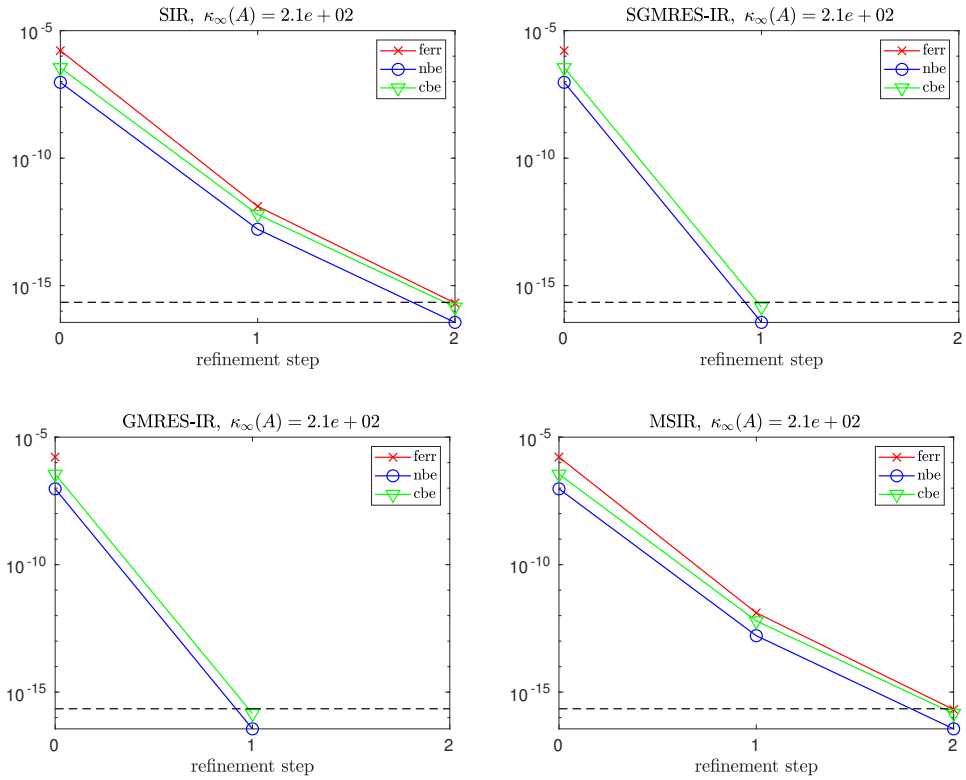

Figure 9.1: Convergence of errors for a 100×100 random dense matrix with $\kappa_2(A) = 10^1$ using SIR (top left), SGMRES-IR (top right), GMRES-IR (bottom left), and MSIR (bottom right), with initial precisions $(u_f, u, u_r) =$ (single, double, quad). Note that for SGMRES-IR and GMRES-IR, the forward error is measured as 0 (in double precision) after one refinement step.

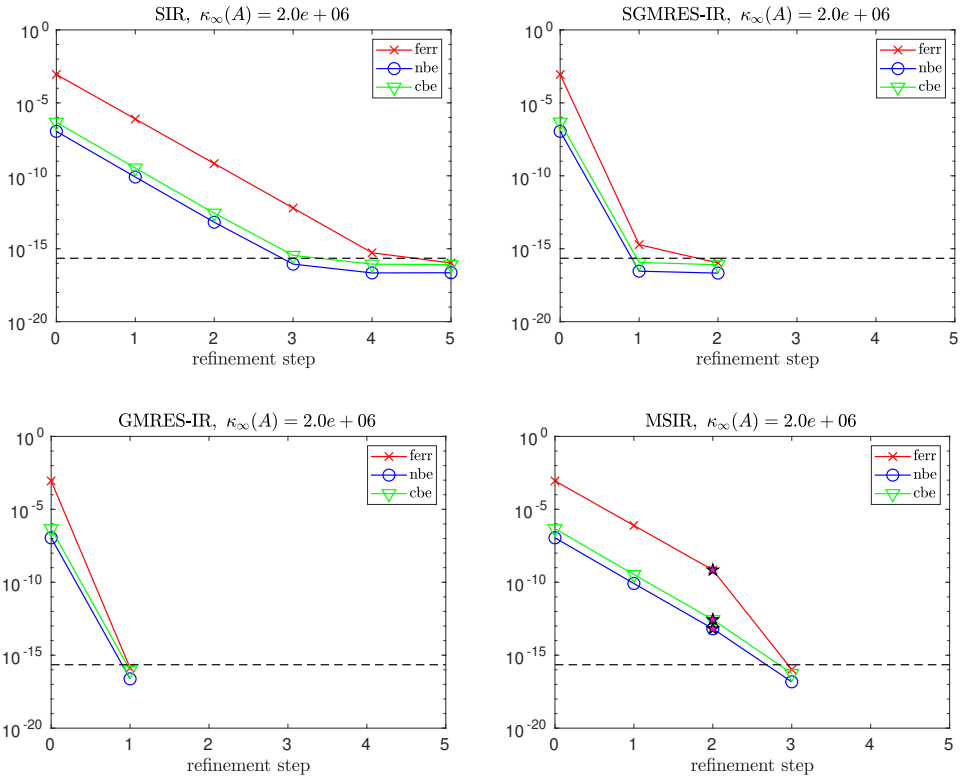

Figure 9.2: Convergence of errors for a 100×100 random dense matrix with $\kappa_2(A) = 10^5$ using SIR (top left), SGMRES-IR (top right), GMRES-IR (bottom left), and MSIR (bottom right), with initial precisions $(u_f, u, u_r) =$ (single, double, quad).

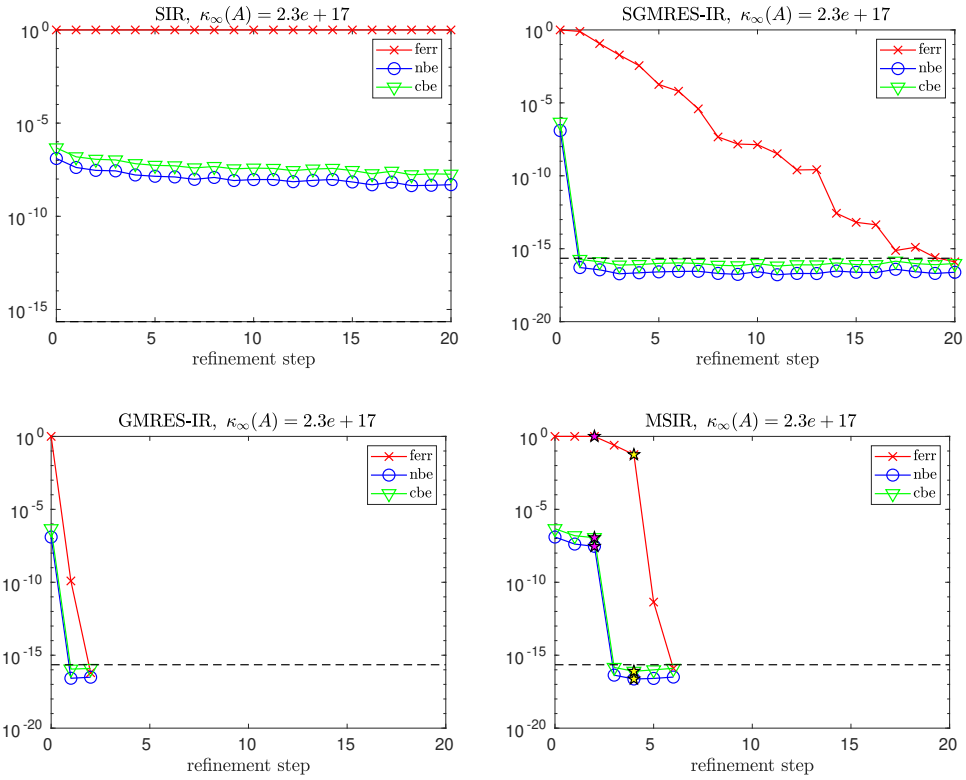

Figure 9.3: Convergence of errors for a 100×100 random dense matrix with $\kappa_2(A) = 10^{16}$ using SIR (top left), SGMRES-IR (top right), GMRES-IR (bottom left), and MSIR (bottom right), with initial precisions $(u_f, u, u_r) =$ (single, double, quad).

inherent tradeoff. Compared to GMRES-IR, we expect MSIR to be less expensive when the problem is well or reasonably well conditioned and thus only SIR or SGMRES-IR are used (see a comparison of costs in Table 9.1). For cases where the problem is extremely ill conditioned relative to the working precision and GMRES-IR converges, we expect MSIR to be a constant factor more expensive than GMRES-IR. Compared to SIR and SGMRES-IR, the benefit is clear: when SIR and/or SGMRES-IR converges reasonably quickly, MSIR will also converge at roughly the same cost, but MSIR can converge for problems where SIR and SGMRES-IR may not.

9.1.1 Algorithm details

We now discuss the MSIR algorithm (Algorithm 21) in more detail. The algorithm uses i to denote the global number of refinement steps of any type. The `iter` variable counts the number of refinement steps of the current refinement variant; e.g., it will first count the number of SIR steps, and is then reset to 0 if the algorithm switches to SGMRES-IR. Unlike the single-stage algorithms, here the input parameter i_{max} specifies the maximum number of refinement steps of each type, meaning, e.g., we will perform up to i_{max} SIR steps before switching to SGMRES-IR.

Stopping criteria and convergence detection

In Demmel et al. [2006], Demmel et al. analyze an iterative refinement scheme in which extra precision is used in select computations and provide reliable normwise and componentwise error bounds for the computed solution as well as stopping criteria. They devise a new approach that adaptively changes the precision with which the approximate solution is stored based on monitoring the convergence; if consecutive corrections to the approximate solution are not decreasing at a sufficient rate, the precision of the approximate solution is increased. This has the effect of improving the componentwise accuracy.

