

# THÈSE

**En vue de l'obtention du**

# DOCTORAT DE L'UNIVERSITÉ DE TOULOUSE

**Délivré par :**

Institut National Polytechnique de Toulouse (Toulouse INP)

**Discipline ou spécialité :**

Mathématiques Appliquées

---

**Présentée et soutenue par :**

M. BASTIEN VIEUBLE

le mercredi 30 novembre 2022

**Titre :**

Raffinement itératif en précision mixte pour la résolution de systèmes linéaires creux de grande taille

---

**Ecole doctorale :**

Mathématiques, Informatique, Télécommunications de Toulouse (MITT)

**Unité de recherche :**

Institut de Recherche en Informatique de Toulouse ( IRIT)

**Directeur(s) de Thèse :**

M. ALFREDO BUTTARI

M. THÉO MARY

**Rapporteurs :**

M. JULIEN LANGOU, UNIVERSITE DE L'ETAT DU COLORADO

MME ERIN CARSON, UNIVERSITE CHARLES A PRAGUE

MME XIAOYE LI, LAWRENCE BERKELEY NATIONAL LABORATORY

**Membre(s) du jury :**

M. SERGE GRATTON, TOULOUSE INP, Président

M. ALFREDO BUTTARI, TOULOUSE INP, Membre

M. EMMANUEL AGULLO, INRIA, Membre

M. MARC BABOULIN, UNIVERSITE PARIS-SACLAY, Membre

M. NICHOLAS HIGHAM, UNIV. MANCHESTER, Invité(e)

M. THÉO MARY, UNIVERSITE SORBONNE, Membre

*À mes parents,*
*À mon frère,*
*À ma grand-mère*

# Abstract

The increasing availability of very low precisions (tfloat32, fp16, bfloat16, fp8) in hardware pushes modern high performance computing to embrace mixed precision standards. By employing mostly low precision and by making wise use of high precision, mixed precision algorithms can leverage the low precision advantages while preserving the quality of the computed solution. Mixed precision iterative refinement is one of the oldest and most famous representatives of these algorithms; this method was shown to be very effective in reducing the resource consumption of linear solvers while delivering accurate solutions in a robust way.

This thesis is dedicated to investigating the use of this algorithm for the solution of large sparse linear systems. We structure the document as follows.

Our first concern is to provide a comprehensive understanding of iterative refinement. This part of our work covers a survey listing the different research studies on this algorithm that explains its evolutions throughout time and a technical description of a selected set of state-of-the-art iterative refinement algorithms.

Then, we focus on improving sparse direct solvers with iterative refinement. We proceed in two steps. First, we relax restrictive requirements on the LU-GMRES-IR3 algorithm, which is a form of iterative refinement capable of handling inaccurate factorizations for ill-conditioned problems. It leads us to propose the LU-GMRES-IR5 algorithm that has five independent precision parameters and is more suited to the solution of large sparse systems. Second, we address the parallel implementation of state-of-the-art iterative refinement algorithms combined with state-of-the-art approximate sparse factorizations to solve real-life problems from academic and industrial applications. Our performance study demonstrates significant reductions in time and memory with respect to a standard sparse direct solver in double precision.

Finally, we use iterative refinement to improve sparse iterative solvers. We develop an analysis for a new mixed precision preconditioned GMRES (M-GMRES-IR6) that aims at covering previous existing implementations that are not yet covered by an analysis and at proposing a new mixed precision strategy based on the application of the preconditioner in a lower precision than the application of the original matrix $A$. Our numerical results pave the way towards promising resource savings for parallel implementations of GMRES.

# Résumé

L'accessibilité grandissante des arithmétiques à précision faible (tfloat32, fp16, bfloat16, fp8) dans les calculateurs encourage le calcul à hautes performances à se tourner vers la précision mixte. En employant principalement des précisions faibles tout en faisant un usage intelligent de précisions hautes, les algorithmes de précision mixte sont capables d'exploiter les bénéfices des précisions faibles tout en préservant la qualité de la solution calculée. Le raffinement itératif est un des plus vieux et des plus célèbres représentants de cette classe d'algorithme; il est capable de réduire efficacement la consommation de ressource des solveurs linéaires tout en conservant la robustesse et la qualité de la solution.

Cette thèse est dédiée à étudier l'utilisation de cette algorithme pour la résolution de systèmes linéaires creux. Elle est architecturée de la façon suivante.

Dans un premier temps nous nous intéressons à produire un récapitulatif complet sur les algorithmes de raffinement itératif. Cela couvre une liste des différentes études de recherche sur le sujet au travers du temps, et une description technique de certains de ces algorithmes les plus à la pointe.

Dans un second temps nous nous intéressons à l'amélioration des solveurs directs creux à l'aide du raffinement itératif. Nous procédons en deux étapes. D'abord nous relaxons des conditions restrictives de l'algorithme LU-GMRES-IR3 qui est une forme de raffinement itératif capable de traiter des factorisations peu précises pour des problèmes mal conditionnés. Pour ce faire, nous proposons l'algorithme LU-GMRES-IR5 utilisant cinq précisions indépendantes. Ensuite, nous abordons l'implémentation parallèle de raffinement itératifs combinés avec des factorisations creuses approximées pour la résolution de problèmes industrielles. Notre étude de performance démontre que ces approches nous permettent d'obtenir des réductions importantes de la consommation en temps et en mémoire.