Following this strategy for monitoring the behavior of iterative refinement from Demmel et al. [2006], the MSIR algorithm will switch to the next variant if any of the following conditions applies:

1. $\frac{\|d_{i+1}\|_\infty}{\|x_i\|_\infty} \leq u$ (the correction d_{i+1} changes solution x_i too little),
2. $\frac{\|d_{i+1}\|_\infty}{\|d_i\|_\infty} \geq \rho_{thresh}$ (convergence slows down sufficiently),
3. `iter` $\geq i_{max}$ (too many iterations of a particular variant have been performed), and,
4. $k_{GMRES} \geq k_{max}$ (too many GMRES iterations are performed in one step of SGMRES-IR or GMRES-IR)

where d_i is the correction of the solution x_i, $\rho_{thresh} \in \mathbb{R}^+$ is a threshold for convergence and $\rho_{thresh} < 1$, $i_{max} \in \mathbb{N}^+$ is the maximum number of iterations, $k_{max} \in \mathbb{N}^+$ is the maximum number of GMRES iterations performed per SGMRES-IR step, and u is the working precision.

As is explained in Demmel et al. [2006], the analyses of Bowdler in 1966 (Bowdler et al. [1966]) and Moler in 1967 (Moler [1967]) showed that $\|d_{i+1}\|_\infty$ should decrease by a factor of at most $\rho = O(u_f)\kappa_\infty(A)$ at each step, and thus the solution x_{i+1} should converge like a geometric sum, meaning that $\sum_{j=i+1}^{\infty} \|d_i\|_\infty \leq \|d_{i+1}\|_\infty/(1-\rho)$. This geometric convergence will end when rounding errors become significant. So if we have $\|d_{i+1}\|_\infty/\|d_i\|_\infty \geq \rho_{thresh}$, it either means that (1) convergence has slowed down due to rounding errors, or (2) convergence is slow from the beginning as the quantity ρ is close to 1 (meaning that the problem is too ill conditioned with respect to the precision u_f). In the case that $\|d_{i+1}\|_\infty > \|d_i\|_\infty$, this indicates that the iterative refinement process is diverging, again because the problem is too ill conditioned with respect to the precision u_f.

We reiterate that the fourth condition above only applies to the switch from SGMRES-IR to GMRES-IR, or GMRES-IR to SIR with increased precision(s). We note that as in line 10 in Algorithm 21, in case SIR produces a correction d_{i+1} containing Inf or NaN, we immediately switch to SGMRES-IR without performing the solution update. This situation can arise as a result of performing the triangular solves in low precision. We have implemented a simple scaling to help avoid this situation, although in certain scenarios it may still occur; see further details in Section 9.1.1. We note that this is less of a concern for GMRES-based approaches, since within GMRES the triangular solves are performed in either precision u (SGMRES-IR) or u^2 (GMRES-IR).

Moreover, the convergence detection in lines 14, 25, and 36 in Algorithm 21 can be performed using the normwise relative error estimate discussed in Demmel et al. [2006],

$$max\left\{\frac{\frac{\|d_{i+1}\|_\infty}{\|x_i\|_\infty}}{1-\rho_{max}}, \gamma u\right\} \approx \frac{\|x_i - x\|_\infty}{\|x\|_\infty}, \tag{9.1}$$

where $\gamma = max(10, \sqrt{n})$, n is the size of the matrix A, and $\rho_{max} := max_{j \leq i} \frac{\|d_{j+1}\|_\infty}{\|d_j\|_\infty}$ is the maximum ratio of successive corrections. The quantity $(\|d_{i+1}\|_\infty/\|x_i\|_\infty)/(1-\rho_{max})$ is stored in Algorithm 21 as the quantity ϕ_i. If the condition in lines 14, 25, and 36 is triggered, one could then test for convergence by checking whether both $\phi_i \leq \sqrt{n}u$ and $\phi_i \geq 0$, and if so, declare convergence (note that $\phi_i < 0$ indicates divergence). If convergence is not detected, we must also decide whether to keep the current approximate solution or reset the approximate solution to x_0, since this will be input to the next stage. We test if $\phi_i > \phi_0$, and if so, we reset the initial solution. We note that other measures, such as for the componentwise error, could also be used to detect convergence.

We note that the criteria used to determine whether the current algorithm should quit iterating are one iteration behind; in other words, we can only detect convergence (or non-convergence) in step i after computing the correction term in step $i+1$. For this reason, the MSIR algorithm will generally compute at least two steps of a particular variant before deciding to switch. The two cases where fewer than 2 steps of a variant will occur are 1) when the computed update contains Infs or NaNs (as in line 10 in Algorithm 21), in which case we switch without updating the current solution, and 2) when, for SGMRES-IR and GMRES-IR, the number of GMRES iterations exceeds k_{max}, since this is detectable immediately in the current step.

In Demmel et al. [2006], to determine ρ_{thresh}, the authors define 'cautious' and

'aggressive' settings. Cautious settings produce maximally reliable error bounds for well-conditioned problems, whereas aggressive ones will lead to more steps on the hardest problems and usually, but not always, give error bounds within a factor of 100 of the true error. For cautious mode, the authors suggest ρ_{thresh} should be set to 0.5, which was used by Wilkinson (Wilkinson [1963]), and for aggressive mode, 0.9. In our experiments we always use the cautious setting, but we note that ρ_{thresh}, i_{max}, and k_{max} should be set according to the relative costs of the different refinement schemes in practice.

Scaling

Because of the smaller range of half precision, simply rounding higher precision quantities to a lower precision can cause overflow, underflow, or the introduction of subnormal numbers. In Higham and Pranesh [2019], the authors develop a new algorithm for converting single and double precision quantities to half precision. This algorithm involves performing a two-sided scaling for equilibration and then an additional scaling is performed to make full use of the range of half precision before finally rounding to half precision. In our algorithm, when half precision is used for the factorization, we first attempt an LU factorization without scaling. We then test whether the resulting L and U factors contain Inf or NaN; if so (marked with a * in the tables in Section 9.2), we retry the LU factorization in line 1 using the two-sided scaling algorithm of Higham and Pranesh [2019]. In all cases, regardless of what precision is used for the factorization, after computing x_0 in line 2 we test whether the initial solution contains Inf or NaN; if so, we simply use the zero vector as the initial approximate solution and proceed.

We also incorporate scaling in each refinement step. After computing the residual r_i, we scale the result to obtain $r_i = r_i/\|r_i\|_\infty$ (line 5 in Algorithm 21). This scaling is then undone when we update the approximate solution, via $x_{i+1} = x_i + \|r_i\|_\infty d_{i+1}$ (lines 11, 22, and 33 in Algorithm 21). As long as $1/\|A\|_\infty$ does not underflow and $\|A^{-1}\|_\infty$ does not overflow, then this scaling avoids the largest element of d_{i+1} overflowing or underflowing.

9.1.2 Error bounds for different variants

There are various scenarios on the usage of precisions which will yield different error bounds and different constraints on condition number. Besides three-precision variants, two precisions or a fixed precision can be used in the algorithms discussed in this study. We summarize the convergence criteria for the precision combinations used in our approach and refer the reader to Carson and Higham [2018] and Amestoy et al. [2021] for more general bounds. In particular, we assume that $u_f \geq u$ and $u_r \leq u^2$. We also restrict ourselves to IEEE precisions (see Table 1), although we note that alternative formats like bfloat16 (Intel Corporation [2018]) could also be used.

Under the assumptions that $u_f \geq u$ and $u_r \leq u^2$, for SIR, both the relative forward and backward errors are guaranteed to converge to the level of the working precision when $\kappa_\infty(A) < u_f^{-1}$. For SGMRES-IR and GMRES-IR, the constraints for convergence of forward and backward errors differ. Summarizing the analysis in Amestoy et al. [2021], for our particular SGMRES-IR variant, the constraint on convergence of the backward error is $\kappa_\infty(A) < u^{-1/3}u_f^{-1/3}$ and

Table 9.3: Constraints on $\kappa_\infty(A)$ for which the relative forward and normwise backward errors are guarantee to converge to the level u for a given combination of precisions for the different variants of IR.

u_f	u	u_r	SIR	SGMRES-IR	GMRES-IR
half	single	double	$2 \cdot 10^3$	$4 \cdot 10^4$	$8 \cdot 10^6$
single	single	double	$2 \cdot 10^7$	$2 \cdot 10^7$	$7 \cdot 10^{10}$
half	double	quad	$2 \cdot 10^3$	$3 \cdot 10^7$	$2 \cdot 10^{11}$
single	double	quad	$2 \cdot 10^7$	$1 \cdot 10^{10}$	$2 \cdot 10^{15}$
double	double	quad	$9 \cdot 10^{15}$	$9 \cdot 10^{15}$	$9 \cdot 10^{23}$

the constraint on the convergence of the forward error is $\kappa_\infty(A) < u^{-1/3}u_f^{-2/3}$. It is interesting to note that, as an artifact of the analysis, the constraint for convergence of the backward error for this variant is more strict than that for the forward error. However, since the backward error is bounded by the forward error, the constraint for both backward and forward error to converge to the level u can be given as $\kappa_\infty(A) < u^{-1/3}u_f^{-2/3}$.

In Carson and Higham [2018], the constraint for convergence of the backward error to level u in GMRES-IR given as is $\kappa_\infty(A) < u^{-1}$. However, the authors in Amestoy et al. [2021] point out that this bound relies on assumptions that may be overly optimistic, and revise this to the tighter constraint $\kappa_\infty(A) < u^{-1/2}u_f^{-1/2}$. The constraint for the convergence of the forward error to this level in GMRES-IR given in Carson and Higham [2018] is $\kappa_\infty(A) < u^{-1/2}u_f^{-1}$. Thus the tighter constraint for the convergence of backward error in Amestoy et al. [2021] is again more strict than the constraint for the convergence of the forward error, and since the backward error is bounded by the forward error, we can take $\kappa_\infty(A) < u^{-1/2}u_f^{-1}$ to be the constraint for the convergence of both backward and forward error to level u in GMRES-IR.

In Table 9.3 we quantify these constraints on $\kappa_\infty(A)$ required for convergence of the normwise relative backward and forward errors to the level of the working precision for various precision combinations. To compute the constraints on $\kappa_\infty(A)$, we have used the unit roundoff values in Table 1.

9.2 Numerical experiments

In this section we present numerical experiments performed in MATLAB on a number of synthetic and real-world matrices from SuiteSparse (Davis and Hu [2011]) for MSIR and single-stage iterative refinement variants in three precisions. The problems we test are small and are meant to demonstrate the numerical behavior of the algorithms. Performance results for SIR and GMRES-based variants for larger problems on modern GPUs can be found in, e.g., Haidar et al. [2018].

For quadruple precision, we use the Advanpix multiprecision computing toolbox for MATLAB (Advanpix LLC.). To simulate half precision floating point arithmetic, we use the `chop` library and associated functions from Higham and Pranesh [2019], available in the repositories `https://github.com/higham/chop` and `https://github.com/SrikaraPranesh/LowPrecision_Simulation`. For single and double precision we use the built-in MATLAB datatypes. All experiments were performed on a computer with an Intel Core i7-9750H processor

with 12 CPUs and 16 GB RAM with OS system Ubuntu 20.04.1 using MATLAB 2020a. Our MSIR algorithm and associated functions are available through the repository `https://github.com/edoktay/msir`, which includes scripts for generating the data and plots in this work.

In all experiments, we use $i_{max} = 2000$ and $\rho_{thresh} = 0.5$. We have chosen to use a very high value for i_{max} in both MSIR and single-stage algorithms as it allows us to see the true convergence behavior of each algorithm (and the benefits of MSIR in cases where convergence for single-stage algorithms happens eventually but is very slow). Of course, in practice, one would use a much smaller value.

For the GMRES convergence tolerance, we use $\tau = 10^{-6}$ when the working precision is single and $\tau = 10^{-10}$ when the working precision is double. We set $k_{max} = 0.1n$. For purposes of discerning the actual attainable accuracy of MSIR, in the experiments we explicitly compute and plot the computed forward and backward errors and use these as stopping criteria rather than the error estimate given by (9.1). For a fair numerical comparison, we also apply the scalings discussed in Section 9.1.1 in SIR, SGMRES-IR, and GMRES-IR.

For each variant, we compare the number of refinement steps required for forward and backward errors to reach the level of accuracy corresponding to the initial working precision. For GMRES-based approaches and MSIR, the number of GMRES iterations per step is given parenthetically (see the explanation in Section 9.1). For the MSIR results, a semicolon indicates a precision switch: the factorization precision is doubled (and other precisions increased if necessary to ensure $u_f \geq u$ and $u_r \leq u^2$) and the algorithm restarts with SIR. A dash (-) in the tables indicates that forward and/or backward errors are not decreasing. Select convergence plots for MSIR are presented in figures while the number of steps and iterations for all variants are shown in tables.

Again, the red, blue, and green lines in the figures show the behavior of the forward error `ferr` (red), normwise relative backward error `nbe` (blue), and componentwise relative backward error `cbe` (green). The dotted black line shows the value of the initial working precision u. Switches in MSIR are denoted by stars. A magenta star indicates a switch from SIR to SGMRES-IR, a yellow start indicates a switch from SGMRES-IR to GMRES-IR, and a cyan star indicates switch from GMRES-IR to SIR with increased precision(s).

9.2.1 Random dense matrices

We first test our algorithm on random dense matrices of dimension $n = 100$ generated in MATLAB via the command `gallery('randsvd',n,kappa(i),2)` (Section 9.2.1) and the command `gallery('randsvd',n,kappa(i),3)` (Section 9.2.1), where `kappa` is the array of the desired 2-norm condition numbers $\kappa_2(A) = \{10^1, 10^2, 10^4, 10^5, 10^7, 10^9, 10^{11}, 10^{14}\}$, and `2` and `3` stand for the modes which dictate the singular value distribution. Mode 2 generates matrices with one singular value equal to $1/\kappa_2(A)$ and the rest of the singular values equal to 1. Mode 3 generates matrices with geometrically distributed singular values. Mode 3 represents a difficult case for GMRES-based iterative refinement methods. It is known that a cluster of singular values near zero causes stagnation of GMRES, and a low precision preconditioner may fail to effectively shift this cluster away from the

origin. We test the algorithms using initial precision combinations (u_f, u, u_r) = (single, double, quad), (u_f, u, u_r) = (half, single, double), and (u_f, u, u_r) = (half, double, quad). For simplicity, b is chosen to be a vector of normally distributed numbers generated by the MATLAB command `randn` for all experiments in this section. For reproducibility, we use the MATLAB command `rng(1)` before generating each linear system.

Random dense matrices with one small singular value

Table 9.4 shows the convergence behavior of SIR, SGMRES-IR, GMRES-IR, and MSIR for matrices generated via the MATLAB command `gallery('randsvd',n,kappa(i),2)` using initial precisions (u_f, u, u_r) = (single, double, quad). From Table 9.4, we can see that SIR no longer converges once $\kappa_\infty(A)$ exceeds 10^{11}. As $\kappa(A)$ increases, the convergence rate of SIR slows down. At the extreme case of $\kappa_2(A) = 10^9$, SIR requires 205 steps to converge! In this case, MSIR thus demonstrates a significant improvement over SIR. Even in cases where only 5 SIR steps are required, notice that the MSIR algorithm deems this too slow and begins switching to SGMRES-IR after 2 SIR steps. SGMRES-IR converges in just a few refinement steps with a small numbers of GMRES iterations per step except for the most ill-conditioned problem. For the most ill-conditioned matrix, SGMRES-IR alone requires 7 steps to converge. MSIR in this case switches twice: from SIR to SGMRES-IR, and then finally to GMRES-IR. The same is observed when $\kappa_2(A) = 10^{11}$. We plot the convergence trajectories of MSIR for the $\kappa_2(A) = 10^4$ and $\kappa_2(A) = 10^{14}$ problems in Figure 9.4.

Table 9.4: Number of SIR, SGMRES-IR, GMRES-IR, and MSIR steps with the number of GMRES iterations for each SGMRES-IR and GMRES-IR step for random dense matrices (mode 2) with various condition numbers $\kappa_\infty(A)$, $\kappa_2(A)$, using initial precisions (u_f, u, u_r) = (single, double, quad).

$\kappa_\infty(A)$	$\kappa_2(A)$	SIR	SGMRES-IR	GMRES-IR	MSIR
$2 \cdot 10^2$	10^1	2	(2)	(2)	2
$2 \cdot 10^3$	10^2	3	(2)	(2)	2, (2)
$2 \cdot 10^5$	10^4	5	(2,3)	(2)	2, (2)
$2 \cdot 10^6$	10^5	5	(2,3)	(2)	2, (2)
$2 \cdot 10^8$	10^7	19	(2,3)	(2,3)	2, (2,3)
$2 \cdot 10^{10}$	10^9	205	(2,2)	(2,3)	2, (2,3)
$2 \cdot 10^{12}$	10^{11}	-	(3,3,4)	(3,4)	2, (3,3), (3)
$2 \cdot 10^{15}$	10^{14}	-	(3,3,3,4,4,4,4)	(3,4)	2, (3,3), (3,4)

In Table 9.5, we show data for experiments with (u_f, u, u_r) = (half, single, double). In this case, as predicted in Table 9.3, SIR fails to converge once $\kappa_\infty(A) > 2 \cdot 10^5$. Moreover, it is seen that MSIR switches to SGMRES-IR even for relatively well-conditioned matrices for which SIR still converges (i.e., for $\kappa_2(A) = 10^2$). Although this switch is technically not necessary, we argue that the difference in cost versus SIR will be a constant factor (SIR requires 3 LU solves in precision u_f whereas here SGMRES-IR requires 2 LU solves in precision u_f and 3 in precision u; see Table 9.1). SGMRES-IR works well up to condition number $\kappa_\infty(A) = 2 \cdot 10^6$, and thus MSIR only performs one switch in these cases.

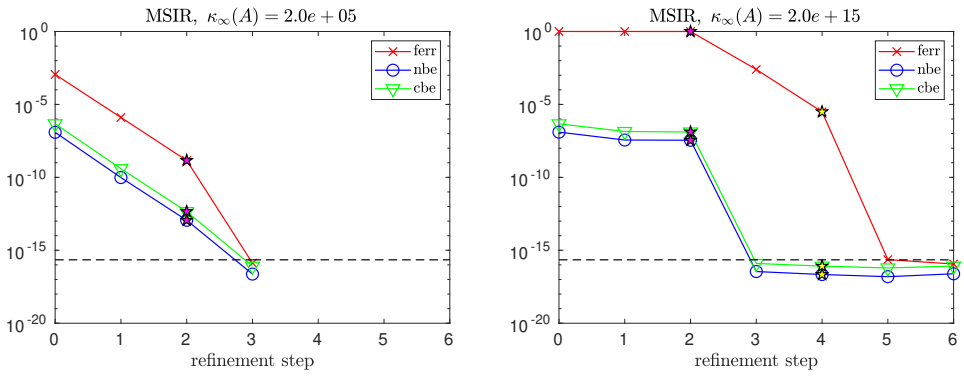

Figure 9.4: Convergence of errors in MSIR for random dense matrices (mode 2) with $\kappa_2(A) = 10^4$ (left), and $\kappa_2(A) = 10^{14}$ (right) for initial precisions (u_f, u, u_r) = (single, double, quad); see also Table 9.4.

It is interesting to notice that the theory from Amestoy et al. [2021] says that we can only expect SGMRES-IR to converge as long as $\kappa_\infty(A) < 4 \cdot 10^4$; see also Table 9.3. However, as this and other experiments in this work show, SGMRES-IR as well as GMRES-IR actually often perform beyond the limits given by the analysis. This confirms our motivation for the multistage MSIR approach; we cannot rely entirely on the existing theoretical bounds to determine whether or not an algorithm will be effective.

For the condition number $\kappa_\infty(A) = 2 \cdot 10^8$, SGMRES-IR convergence eventually slows down, and as a result MSIR switches a second time to GMRES-IR. Beyond this limit, for $\kappa_\infty(A) = 2 \cdot 10^8$, GMRES-IR still converges (despite that the theory only guarantees that GMRES-IR will work up to condition number $8 \cdot 10^6$). Starting at $\kappa_\infty(A) = 2 \cdot 10^{10}$, GMRES convergence slows down significantly (requiring n iterations per refinement step), and thus MSIR switches to precisions (u_f, u, u_r) = (single, single, double) and starts again with SIR. We show plots of MSIR convergence behavior for the cases $\kappa_2(A) \in \{10^1, 10^2, 10^7, 10^{14}\}$ in Figure 9.5.