Finalement, nous nous intéressons à l'amélioration des solveurs itératifs creux. Nous développons une analyse pour un nouvel algorithme de GMRES préconditionné en précision mixte (M-GMRES-IR6), cette analyse aspire en particulier à couvrir des implémentations existantes de GMRES et à proposer une nouvelle stratégie de précision mixte consistant à appliquer le préconditionneur dans une précision plus basse que le produit matrice–vecteur. Nos résultats numériques ouvrent la voie à des implémentations parallèles de GMRES plus économes en temps et en mémoire.

# Remerciements

Le temps des remerciements est toujours un moment un peu spécial, c'est une des choses que vous lisez en premier mais que j'écris en dernier. C'est donc non sans émotions que je viens clôturer trois années de travail avec ces quelques lignes, en remerciant, finalement, tous les gens brillants qui m'ont entouré et sans qui, par la force des choses, ce document n'aurait jamais existé.

Les premières personnes directement responsables de la qualité de ce manuscrit sont mes deux superviseurs Alfredo et Théo. Quand on a été aussi bien supervisé que moi, par des gens scientifiquement et humainement au top, c'est difficile de ne pas produire quelque chose de satisfaisant. Au-delà même de la rigueur scientifique, de la méthodologie et des connaissances qu'ils m'ont transmise, ce sont des conseils, des opportunités, une visibilité et une communauté qu'ils m'ont offert. En ce sens, ils n'ont pas "juste" été irréprochables, mais ils m'ont aussi préparé une rampe de lancement pour l'avenir. Je tacherai d'être à la hauteur de ce cadeau et de suivre leur modèle pour peut-être à mon tour, un jour, rendre l'appareil. Pour toutes ces choses, pour le temps et l'énergie qu'ils m'ont consacrés, pour leurs modesties, leurs gentillesses et leurs aides, pour leurs ambitions et leurs dévotions à la discipline, je tiens à leur dire qu'ils ont toute ma reconnaissance et mon respect, que cela a été un honneur de travailler avec de grands chercheurs comme eux, merci !

Durant ma thèse, j'ai eu la chance de travailler avec Patrick, Jean-Yves et Chiara (MUMPS Tech). Je tiens à les remercier, premièrement, pour notre collaboration qui m'a permis d'apporter une saveur concrète et pratique à ce travail, mais aussi pour leurs conseils, leurs aident et pour certains mêmes, d'avoir été les supers professeurs qui m'ont introduit à ce domaine et motivé à continuer en thèse.

La dernière personne ayant apporté le plus de soutien scientifique et matériel à cette thèse est Nick. Il m'a aidé, conseillé et accueilli chaleureusement dans son équipe à l'University of Manchester. Ses standards d'excellence et son expertise pointue ont grandement participé à la qualité de ce travail, il a toute ma gratitude. Je remercie en passant toute son équipe, le Numerical Linear Algebra group, avec qui j'ai eu des supers pauses café et discussions !

Je remercie chaleureusement Julien Langou, Sherry Li et Erin Carson pour avoir accepté de donner leurs temps pour évaluer ce manuscrit et écrire un rapport, en plus d'avoir fait

le (long) voyage jusqu'à Toulouse pour assister à la défense. Je remercie évidemment le reste du jury Emmanuel Agullo, Marc Baboulin et Serge Gratton pour leurs retours sur mon travail et les discussions stimulantes qu'y en ont suivie.

Je remercie l'ensemble de la communauté scientifique qui a construit pièce par pièce les différents éléments qui ont rendu nos contributions possibles. En particulier, sans le travail préalable d'Erin et Nick sur lequel les contributions de ce manuscrit sont majoritairement basées, cette thèse n'aurait pas été possible.

Je remercie aussi les partenaires industrielles de MUMPS Tech et le projet EoCoE pour m'avoir fourni quelques-uns des problèmes qui sont venus nourrir ce manuscrit. J'aimerais aussi remercier l'équipe du CALMIP qui nous a offert l'accès au supercalculateur OLYMPE et sur lequel toutes nos expériences de calcul parallèle ont été tournées.

Probablement que l'idée même de faire une thèse n'aurait pas germée dans mon esprit sans les opportunités de stages en recherche extraordinaires que mes professeurs à l'ENSEEIHT m'ont déniché. Je pense notamment à Axel Carlier et Vincent Charvillat qui m'ont permis de faire une année de césure à l'IPAL (Singapour) encadrée par Christophe Jouffrais. Ou encore Serge Gratton qui m'a envoyé à l'AMSS (Pékin) encadré par Pr. Xin Liu pour mon projet de fin d'étude. J'en profite pour remercier sincèrement mes encadrants Christophe et Xin, qui m'ont accordé leur confiance et m'ont introduit au monde de la recherche.

Avant et durant toute la durée de ma thèse, j'ai eu la chance d'être entouré de gens bienveillants et stimulants qui ont indirectement contribué à l'aboutissement de ce travail.