Table 9.5: Number of SIR, SGMRES-IR, GMRES-IR, and MSIR steps with the number of GMRES iterations for each SGMRES-IR and GMRES-IR step for random dense matrices (mode 2) with various condition numbers $\kappa_\infty(A)$, $\kappa_2(A)$, using initial precisions (u_f, u, u_r) = (half, single, double).

$\kappa_\infty(A)$	$\kappa_2(A)$	SIR	SGMRES-IR	GMRES-IR	MSIR
$2 \cdot 10^2$	10^1	3	(3)	(3)	3
$2 \cdot 10^3$	10^2	3	(3)	(3)	2, (3)
$2 \cdot 10^5$	10^4	19	(3,4)	(3,4)	2, (3,4)
$2 \cdot 10^6$	10^5	-	(3,4)	(3,4)	2, (3,4)
$2 \cdot 10^8$	10^7	-	(5,12,9,17,5,5,5)	(5,5)	2, (5,10), (10)
$2 \cdot 10^{10}$	10^9	-	-	(100,100,100,22,11,3)	2, (5,9), (10); 2, (2,2,2), (2,3,3)
$2 \cdot 10^{12}$	10^{11}	-	-	(100,63,10)	2, (10), (10); 2, (2,2), (2)
$2 \cdot 10^{15}$	10^{14}	-	-	(100,63,10)	2, (10), (10); 2, (2,2), (2)

Finally, in Table 9.6, we test initial precisions (u_f, u, u_r) = (half, double, quad). Except for SIR, which again fails to converge once $\kappa_\infty(A) > 2 \cdot 10^5$, all algorithms converge for $\kappa_\infty(A) < 10^{16}$. Even in cases where SIR converges,

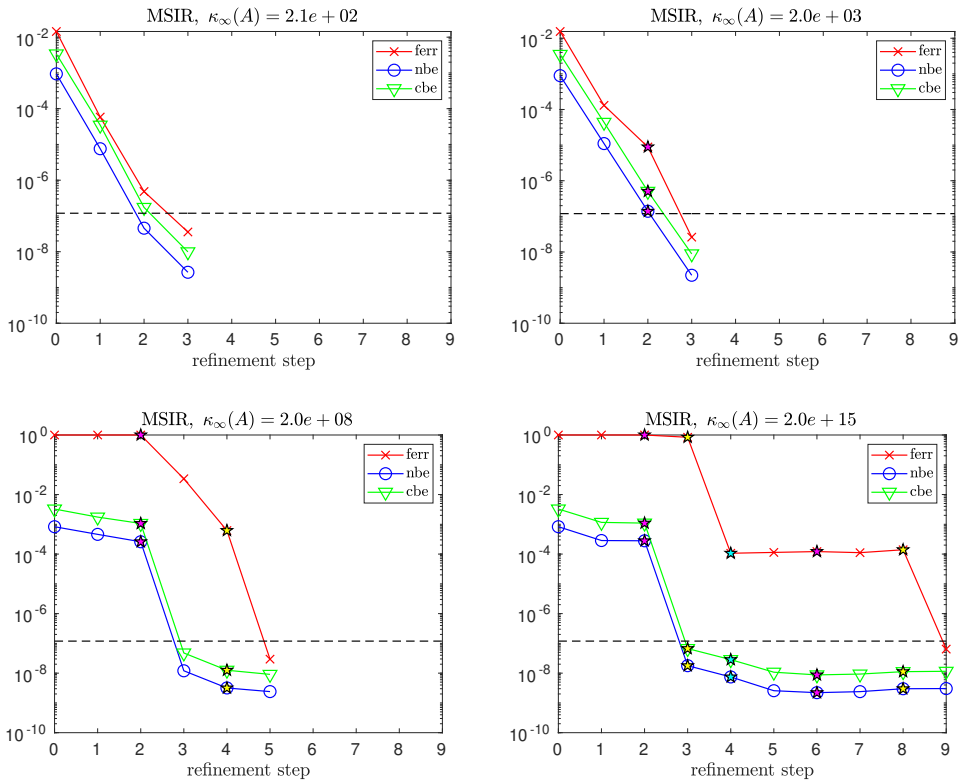

Figure 9.5: Convergence of errors in MSIR for random dense matrices (mode 2) with $\kappa_2(A) = 10^1$ (top left), $\kappa_2(A) = 10^2$ (top right), $\kappa_2(A) = 10^7$ (bottom left), and $\kappa_2(A) = 10^{14}$ (bottom right) for initial precisions (u_f, u, u_r) = (half, single, double); see also Table 9.5.

the SIR convergence is slow enough that MSIR always switches to SGMRES-IR. MSIR only performs the second switch to GMRES-IR for the two most ill-conditioned problems (although this switch is actually unnecessary at least for $\kappa_2(A) = 10^{11}$; see also the convergence trajectory in the bottom left plot in Figure 9.6). Again, this emphasizes the need for a multistage, adaptive approach; if we were to go by the theoretical bounds, we would not expect SGMRES-IR to work beyond $\kappa_\infty(A) = 3 \cdot 10^7$, but here it works well even for a matrix with a condition number five orders of magnitude greater. For the most ill-conditioned case, MSIR also performed a precision switch from (u_f, u, u_r) = (half, double, quad) to (u_f, u, u_r) = (single, double, quad) since GMRES-IR required too many GMRES iterations. We plot convergence trajectories for MSIR for the problems with $\kappa_2(A) \in \{10^1, 10^9, 10^{11}, 10^{14}\}$ in Figure 9.6.

Random dense matrices with geometrically distributed singular values

We now test the convergence behavior of the IR variants for matrices generated via the MATLAB command `gallery('randsvd',n,kappa(i),3)`. As mentioned, we expect that these are more challenging problems for the GMRES-based iterative refinement schemes since they contain clusters of eigenvalues close to the origin which may fail to be effectively shifted away from the origin by a low precision preconditioner.

Table 9.6: Number of SIR, SGMRES-IR, GMRES-IR, and MSIR steps with the number of GMRES iterations for each SGMRES-IR and GMRES-IR step for random dense matrices (mode 2) with various condition numbers $\kappa_\infty(A)$, $\kappa_2(A)$, using initial precisions (u_f, u, u_r) = (half, double, quad).

$\kappa_\infty(A)$	$\kappa_2(A)$	SIR	SGMRES-IR	GMRES-IR	MSIR
$2 \cdot 10^2$	10^1	7	(5,5)	(5,5)	2, (5)
$2 \cdot 10^3$	10^2	9	(5,5)	(5,5)	2, (5)
$2 \cdot 10^5$	10^4	47	(5,6)	(5,6)	2, (5,6)
$2 \cdot 10^6$	10^5	-	(5,6)	(5,6)	2, (5,6)
$2 \cdot 10^8$	10^7	-	(6,7)	(6,7)	2, (6,7)
$2 \cdot 10^{10}$	10^9	-	(7,8)	(7,8)	2, (7,8)
$2 \cdot 10^{12}$	10^{11}	-	(8,8,9)	(8,9)	2, (8,8), (9)
$2 \cdot 10^{15}$	10^{14}	-	(100,100,100,25,10,10)	(100,32,10)	2, (10), (10); 3, (4,4,4)

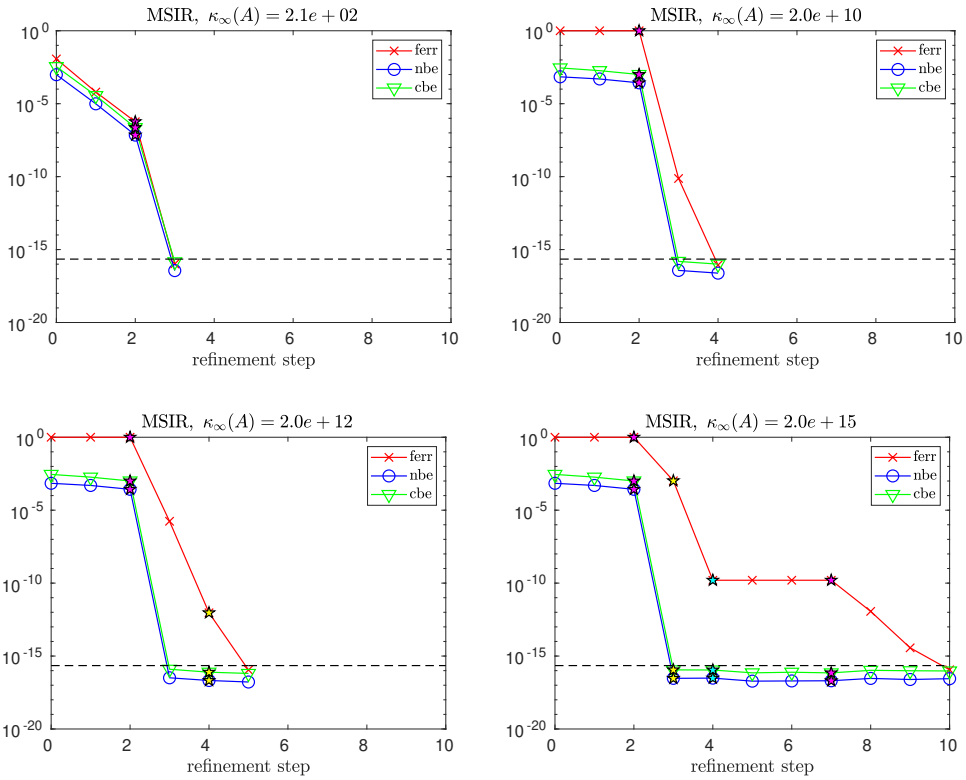

Figure 9.6: Convergence of errors in MSIR for random dense matrices (mode 2) with $\kappa_2(A) = 10^1$ (top left), $\kappa_2(A) = 10^9$ (top right), $\kappa_2(A) = 10^{11}$ (bottom left), and $\kappa_2(A) = 10^{14}$ (bottom right) for initial precisions (u_f, u, u_r) = (half, double, quad); see also Table 9.6.