Je ne peux pas passer à côté d'une petite dédicace à Justin et Matthieu, qui ont constamment essayé de me décourager de faire un doctorat, ce qui, par plaisir de contradiction, a eu l'effet totalement opposé. Je me plais donc à penser que c'est grâce à eux que je me suis lancé dans cette aventure. C'est notamment avec eux que j'ai appris les rudiments du bon thésard, et ils resteront à jamais mes guides spirituelles.

Je pense aussi à mes collègues Antoine J., Sophie, Théo, Sadok, Rémy, Valentin et Olivier, avec qui l'ambiance au labo était au top et qui seront tous j'en suis sûr, d'ici une toute petite poignée d'années, des docteurs (ou docteur avec l'HDR pour le concerné). Dans le tas, il y a mes camarades de bureau Antoine B. et Jean-Paul, avec qui ont formé le trio de vieux briscards insupportables de la F321. Les compagnons de bureau c'est un peu comme la famille : on ne les choisit pas. Cependant, avec tout ce qu'on a traversé ensemble, je pense que maintenant on peut dire qu'on est une famille !

Je ne peux évidemment pas faire l'impasse sur mes vieux compagnons Johary, Auriann et Aurélien. J'ai l'impression que nos projets respectifs ont grandi ensemble depuis presque une décennie maintenant. Nos vies et nos décisions ont été tellement entrelacées qu'il m'est impossible d'évaluer à quel point ils ont influencé ce travail. Néanmoins, j'ai la certitude que je leur dois beaucoup et que leurs talents et leurs passions les mèneront sur la voie des spiritual warriors.

Il y a de ces personnes dont vous avez du mal à quantifier à quel point elles vous ont aidés et soutenus quand leur bienveillance naturelle devient une sorte de routine à laquelle on s'habitue. Peng (alias 海盗猫 ) a été une de ces personnes pour moi. En trois ans, elle a dû totaliser plus de quelques milliers d'heures à m'écouter me plaindre au sujet de la thèse.

Elle en sait d'ailleurs probablement plus que n'importe qui sur les backstages et détails croustillants entourant ce manuscrit. Le soutien qu'elle m'a donné durant cette thèse a une valeur inestimable à mes yeux et j'espère avoir réussi à lui en rendre un peu.

Naturellement, je ne peux pas finir ces remerciements sans mentionner les personnes les plus importantes : mes parents, mon frère et ma grand-mère. C'est grâce à eux que j'en suis ici et je leur dois tout. Comme ça a dû être pas mal de boulot de m'élever et de me supporter, ce manuscrit est en quelque sorte un peu le résultat de leur travail aussi, c'est pour ça que je leurs dédie.

Malheureusement, je n'ai pas la place pour citer toutes les personnes proches : famille, colocs, vieux amis ou amis récents, etc. Néanmoins, sachez que je ne vous oublie pas. Merci d'avoir été là ! Vous avez contribué aussi à ce travail.

Ironiquement, le chapitre des remerciements est aussi le meilleur endroit pour dire qui je ne remercie pas. En particulier, je ne remercie pas la COVID-19 qui a démarré quatre mois après le commencement de ma thèse et nous a tous bien ennuyé !

Finalement, je vous remercie vous, lecteur, qui que vous soyez, pour vous intéresser à ce travail, et dans le même temps, à faire vivre ces résultats scientifiques et donner du sens à ce manuscrit.

Il est donc temps pour moi de conclure et pour vous de commencer la lecture.

# Contents

**Abstract** i

**Résumé** iii

**Remerciements** v

**Acronyms** xiii

**List of Figures** xv

**List of Tables** xvii

**List of Algorithms** xix

**1 Introduction** 1

**2 Background** 5

2.1 Floating-point arithmetic . . . 5

2.1.1 Basics . . . 5

2.1.2 Commonly available floating-point arithmetics in computers . . . 8

2.1.3 Low precision (floating-point) arithmetics . . . 9

2.1.4 Rounding error analysis notations . . . 13

2.2 Direct solvers . . . 15

2.2.1 LU solver . . . 16

2.2.2 Least squares problem and QR solver . . . 20

2.2.3 The multifrontal sparse direct solver . . . 21

2.2.4 Numerical approximations in sparse factorization . . . 26

2.3 Iterative solvers . . . 30

2.3.1 GMRES . . . 30

2.3.2 Preconditioners . . . 34

2.3.3 Other iterative solvers . . . 37

**3 Iterative refinement history** **39**
3.1 Newton's method (17th century) . . . 39
3.2 From the 40s to the 70s . . . 40
3.3 From the 70s to the 2000s . . . 43
3.4 From the 2000s to the 2010s . . . 47
3.5 From the 2010s to 2022 . . . 50
3.6 Summary . . . 54

**4 State-of-the-art iterative refinement** **59**
4.1 On the understanding of refining a linear system . . . 59
4.2 Generalized iterative refinement . . . 61
4.2.1 Preliminaries . . . 61
4.2.2 Forward and backward errors analyses . . . 63
4.2.3 Various practical comments . . . 65
4.2.4 Targeting low precisions . . . 66
4.3 LU-IR3 . . . 66
4.3.1 Error analysis . . . 67
4.3.2 Targeting low precisions . . . 67
4.4 LU-GMRES-IR3 . . . 67
4.4.1 Error analysis . . . 68
4.4.2 LU-GMRES-IR3 vs LU-IR3 . . . 69
4.4.3 GMRES-IR . . . 70
4.5 Extension to least squares problem . . . 70
4.5.1 Iterative refinement on the normal equations . . . 70
4.5.2 Iterative refinement on the overdetermined system . . . 71
4.5.3 Iterative refinement on the augmented system . . . 72
4.6 Stopping criteria . . . 75
4.7 Scaling . . . 76
4.8 Summary . . . 77

**5 LU-GMRES-IR in five precisions** **79**
5.1 From LU-GMRES-IR3 to LU-GMRES-IR5 . . . 79
5.2 Rounding error analysis . . . 81
5.2.1 Error analysis of MGS-GMRES with arbitrary matrix–vector products 82
5.2.2 Error analysis of LU-GMRES-IR5 with general $\mathbf{u_g}$ and $\mathbf{u_p}$ precisions . 85
5.2.3 Convergence conditions on $\kappa(\mathbf{A})$ . . . 87
5.2.4 Comments on the results of the analysis . . . 88
5.3 Identifying meaningful combinations of precisions . . . 89
5.4 GMRES stopping criterion . . . 92
5.4.1 On the cost of refinement iterations . . . 92
5.4.2 On the convergence behavior . . . 93
5.4.3 Rounding error analysis with stopping criterion . . . 96
5.5 Numerical experiments . . . 97
5.5.1 Random dense matrices . . . 97

5.5.2 Real-life matrices from SuiteSparse . . . 100
5.6 Practical advice . . . 103
5.7 LU-GMRES-IR5 for least squares problem . . . 105
5.8 Conclusion . . . 110

**6 Iterative refinement for sparse approximate factorizations 111**
6.1 Reducing the computational cost of sparse direct solvers . . . 111
6.2 Specific features . . . 113
6.3 Error analysis with approximate factorization . . . 114
6.3.1 Approximate factorization model . . . 115
6.3.2 Error analysis for LU-IR3 . . . 116
6.3.3 Error analysis for LU-GMRES-IR5 . . . 117
6.3.4 Summary of the error analysis and interpretation . . . 118
6.3.5 Convergence conditions for BLR and static pivoting . . . 119
6.4 Performance analysis . . . 119
6.4.1 Implementation details . . . 120
6.4.2 Experimental setting . . . 121
6.4.3 Cast of the factors . . . 124
6.4.4 Performance with standard factorization . . . 127
6.4.5 Performance with approximate factorizations . . . 134
6.4.6 Performance summary . . . 143
6.4.7 Scalability and parallelism . . . 144
6.5 Conclusions . . . 147

**7 Iterative refinement with preconditioned GMRES 149**
7.1 State-of-the-art mixed precision strategies for GMRES . . . 149
7.2 Left-preconditioned MGS-GMRES in mixed precision . . . 152
7.2.1 Backward stability of left MGS-GMRES in mixed precision . . . 154
7.2.2 Differentiating the precisions $\mathbf{u_a}$ and $\mathbf{u_m}$ . . . 155
7.2.3 Numerical experiments . . . 158
7.3 M-GMRES-IR6 . . . 162
7.3.1 Equivalence between restarted GMRES and iterative refinement . . . 163
7.3.2 Error analysis and convergence conditions . . . 164
7.3.3 Numerical experiments . . . 168
7.4 Conclusion . . . 173

**8 Conclusion 175**
8.1 Summary . . . 175
8.2 Future work . . . 176

**Scientific communications 181**

**Bibliography 183**

**Appendix 205**

# Acronyms

**AQR-GMRES-IR3** QR preconditioned GMRES-based iterative refinement on the augmented system in three precisions 71, 73, 74, 103

**AQR-GMRES-IR5** QR preconditioned GMRES-based iterative refinement on the augmented system in five precisions 71, 103, 104

**AQR-IR3** QR-based iterative refinement on the augmented system in three precisions 69–71, 73, 74

**BiCG** BiConjugate Gradient 37, 174

**BLR** block low-rank xvii, xix, 26–29, 109–113, 117–119, 122, 123, 132, 133, 136, 138, 140–142, 146

**CG** Conjugate Gradient 37, 48, 51, 148, 149, 174

**CGS** Classical Gram-Schmidt xix, 32, 33, 150

**CGS2** Classical Gram-Schmidt with reorthogonalization 32

**CPU** Central Processing Unit 9, 11, 48, 175

**FGMRES** Flexible Generalized Minimal RESidual 35, 48, 148

**FOM** Full Orthogonalization Method 37

**FPGA** Field-Programmable Gate Array 48, 52

**GE** Gaussian Elimination 10, 16, 18, 19, 21, 22, 41, 64

**GEPP** Gaussian Elimination with Partial Pivoting 19, 35, 41, 43, 44, 51, 53, 64, 66, 79, 94, 112, 175

**GMRES** Generalized Minimal RESidual i, 3, 4, 30–35, 37, 46, 48, 51, 52, 55, 65–67, 72, 77, 78, 80, 83, 87, 89–91, 93–95, 97, 98, 101–103, 108, 115, 118, 119, 124, 129, 131, 132, 136, 142, 143, 147–154, 156–159, 161, 162, 166, 169, 171, 173–176