In Table 9.7 we test initial precisions (u_f, u, u_r) = (single, double, quad). Here, SIR fails to converge once $\kappa_\infty(A)$ exceeds 10^9. Once $\kappa_\infty(A)$ exceeds 10^9, the number of GMRES iterations required per SGMRES-IR and GMRES-IR step begins to increase dramatically. This is expected since there is a cluster of eigenvalues remaining close to the origin even after low precision preconditioning. Notice that in these cases, MSIR performs the second switch to GMRES-IR after just one SGMRES-IR step and then performs a precision switch from (u_f, u, u_r)

= (single, double, quad) to (u_f, u, u_r) = (double, double, quad) after just one GMRES-IR step, since the number of GMRES iterations per refinement step for both SGMRES-IR and GMRES-IR exceeds our specified value of $k_{max} = 0.1n$. We show plots for $\kappa_2(A) \in \{10^1, 10^4, 10^9, 10^{14}\}$ in Figure 9.7.

Table 9.7: Number of SIR, SGMRES-IR, GMRES-IR, and MSIR steps with the number of GMRES iterations for each SGMRES-IR and GMRES-IR step for random dense matrices having geometrically distributed singular values (mode 3) with various condition numbers $\kappa_\infty(A)$, $\kappa_2(A)$, using initial precisions (u_f, u, u_r) = (single, double, quad).

$\kappa_\infty(A)$	$\kappa_2(A)$	SIR	SGMRES-IR	GMRES-IR	MSIR
$2 \cdot 10^2$	10^1	2	(2)	(2)	2
10^3	10^2	2	(2)	(2)	2
$9 \cdot 10^4$	10^4	3	(3)	(3)	2, (3)
$8 \cdot 10^5$	10^5	4	(4)	(4)	2, (4)
$7 \cdot 10^7$	10^7	13	(7,7)	(7,7)	2, (6,7)
$6 \cdot 10^9$	10^9	-	(22,23,25)	(22,26)	2, (10), (10); 2
$6 \cdot 10^{11}$	10^{11}	-	(53,54,55)	(53,55)	2, (10), (10); 2, (2)
$5 \cdot 10^{14}$	10^{14}	-	(84,84,84,85,85)	(84,88)	2, (10), (10); 2, (3,3,3)

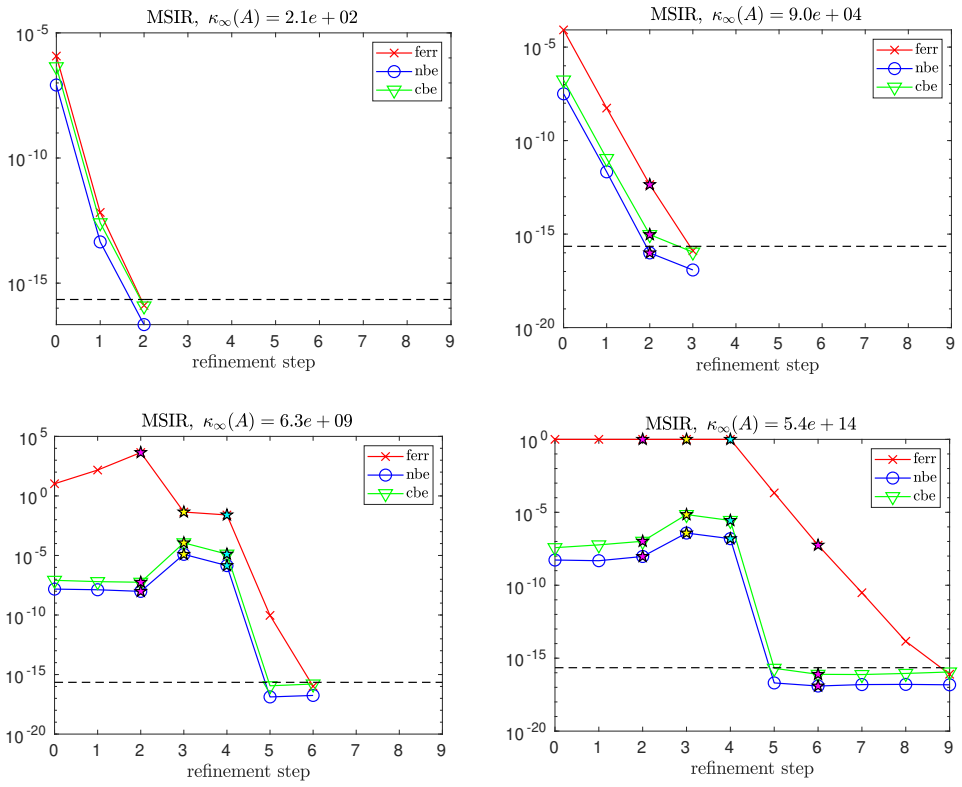

Figure 9.7: Convergence of errors in MSIR for random dense matrices having geometrically distributed singular values (mode 3) with $\kappa_2(A) = 10^1$ (top left), $\kappa_2(A) = 10^4$ (top right), $\kappa_2(A) = 10^9$ (bottom left), and $\kappa_2(A) = 10^{14}$ (bottom right) for initial precisions (u_f, u, u_r) = (single, double, quad); see also Table 9.7.

The story for initial precisions (u_f, u, u_r) = (half, single, double), shown in

Table 9.8, is similar. Once the condition number exceeds around $\sqrt{u}$, the number of GMRES iterations required per refinement step increases significantly for both SGMRES-IR and GMRES-IR. Again, this means that MSIR switches from SGMRES-IR to GMRES-IR and then performs a precision switch (to (u_f, u, u_r) = (single, single, double)) since k_{max} is exceeded. When κ_∞ exceeds 10^9, however, using single precision for the factorization is not enough to combat the slow GMRES convergence, and thus a second precision switch occurs, switching to (u_f, u, u_r) = (double, double, quad). For the most ill-conditioned problem for which SGMRES-IR converges, MSIR does significantly fewer GMRES iterations in total than SGMRES-IR (although we use higher precision for u_f (single), the computational cost will still less considering the total number of GMRES steps performed in SGMRES-IR). Notice also that for $\kappa_2(A) \geq 10^5$, MSIR performs no SIR steps at the initial precision setting (u_f, u, u_r) = (half, single, double) since an Inf or NaN is detected in the first correction term. We plot MSIR convergence for $\kappa_2(A) \in \{10^1, 10^4, 10^7, 10^{14}\}$ in Figure 9.8.

Table 9.8: Number of SIR, SGMRES-IR, GMRES-IR, and MSIR steps with the number of GMRES iterations for each SGMRES-IR and GMRES-IR step for random dense matrices having geometrically distributed singular values (mode 3) with various condition numbers $\kappa_\infty(A)$, $\kappa_2(A)$, using initial precisions (u_f, u, u_r) = (half, single, double). For $\kappa_2(A) = 10^{11}$, GMRES-IR used (100,100,100,100,100,100,100,100,100,100,99,95,100,97) iterations.

$\kappa_\infty(A)$	$\kappa_2(A)$	SIR	SGMRES-IR	GMRES-IR	MSIR
$2 \cdot 10^2$	10^1	3	(3)	(3)	2, (3)
10^3	10^2	4	(4)	(4)	2, (4)
$9 \cdot 10^4$	10^4	856	(13,14)	(13,14)	2, (10), (10)
$8 \cdot 10^5$	10^5	-	(38,40)	(38,41)	0, (10), (10); 3
$7 \cdot 10^7$	10^7	-	(100,100,100,100,100,83)	(100,100)	0, (10), (10); 2, (5,5,5)
$6 \cdot 10^9$	10^9	-	-	(100,100,100,100)	0, (10), (10);1, (10), (10); 1
$6 \cdot 10^{11}$	10^{11}	-	-	(100,...) see caption.	0, (10), (10); 1, (10), (10); 1
$5 \cdot 10^{14}$	10^{14}	-	-	-	0, (10), (10); 1, (10), (10); 2

In Table 9.9 we show results for initial precisions (u_f, u, u_r) = (half, double, quad). Here the increase in the number of GMRES iterations per GMRES-based refinement step as $\kappa(A)$ increases is also dramatic. One interesting thing to notice here and even moreso in the previous examples with mode 3 matrices is that, in cases where both SGMRES-IR and GMRES-IR converge, the extra precision used in GMRES-IR does not help improve the convergence rate of GMRES (nearly the same number of GMRES iterations per refinement step are required). The aspect that improves is the number of refinement steps. This is likely related to the GMRES convergence trajectories for this class of problems; the residual will nearly stagnate (or at least not make much progress) for a number of iterations and then convergence will happen suddenly. The difference is likely that when this convergence does happen, the extra precision in GMRES-IR results in the approximate solution being found to greater accuracy than in the uniform precision iterations in SGMRES-IR.

As a result of this slow convergence of GMRES, MSIR does a precision switch from (u_f, u, u_r) = (half, double, quad) to (u_f, u, u_r) = (single, double, quad) for matrices with condition number $\kappa_2(A) \geq 10^4$. For matrices with condition number $\kappa_2(A) \geq 10^9$, MSIR does a precision switch a second time, from (u_f, u, u_r)

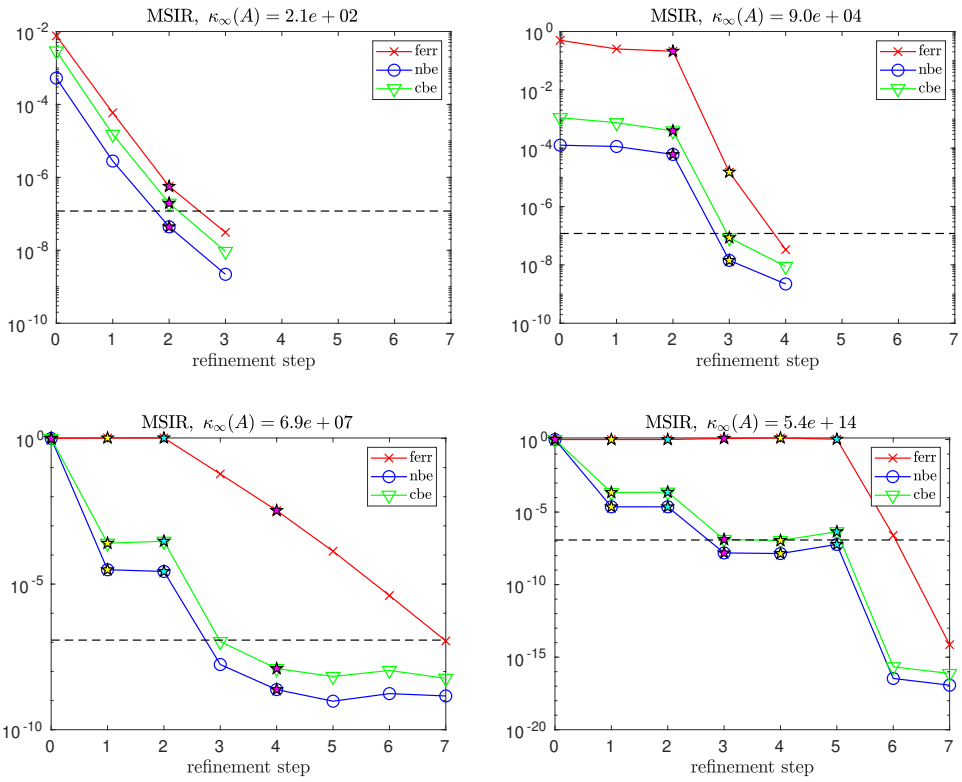

Figure 9.8: Convergence of errors in MSIR for random dense matrices having geometrically distributed singular values (mode 3) with $\kappa_2(A) = 10^1$ (top left), $\kappa_2(A) = 10^4$ (top right), $\kappa_2(A) = 10^7$ (bottom left), and $\kappa_2(A) = 10^{14}$ (bottom right) for initial precisions (u_f, u, u_r) = (half, single, double); see also Table 9.8.

= (single, double, quad) to (u_f, u, u_r) = (double, double, quad). Even though this means that MSIR computes 3 LU factorizations, one in half, one in single, and one in double, this is still likely to be much faster than SGMRES-IR, which does a total of 1841 $O(n^2)$ triangular solves in double precision, or GMRES-IR, which does a total of 927 $O(n^2)$ triangular solves in quadruple precision. We show MSIR convergence for $\kappa_2(A) \in \{10^1, 10^4, 10^5, 10^{11}\}$ in Figure 9.9.

Table 9.9: Number of SIR, SGMRES-IR, GMRES-IR, and MSIR steps with the number of GMRES iterations for each SGMRES-IR and GMRES-IR step for random dense matrices having geometrically distributed singular values (mode 3) with various condition numbers $\kappa_\infty(A)$, $\kappa_2(A)$, using initial precisions (u_f, u, u_r) = (half, double, quad). For $\kappa_2(A) = 10^{14}$, SGMRES-IR used (100,100,100,100,100,85,96,93,90,84,94,92,90,81,95,84,89,82,90,96) iterations.

$\kappa_\infty(A)$	$\kappa_2(A)$	SIR	SGMRES-IR	GMRES-IR	MSIR
$2 \cdot 10^2$	10^1	7	(5,5)	(5,5)	2, (5)
10^3	10^2	9	(6,6)	(6,6)	2, (6,6)
$9 \cdot 10^4$	10^4	-	(19,20)	(19,20)	2, (10), (10); 2
$8 \cdot 10^5$	10^5	-	(43,46)	(43,46)	0, (10), (10); 3, (3)
$7 \cdot 10^7$	10^7	-	(83,85)	(83,85)	0, (10), (10); 2, (6,7)
$6 \cdot 10^9$	10^9	-	(100,100)	(100,100)	0, (10), (10); 2, (10), (10); 2
$6 \cdot 10^{11}$	10^{11}	-	(100,100,100)	(100,100)	0, (10), (10); 3, (10), (10); 2, (2)
$5 \cdot 10^{14}$	10^{14}	-	(100,...)	(100,100,100,91,84,88,83,96,85,100)	0, (10), (10); 2, (10), (10); 2, (3,3,3)

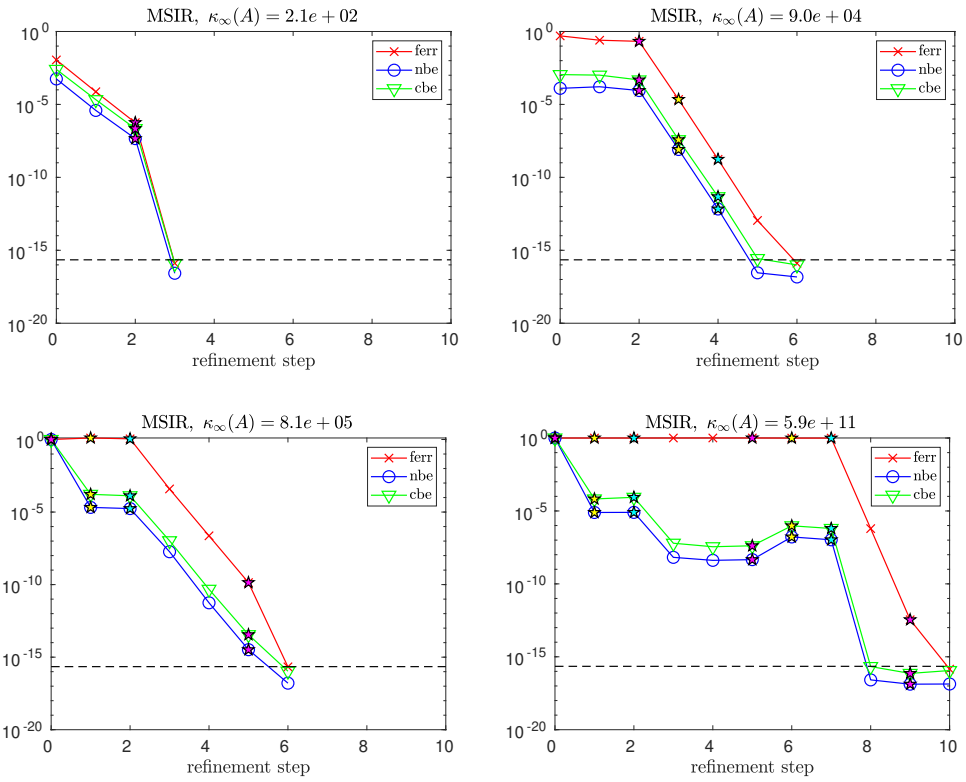

Figure 9.9: Convergence of errors in MSIR for random dense matrices having geometrically distributed singular values (mode 3) with $\kappa_2(A) = 10^1$ (top left), $\kappa_2(A) = 10^4$ (top right), $\kappa_2(A) = 10^5$ (bottom left), and $\kappa_2(A) = 10^{11}$ (bottom right) for initial precisions (u_f, u, u_r) = (half, double, quad).

9.2.2 SuiteSparse Matrices

We now test a subset of matrices taken from the SuiteSparse Collection (Davis and Hu [2011]) whose properties are listed in Table 9.10. As before, we test SIR, SGMRES-IR, GMRES-IR, and MSIR with initial precisions (u_f, u, u_r) = (single, double, quad), (u_f, u, u_r) = (half, single, double), and (u_f, u, u_r) = (half, double, quad). For these problems, we always set the right-hand side b to be the vector of ones. Tables 9.11-9.13 show the convergence behavior of the different algorithmic variants. For each precision combination, we show a few interesting convergence trajectories in Figures 9.10-9.12.

In Table 9.11 for initial precisions (u_f, u, u_r) = (single, double, quad), it is observed that SIR converges for all matrices except ww_36_pmec_36 (the most ill-conditioned one). Since SIR converges in few steps, MSIR does not switch to SGMRES-IR. For ww_36_pmec_36, however, MSIR detects nonconvergence of SIR and switches to SGMRES-IR, which then converges in 3 steps.

In Table 9.12 for initial precisions (u_f, u, u_r) = (half, single, double), it is seen that SIR converges for the matrices with $\kappa_\infty(A) < 10^6$. It is also observed that SGMRES-IR fails to converge for the extremely ill-conditioned matrix ww_36_pmec_36. MSIR switches to GMRES-IR for the two most ill-conditioned matrices, although for the steam3 case, the second switch is not technically necessary. It is not clear why for the steam3 matrix, MSIR performs two SGMRES-IR

Table 9.10: Matrices from Davis and Hu [2011] used for numerical experiments and their properties.

Name	Size	Nnz	κ_∞	Group	Kind
cage6	93	785	2.34E+01	vanHeukelum	Directed Weighted Graph
tols90	90	1746	3.14E+04	Bai	Computational Fluid Dynamics Problem
bfwa62	62	450	1.54E+03	Bai	Electromagnetics Problem
cage5	37	233	2.91E+01	vanHeukelum	Directed Weighted Graph
d_dyn	87	230	8.71E+06	Grund	Chemical Process Simulation Problem
d_ss	53	144	6.02E+08	Grund	Chemical Process Simulation Problem
Hamrle1	32	98	5.51E+05	Hamrle	Circuit Simulation Problem
ww_36_pmec_36	66	1194	4.283E+11	Rommes	Eigenvalue/Model Reduction Problem
steam3	80	314	7.64E+10	HB	Computational Fluid Dynamics Problem

Table 9.11: Number of SIR, SGMRES-IR, GMRES-IR, and MSIR steps with the number of GMRES iterations for each SGMRES-IR and GMRES-IR step for real matrices with initial precisions (u_f, u, u_r) = (single, double, quad).