**GMRES-IR** GMRES-based iterative refinement 67, 78, 161

**GPU** Graphics Processing Unit 1, 3, 8, 9, 11, 12, 48, 51, 52, 110, 175

**ILU** Incomplete LU xv, 36, 67, 117, 149, 155, 161

**ILUT** Incomplete LU with threshold 36, 148, 169, 171

**LS** least squares 20, 42, 51, 55, 68, 70, 71, 103, 108, 181

**LU-GMRES-IR** LU preconditioned GMRES-based iterative refinement 78, 86, 101, 107

**LU-GMRES-IR3** LU preconditioned GMRES-based iterative refinement in three precisions i, xvii, 3, 55, 65–68, 71–74, 77–80, 86, 87, 99, 103, 107, 108, 110, 112, 148, 173, 174

**LU-GMRES-IR5** LU preconditioned GMRES-based iterative refinement in five precisions i, xv, xvii, 3, 4, 66, 77–80, 87–92, 94–103, 107–115, 117–119, 122–133, 136, 138, 139, 142–144, 146, 149, 153, 161, 162, 164, 165, 171, 174–176

**LU-IR** LU-based iterative refinement xix, 64, 78, 111

**LU-IR3** LU-based iterative refinement in three precisions xv, xvii, 55, 64, 65, 67, 68, 72–74, 78, 87–89, 91, 95–103, 108–115, 118, 119, 124–133, 136, 138, 141–144, 146, 174–176

**M-GMRES-IR6** GMRES-based iterative refinement with arbitrary preconditioner $M$ in six precisions i, xv, 4, 147, 150, 160–172, 174, 176, 177

**MGS** Modified Gram-Schmidt xix, 31–33, 150, 182

**MGS-GMRES** Modified Gram-Schmidt Generalized Minimal RESidual 30, 32, 34, 66, 71, 79–83, 90, 94, 104, 108, 116, 118, 147, 150–153, 156, 157, 160–162, 171, 174, 181–183

**MINRES** MINimal RESidual 37, 174, 175

**MUMPS** MUltifrontal Massively Parallel sparse direct Solver 26, 45, 52, 109, 118–120, 122–125, 132, 136, 138, 140, 142–144, 146, 176

**QR-IR3** QR-based iterative refinement in three precisions 68

**restarted GMRES** restarted Generalized Minimal RESidual 33, 52, 67, 95, 161, 165

**TPU** Tensor Processing Unit 1, 9, 11

# List of Figures


2.1 Bits distribution on low precision arithmetics. . . . . 10
2.2 Dense tile LU (left-looking) factorization. . . . . 18
2.3 A sparse matrix and its elimination tree. . . . . 21
2.4 Fill-in. . . . . 22
2.5 Right-looking, left-looking, and multifrontal approaches. . . . . 24
2.6 Assembly tree. . . . . 24
2.7 Compression by block of a dense matrix. . . . . 27
2.8 Entries in $A$ and in the ILU and full LU factors. . . . . 36
2.9 A block diagonal decomposition of a matrix. . . . . 37

4.1 Newton's method in the presence of rounding errors. . . . . 60

5.1 Experimental convergence conditions of LU-GMRES-IR5. . . . . 98
5.2 Experimental convergence conditions of five precisions variants. . . . . 99
5.3 Condition numbers of SuiteSparse matrices after scaling. . . . . 100
5.4 Performance profile of different variants of LU-IR3 and LU-GMRES-IR5. . . . 102
5.5 Forward error achieved by different variants of LU-GMRES-IR5 on SuiteSparse matrices. . . . . 104

6.1 Recursive cast in-place. . . . . 125
6.2 Memory consumption of LU-GMRES-IR5. . . . . 126
6.3 Forward error achieved by three IR variants on our set of matrices. . . . . 129
6.4 Execution time on a subset of large sparse matrices. . . . . 130
6.5 Memory consumption on a subset of large sparse matrices. . . . . 132
6.6 Execution time for different numbers of threads. . . . . 145
6.7 Execution time and memory consumption for different numbers of MPI. . . 146
6.8 Ratio between the DMUMPS solve and factorization operations. . . . . 147

7.1 Evolution of $\rho_A$ and $\rho_M$ with varying condition numbers on $M$. . . . . 160
7.2 Evolution of $\rho_A$ and $\rho_M$ with varying condition numbers on $A$. . . . . 161
7.3 Number of iterations according to $\kappa(A)$ and $\kappa(M)$ on random matrices. . . . . 169

7.4 Supplement for Figure 7.3 . . . . . 171
7.5 Forward error for different variants of M-GMRES-IR6 on SuiteSparse matrices. 172

# List of Tables


2.1 List of commonly available floating-point arithmetics. . . . . 8
2.2 Theoretical complexities of the multifrontal sparse solver. . . . . 25
2.3 Theoretical complexities of the BLR multifrontal sparse solver. . . . . 29