Matrix	SIR	SGMRES-IR	GMRES-IR	MSIR
cage6	2	(2)	(2)	2
tols90	2	(2)	(2)	2
bfwa62	2	(2)	(2)	2
cage5	2	(2)	(2)	2
Hamrle1	2	(2)	(2)	2
d_ss	2	(2)	(2)	2
d_dyn	2	(2)	(2)	2
ww_36_pmec_36	-	(2,3,3)	(2,3,3)	2, (2,3,3)
steam3	2	(2)	(2)	2

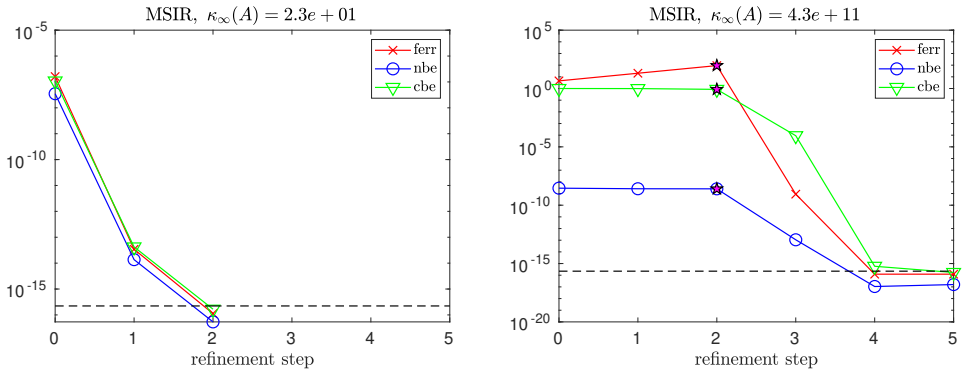

Figure 9.10: Convergence of errors in MSIR for *cage6* (left) and *ww_36_pmec_36* (right) using initial precisions (u_f, u, u_r) = (single, double, quad).

steps and one GMRES-IR step when SGMRES-IR alone only requires one step. For ww_36_pmec_36, MSIR also switches from (u_f, u, u_r) = (half, single, double) to (u_f, u, u_r) = (single, single, double) since GMRES-IR required too many GMRES iterations. It is lastly seen that for steam3, scaling is performed due to the resulting L and U factors containing Inf or NaN when the factorization is performed in half precision, as explained in Section 9.1.1.

In Table 9.13, using initial precisions (u_f, u, u_r) = (half, double, quad), we

Table 9.12: Number of SIR, SGMRES-IR, GMRES-IR, and MSIR steps with the number of GMRES iterations for each SGMRES-IR and GMRES-IR step for real matrices with initial precisions (u_f, u, u_r) = (half, single, double). Cases where scaling was required prior to LU factorization are marked with a *.

Matrix	SIR	SGMRES-IR	GMRES-IR	MSIR
cage6	2	(3)	(3)	2
tols90	2	(2)	(2)	2
bfwa62	4	(3)	(3)	2, (3)
cage5	2	(3)	(3)	2
Hamrle1	2	(2)	(2)	2
d_ss	-	(3,3)	(3,3)	0, (3,3)
d_dyn	-	(2,2)	(2,2)	0, (2,2)
ww_36_pmec_36	-	-	(66,66,7)	0, (7), (7); 2, (2,2,3,3), (2)
steam3*	-	(2)	(2)	2, (2,2), (2)

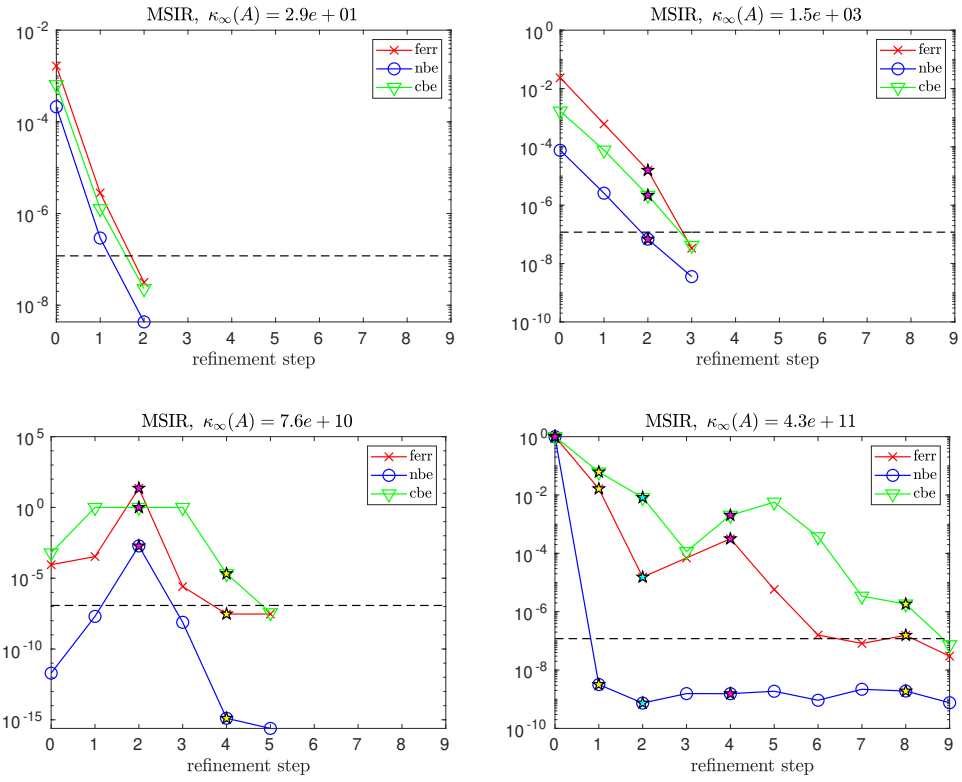

Figure 9.11: Convergence of errors in MSIR for *cage5* (top left), *bwfa62* (top right), *steam3* (bottom left), and *ww_36_pmec_36* (bottom right) using initial precisions (u_f, u, u_r) = (half, single, double).

see that MSIR is able to converge for all test problems and switches at least to SGMRES-IR in every case due to slow convergence of SIR. For the matrix ww_36_pmec_36, MSIR makes the second switch to GMRES-IR and then does a precision switch from (u_f, u, u_r) = (half, double, quad) to (u_f, u, u_r) = (single, double, quad) after one step due to the constraint on the number of GMRES iterations per step. As in the (u_f, u, u_r) = (half, single, double) case, due to the use of half precision in the factorization process, scaling is required for steam3.

Overall, we note that for these real-world sparse problems, to an even greater extent than in the dense test cases, SGMRES-IR and GMRES-IR often still perform well for matrices with condition numbers greatly exceeding constraints given in Table 9.3. We again stress that this motivates our multistage approach, since the theoretical analysis cannot necessarily give indication of which algorithm variant will be the best choice.

Table 9.13: Number of SIR, SGMRES-IR, GMRES-IR, and MSIR steps with the number of GMRES iterations for each SGMRES-IR and GMRES-IR step for real matrices with initial precisions (u_f, u, u_r) = (half, double, quad). Cases where scaling was required prior to LU factorization are marked with a *.

Matrix	SIR	SGMRES-IR	GMRES-IR	MSIR
cage6	5	(4,4)	(4,4)	2, (4)
tols90	5	(3,3)	(3,3)	2, (3)
bfwa62	9	(4,5)	(4,5)	3, (4)
cage5	5	(4,4)	(4,4)	2, (3)
Hamrle1	5	(3,3)	(3,3)	2, (3)
d_ss	-	(4,5)	(4,5)	0, (4,5)
d_dyn	-	(4,4)	(4,4)	0, (4,4)
ww_36_pmec_36	-	(8,8)	(8,9)	0, (7), (7); 2, (3)
steam3*	-	(3,3)	(3,3)	2, (3,3,3)

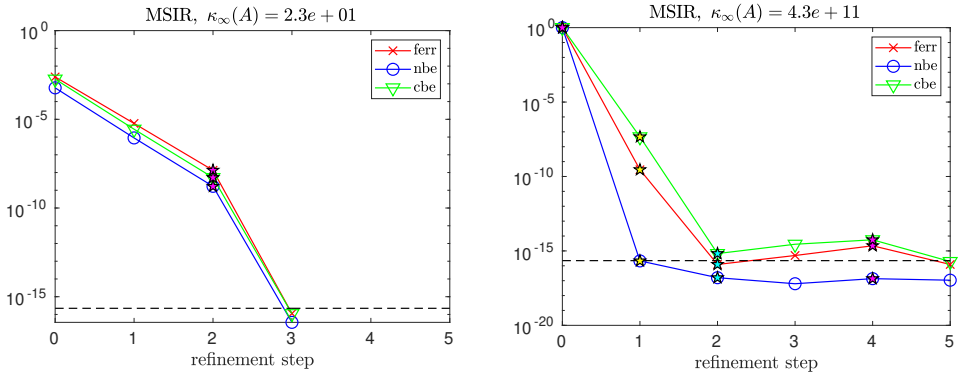

Figure 9.12: Convergence of errors in MSIR for *cage6* (left) and *ww_36_pmec_36* (right) using initial precisions (u_f, u, u_r) = (half, double, quad).

9.3 Conclusions and future work

Mixed precision iterative refinement, and in particular, GMRES-based iterative refinement, has seen great use recently due to the emergence of mixed precision hardware. The freedom to choose different precision combinations as well as a particular solver leads to an explosion of potential iterative refinement variants, each of which has different performance costs and different condition number constraints under which convergence is guaranteed. In practice, particular iterative refinement variants often work well for problems beyond what is indicated by the

analysis, making it difficult for a user to select a priori the least expensive variant that will converge. Motivated by improving usability and reliability, we have developed a multistage mixed precision iterative refinement approach, called MSIR. For a given combination of precisions, MSIR switches between three different iterative refinement variants distinguished by the way in which they solve for the correction to the approximate solution in order of increasing cost, according to stopping criteria adapted from Demmel et al. [2006]. Then, only if necessary, it increases the precision(s), refactorizes the matrix in a higher precision, and begins again. We discuss details of the algorithm and perform extensive numerical experiments on both random dense matrices and matrices from SuiteSparse (Davis and Hu [2011]) using a variety of initial precision combinations. Our experiments confirm that the algorithmic variants often outperform what is dictated by the theoretical condition number constraints and demonstrate the benefit of the multistage approach; in particular, our experiments show that there can be an advantage to first trying other solvers before resorting to increasing the precision and refactorizing.