3.1 Summary of existing scientific papers about iterative refinement. . . . . 56

4.1 Convergence conditions of LU-IR3 vs LU-GMRES-IR3. . . . . 69
4.2 List of convergence conditions. . . . . 78
4.3 List of limiting accuracies. . . . . 78

5.1 Convergence conditions of LU-GMRES-IR5. . . . . 90
5.2 Convergence conditions and limiting accuracies of LU-GMRES-IR5. . . . . 94
5.3 Number of LU solves for different variants of LU-IR3 and LU-GMRES-IR5. . 101

6.1 Set of large sparse matrices. . . . . 123
6.2 Execution time comparison between cast on the fly and cast in-place. . . . . 127
6.3 Execution time and memory consumption on our set of matrices. . . . . 128
6.4 Execution time and memory consumption with BLR on industrial matrices. 136
6.5 Supplement for Table 6.4. . . . . 137
6.6 Execution time and memory consumption with compressed active memory. 139
6.7 Execution time with static pivoting on industrial matrices. . . . . 141
6.8 Execution time with BLR and static pivoting on industrial matrices. . . . . 142
6.9 Best execution time and memory consumption. . . . . 144

7.1 Summary of existing scientific papers about mixed precision GMRES. . . . . 152

# List of Algorithms


2.1 Dense point LU factorization (without pivoting). . . . . 17
2.2 Dense tile forward substitution and backward substitution. . . . . 17
2.3 Dense tile LU (left-looking) factorization (without pivoting). . . . . 18
2.4 BLR LU factorization (without pivoting). . . . . 28
2.5 Arnoldi with MGS. . . . . 31
2.6 MGS-GMRES. . . . . 32
2.7 CGS and MGS. . . . . 33
2.8 Restarted GMRES. . . . . 33
2.9 Left- and right-preconditioned MGS-GMRES. . . . . 35
4.1 Generalized iterative refinement. . . . . 62
4.2 LU-IR3. . . . . 66
4.3 LU-GMRES-IR3. . . . . 68
4.4 QR-IR3. . . . . 71
4.5 AQR-IR3. . . . . 72
4.6 AQR-GMRES-IR3. . . . . 74
4.7 Squeezing a matrix to precision $u_f$. . . . . 77
5.1 LU-GMRES-IR5. . . . . 81
5.2 AQR-GMRES-IR5. . . . . 106
6.1 Complexities of LU-IR and LU-GMRES-IR. . . . . 113
7.1 Left-preconditioned MGS-GMRES in mixed precision. . . . . 153
7.2 M-GMRES-IR6. . . . . 163

# 1 Introduction

This thesis is essentially interested in computing efficiently, accurately, and reliably the solution of a large sparse square linear system

$$Ax = b, \qquad A \in \mathbb{R}^{n \times n}, \qquad b, x \in \mathbb{R}^n. \tag{1.1}$$

This linear algebra kernel is central for many applications, particularly in numerical simulations that require the solution of increasingly high-dimensional problems on supercomputers of increasing size and power.

Approximately every decade, the peak flops rate of the best supercomputer of the Top500 [2], listing the 500 first most powerful supercomputers in the world, is increased by three orders of magnitude. Just recently, in spring 2022, the Frontier supercomputer located at the Oak Ridge National Laboratory (USA) was announced to be officially the first supercomputer to attain the Exaflops. However, leveraging this power for real-life scientific computing applications is not straightforward. Primarily because Moore's law is approaching its end (Hennessy and Patterson [105]), and it becomes more and more challenging to increase the density of the transistors inside the processors. Consequently, the increasing power of modern supercomputers does not entirely rely anymore on the increasing speed and number of cores on a single chip but, instead, relies on two major design choices: first, the use of massive parallelism where one supercomputer gathers thousands of processors and, so, hundreds of thousands (sometimes millions) of cores; second, the use of specialized processing units that are extremely fast but only for a selected type of operations (e.g., Graphics Processing Units (GPUs), Google's Tensor Processing Units (TPUs), or low precision units); the cohabitation of these units of different capabilities in the same computer is referred to as heterogeneity. Therefore, to benefit from the power of modern supercomputers, it is essential to design algorithms to distribute the work amongst many cores of processing units of different types.

While parallelism will be discussed to some extent, this manuscript mainly focuses on improving the solution of (1.1) by harnessing the capabilities of a specific kind of processing unit: the low precision units. By using a smaller number of bits to represent the numbers and perform the operations, these low precision units offer a significant avenue to reduce the execution time, the memory, and the energy consumption but intrinsically worsen the

accuracy of the computations. Some applications, such as machine learning that can be very greedy in resources and for which these units were initially made, can settle for low accuracy. This is unfortunately not the case for many scientific computing problems; in particular, solving (1.1) exclusively in low precision would generally not achieve satisfactory accuracy on the solution with respect to the requirement of the application. This awakened the interest around mixed precision algorithms whose philosophy nowadays is to perform the high resource demanding parts of the computations in low precisions and recover the accuracy on the solution by making strategic use of more costly high precisions. We generally want the cost of the computations done in more expensive high precisions to stay limited, such that the application can benefit as much as possible from the low precision advantages.

In this manuscript, we aim at improving with mixed precision the two traditional choices of solvers for the solution of large sparse linear systems: direct and iterative solvers. Direct methods are based on a factorization of the matrix $A$ that is used to compute directly the solution $x$ of (1.1); they are generally considered very robust as they are capable of computing accurately and reliably the solution without the need for the user to provide suitable parameters nor understand how the solver works: direct methods work out of the box. They are also the methods of choice when the problem is numerically very difficult as they can generally still deliver the solution using a predictable amount of resources. However, these advantages come at the cost of high computational complexity. On the other hand, iterative methods compute iterates $x_i$ converging towards the solution $x$ of (1.1); they can compute the solution of a numerically simple problem with a far smaller amount of resources. Actually, for extremely high-dimensional problems, they might be the only possible approach because direct solvers might require an unbearable amount of resources to compute the solution. Their downsides are that their efficiency is very dependent on the numerical properties of the problem and the user parametrization. Note that there is a third kind of solver, the hybrid solver, that aims at combining the strengths of both direct and iterative methods (e.g., domain decomposition methods Dolean et al. [63], Agullo et al. [6] or block-projection methods Duff et al. [71]).

The oldest and the most well-known mixed precision approach for improving these two kinds of solvers is iterative refinement; this is the central algorithm of this document on which all our work is built upon. Iterative refinement refines a first inaccurate solution of (1.1) into a more accurate solution by applying a finite predictable number of refinement steps on top of a given linear solver. A major characteristic of iterative refinement is that the solution is guaranteed to be improved under some conditions on the matrix $A$ and the accuracy of the linear solver used. This algorithm is generally attributed to Wilkinson [213], who implemented it on the Automatic Computing Engine (ACE) in the 40s alongside his work with Alan Turing. Since then, iterative refinement has evolved continuously and has been proven very efficient in improving the performance of different kinds of linear solvers by targeting low precision computations (e.g., Langou et al. [136] for LU direct solver, Turner and Walker [205] for Krylov subspace based iterative solver, and Göddeke et al. [87] for multigrid solver).

Recently, important breakthroughs were achieved on iterative refinement combined

with direct solvers. Specifically, Carson and Higham [44; 45] used a new general analysis of iterative refinement to derive two new iterative refinement variants, both built on top of an LU direct solver and using up to three different precisions to produce fast, reliable, and highly accurate solution for (1.1). One of these variants uses a Generalized Minimal RESidual (GMRES) iterative solver preconditioned by the LU factors computed in low precision; all the operations within GMRES are carried out in the working precision, except for the matrix-vector products and the application of the preconditioner, which require the use of extra-precision. This so-called LU preconditioned GMRES-based iterative refinement in three precisions (LU-GMRES-IR3) is more resilient than the traditional iterative refinement variant to very low precision factorizations and ill-conditioned systems. Haidar et al. [101; 103; 102]) demonstrated that these new variants could efficiently take advantage of half precision within GPU tensor cores to accelerate up to a factor 4–5 a double precision direct solver for the solution of dense linear systems.

These significant performance improvements are extremely encouraging, but unfortunately, recent equivalent studies for the direct solution of large sparse linear systems are few, and most of them date back to the late 2000s (Buttari et al. [42], Baboulin et al. [29]). In particular, these studies do not cover the most recent forms of iterative refinement and sparse factorization, nor the new possibilities in terms of mixed precision offered by the recent hardware and software. For this reason, this document's first important concern is adapting and implementing the state-of-the-art iterative refinement methods for the efficient direct solution of sparse linear systems. To do so, we proceed in two steps.

First, we revisit LU-GMRES-IR3. Because the use of extra-precision for applying the LU factors in the preconditioned matrix–vector product can be expensive and especially unattractive if it is not available in hardware, this algorithm is unsuitable for many applications requiring the efficient parallel computation of the solution of a large sparse system. It is also the reason why existing implementations of LU-GMRES-IR3 targeting the solution of dense linear systems have not used extra-precision, despite the absence of error analysis for this approach. We propose to relax the requirements on the precisions used within GMRES, allowing the use of arbitrary precisions for applying the preconditioner and for the rest of the operations. We obtain the algorithm we call LU preconditioned GMRES-based iterative refinement in five precisions (LU-GMRES-IR5) on which we carry out a rounding error analysis that generalizes that of LU-GMRES-IR3.

Second, we implement these new state-of-the-art mixed precision iterative refinement variants on top of state-of-the-art sparse factorizations for the parallel solution of (1.1). In particular, we develop a new error analysis for these variants under a general model of LU factorization that accounts for the approximation methods typically used by modern sparse solvers, such as low-rank approximations or relaxed pivoting strategies. We then provide a detailed performance analysis of both the execution time and memory consumption of different iterative refinement algorithms based on different approximate sparse factorizations. Our performance study demonstrates considerable reductions in both time and memory consumption on large, sparse problems coming from a variety of real-life and industrial applications.