There are a number of potential extensions to the work presented here. First, while we have only tested IEEE precisions, it is also worthwhile to perform experiments using non-IEEE floating point formats such as bfloat16 (Intel Corporation [2018]). We also plan to extend the multistage approach to least squares problems, which can be accomplished by combining the error bounds derived in Demmel et al. [2009] and the three-precision iterative refinement schemes for least squares problems in Carson et al. [2020]. Finally, while we have roughly used the number of triangular solves in a given precision as a metric for discussing probable performance characteristics, this may not be a good indication of performance in practice. Thorough high-performance experiments on modern GPUs are needed to appropriately gauge the overhead of the MSIR approach and the algorithm's suitability in practice.

Bibliography

Advanpix LLC. Multiprecision computing toolbox for MATLAB. URL `http://www.advanpix.com/`.

Patrick Amestoy, Alfredo Buttari, Nicholas J. Higham, Jean-Yves L'Excellent, Theo Mary, and Bastien Vieublé. Five-precision GMRES-based iterative refinement. Technical Report 2021.5, April 2021. URL `http://eprints.maths.manchester.ac.uk/2807/`.

J Bowdler, Hilary, RS Martin, Gwendoline Peters, and JH Wilkinson. Solution of real and complex systems of linear equations. *Numerische Mathematik*, 8 (3):217–234, 1966.

Erin Carson and Nicholas J. Higham. A new analysis of iterative refinement and its application to accurate solution of ill-conditioned sparse linear systems. *SIAM Journal on Scientific Computing*, 39(6):A2834–A2856, 2017. doi: 10.1137/17M1122918.

Erin Carson and Nicholas J. Higham. Accelerating the solution of linear systems by iterative refinement in three precisions. *SIAM Journal on Scientific Computing*, 40(2):A817–A847, 2018. doi: 10.1137/17M1140819.

Erin Carson, Nicholas J. Higham, and Srikara Pranesh. Three-precision GMRES-based iterative refinement for least squares problems. *SIAM Journal on Scientific Computing*, 42(6):A4063–A4083, 2020. doi: 10.1137/20M1316822.

Timothy A. Davis and Yifan Hu. The University of Florida Sparse Matrix Collection. *ACM Transactions on Mathematical Software*, 38(1), 2011. doi: 10.1145/2049662.2049663.

James Demmel, Yozo Hida, William Kahan, Xiaoye S. Li, Sonil Mukherjee, and E. Jason Riedy. Error bounds from extra-precise iterative refinement. *ACM Trans. Math. Softw.*, 32(2):325–351, June 2006. ISSN 0098-3500. doi: 10.1145/1141885.1141894.

James Demmel, Yozo Hida, E Jason Riedy, and Xiaoye S Li. Extra-precise iterative refinement for overdetermined least squares problems. *ACM Transactions on Mathematical Software (TOMS)*, 35(4):1–32, 2009.

A. Haidar, S. Tomov, J. Dongarra, and N. J. Higham. Harnessing GPU tensor cores for fast fp16 arithmetic to speed up mixed-precision iterative refinement solvers. In *SC18: International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 603–613, 2018. doi: 10.1109/SC.2018.00050.

Nicholas J Higham and Srikara Pranesh. Simulating low precision floating-point arithmetic, mims eprint 2019.4. *Manchester Institute for Mathematical Sciences, The University of Manchester, UK*, 2019.

HPL-MxP. HPL-MxP mixed-precision benchmark. `https://icl.bitbucket.io/hpl-ai/`, November 2019.

Intel Corporation. Bfloat16 – hardware numerics definition. Technical Report 338302-001US, Revision 1.0, Intel, November 2018.

M Jankowski and H Woźniakowski. Iterative refinement implies numerical stability. *BIT Numerical Mathematics*, 17(3):303–311, 1977.

Shuhei Kudo, Keigo Nitadori, Takuya Ina, and Toshiyuki Imamura. Prompt report on exa-scale HPL-AI benchmark. In *2020 IEEE Int. Conf. Cluster Comput. (CLUSTER)*, pages 418–419. IEEE, 2020.

Julie Langou, Julien Langou, Piotr Luszczek, Jakub Kurzak, Alfredo Buttari, and Jack Dongarra. Exploiting the performance of 32 bit floating point arithmetic in obtaining 64 bit accuracy (revisiting iterative refinement for linear systems). In *SC'06: Proceedings of the 2006 ACM/IEEE conference on Supercomputing*, pages 50–50. IEEE, 2006.

Cleve B Moler. Iterative refinement in floating point. *Journal of the ACM (JACM)*, 14(2):316–321, 1967.

Christopher C. Paige, Miroslav Rozložník, and Zdeněk Strakoš. Modified Gram-Schmidt (MGS), least squares, and backward stability of MGS-GMRES. *SIAM Journal on Matrix Analysis and Applications*, 28(1):264–284, 2006. doi: 10.1137/050630416.

Stanimire Tomov, Jack Dongarra, and Marc Baboulin. Towards dense linear algebra for hybrid GPU accelerated manycore systems. *Parallel Computing*, 36 (5-6):232–240, June 2010. ISSN 0167-8191. doi: 10.1016/j.parco.2009.12.005.

TOP500. TOP500. Online, June 2021. URL `https://www.top500.org/`.

James Hardy Wilkinson. Progress report on the automatic computing engine. Technical Report MA/17/1024, Mathematics Division, Department of Scientific and Industrial Research, National Physical Laboratory, Teddington, UK, 1948.

James Hardy Wilkinson. *Rounding errors in algebraic processes*. Prentice-Hall, 1963.

Conclusion

High-performance computations in numerical linear algebra have become necessary for solving extremely large problems coming from computational and data science applications. Using mixed-precision is beneficial in such computations. Combining the performance advantages of using low precision and accuracy of high precision, the communication and computation cost of algorithms can be reduced significantly, resulting in low energy consumption and improved runtimes. However, there are still many open questions for different algorithms and their finite precision analysis. With this motivation, this thesis focuses on developing new mixed-precision algorithms and analyzing their performance and errors.

Orthogonalization processes are the core of Krylov subspace methods. The stability of these iterative approaches is directly related to the loss of orthogonality in the orthogonalization processes. However, preserving orthogonality is not easy. In general, the rate of the loss of orthogonality is related to the conditioning of the input matrix with various strict assumptions on the condition number. To relax these assumptions, techniques such as reorthogonalization can be used. On the other hand, having a stable method be expensive since orthogonalization processes can dominate the cost of Krylov subspace methods. One can thus reduce the cost of these processes using lower precisions in certain parts of the algorithm. We introduced and analyzed a mixed-precision variant of the reorthogonalized low-synchronization block CGS orthogonalization process to reduce the computational cost of the uniform precision procedure. While reorthogonalization maintains stability, low-synchronization reduces the communication cost, and mixed-precision increases performance by reducing the computation cost of the algorithm (Chapter 4).

Another technique to reduce loss of orthogonality is using the Pythagorean theorem. However, although such techniques can relax the constraints on the condition number, the loss of orthogonality might still not be low enough to guarantee the stability of the Krylov subspace method. For this reason, we introduce two new reorthogonalized BCGSI+P variants to improve stability using the Pythagorean theorem twice (Chapter 5). Using reorthogonalization and the Pythagorean theorem together, we obtain algorithms that better preserve orthogonality than the current state-of-the-art block orthogonalization processes. Using mixed-precision in a similar way as in the low-synchronization variant, we increased the range of applicability of the block Gram-Schmidt processes while preserving orthogonality.

To increase the range of applicability of stationary iterative solvers, we can use a hybrid approach which uses a preconditioned Krylov subspace method as the inner solver. We focus on using variants of Newton's approach, such as RQI, with a Krylov subspace method, such as the PCG algorithm. To solve large-scale total least squares problems, for instance, RQI can be used as the main iterative solver. To cheaply solve linear systems inside the RQI algorithm, PCG can be used under several assumptions. However, the resulting algorithm is still expensive due to the QR factorization. We therefore introduced a mixed-precision variant of this approach to reduce the computation cost while maintaining accuracy (Chapter 6).

Another variant of Newton's method is the iterative refinement algorithm. As RQI, IR algorithms require a linear solver in each outer iteration. To increase the range of problems that can be solved, a Krylov subspace method, such as preconditioned GMRES, can be used to solve the linear systems as in RQI-PCGTLS; this approach is called GMRES-IR. GMRES-IR can be much more expensive than SIR depending on the number of iterations performed. We introduced recycling in GMRES to reduce the GMRES iteration count and thus the cost of the overall algorithm (Chapter 7).

GMRES-IR can also be used to solve least squares problems by working with an augmented system; this approach is called GMRES-LSIR. Using the idea of GMRES-LSIR with different preconditioners, we showed that weighted least squares problems can also be solved using a GMRES-based method in mixed-precision (Chapter 8).

In some cases, standard IR can fail depending on the conditioning of the matrix and the precisions used. However, using GMRES-IR can be more expensive since one GMRES-IR iteration is more expensive than one SIR iteration. To benefit from both approaches and their variants, we proposed a multistage IR approach to reduce the computation cost while improving applicability. We start with SIR and then switch to GMRES-IR variants if slow convergence or divergence is detected. If necessary, we increase the precision and start from SIR once again to achieve the desired accuracy (Chapter 9).

Mixed-precision is still a popular tool in scientific computing. Combining it with other tools, such as randomization, can improve performance even more. There are still many open questions in using multiple precisions in numerical linear algebra computations. One of the most important concerns of mixed-precision algorithms is stability. The backward stability of mixed-precision randomized or communication-avoiding approaches may not be guaranteed. Another open field comes from the use of extremely low precision formats. For instance, with the emergence of fp8-supporting hardware, stability remains a major problem.

Mixed-precision is not only used in solving linear problems. There are new variants for nonlinear problems, such as inverse problems or matrix functions. With this motivation, mixed precision variants of more expensive solvers, such as rational Krylov subspace methods, can be investigated. However, as stated above, stability will be the main concern of all these new developments.