While these previous contributions primarily focus on the improvement of sparse di-

rect solvers, the other important concern of this manuscript is the improvement of the GMRES iterative solver for the solution of (1.1). Many efforts have already been devoted to the topic, and many different mixed precision strategies were proposed (Turner and Walker [205], Buttari et al. [42], Arioli and Duff [25], Gratton et al. [95] to cite a few). However, this high number of different specialized strategies lacks a consistent, shared, and up-to-date analysis that would serve as a general framework. Therefore, we propose to study an arbitrarily preconditioned GMRES that makes use of an iterative refinement process combined with a new mixed precision scheme for the computation of the preconditioned matrix–vector products; we call it GMRES-based iterative refinement with arbitrary preconditioner $M$ in six precisions (M-GMRES-IR6). This algorithm covers most of the previous mixed precision strategies for GMRES in addition to providing new ones of potential interest regarding the reduction of the resource consumption.

The remaining of this manuscript is about giving a comprehensive understanding of the central algorithm of our work: iterative refinement. To do so, we first provide a survey gathering, in chronological order, the different research studies on this algorithm. It allows us to describe its different evolutions and contextualize them in light of the hardware and software characteristics of different computing eras. Then, we consider the most recent forms of iterative refinement on which we provide finer technical details that are needed for a good understanding of the main contributions of this document.

Although solving sparse systems is the underlying focus of most of this manuscript's contributions, many of our findings also apply to the solution of dense systems.

This manuscript is organized as follows. In order to make this document self-contained, we provide in chapter 2 general background on floating-point rounding error analysis and sparse linear solvers. In chapters 3 and 4 we present, respectively, our chronological survey on iterative refinement and the technical details on a few of its most recent variants. In chapter 5 we introduce and study LU-GMRES-IR5, and in chapter 6 we present our performance analysis on large sparse linear systems. We present and study M-GMRES-IR6 in chapter 7. Finally, we provide our conclusions and final insights in chapter 8.

# 2 Background

For the proper understanding of this manuscript, we recall in this chapter all the fundamentals, key notions, and results on which we base our contributions.

The topics covered in this document are distributed over three main blocks. The first is floating point rounding error analysis used to study our mixed precision algorithms; we cover it in section 2.1. The second and the third are direct and iterative solvers that we both aim at improving through mixed precision; they are covered respectively in sections 2.2 and 2.3.

## 2.1 Floating-point arithmetic

The concept of mixed precision algorithm, which we investigate, is tightly related to the floating-point representations of numbers in computers and the intrinsic rounding errors generated by their computations. For this reason, we recall the basics of floating-point arithmetic and the standard rounding error model in section 2.1.1. We present some commonly available floating-point arithmetics that we will use throughout this manuscript in section 2.1.2. We discuss the recent and rapid evolution of low precisions motivating the use of mixed precision algorithms in section 2.1.3. We finally present the different notations and notions that will be in use for the error analyses carried out all over this manuscript in section 2.1.4.

### 2.1.1 Basics

*2.1.1.1 Floating-point numbers and various definitions.* The *floating-point arithmetic* is an inexact arithmetic aiming at approximately representing real numbers with a *significand* scaled by an *exponent*, both encoded with a fixed number of digits in a given base. In particular, we call a *bit* a base-two/binary digit. A floating-point number $y$ is of the form

$$y = \pm m \times \beta^{e-t}, \tag{2.1}$$

where

- $\beta$ is the base (usually base-two/binary in computers).
- $\pm$ is the sign. It can be carried on one bit in binary base; for example, 0 represents a positive number, while 1 represents a negative one.
- $m$ is the significand (or mantissa). It is a positive integer carrying the $t$ significant digits of the number; therefore, $m$ satisfies $0 \leq m \leq \beta^t - 1$.
- $e$ is the biased exponent. It is a positive integer encoded on a given number of digits and biased to reach negative values; it is bounded by $e_{\min} \leq e \leq e_{\max}$.

We can also express a floating-point number $y$ by the equivalent form

$$y = \pm\beta^e \left( \frac{d_1}{\beta} + \frac{d_2}{\beta^2} + \cdots + \frac{d_t}{\beta^t} \right) = \pm\beta^e \times .d_1 d_2 \ldots d_t, \tag{2.2}$$

where each digit satisfies $0 \leq d_i \leq \beta - 1$.

We define a *floating-point number system* as the finite subset of the real numbers, noted $\mathscr{F} \subset \mathbb{R}$, composed of the representable numbers by (2.1) for a given base $\beta$ and given integers $t$, $e_{\min}$, and $e_{\max}$. Defined as such, the representation of a given $y \in \mathscr{F}$ is not necessarily unique; for example, for a system $\mathscr{F}$ characterized by $\beta = 10$ and $t = 5$, both ($m = 01234$, $e = 1$) and ($m = 12340$, $e = 0$) can represent $y = 0.1234$. To ensure the uniqueness of the representation, we assume for all $y \neq 0$ that $d_1 \neq 0$ or equivalently $m \geq \beta^{t-1}$; the system $\mathscr{F}$ is then normalized. In computers with binary base, this normalization is translated by having a phantom bit $d_1$ not directly stored and implicitly set to 1.

On a computer with a given floating-point number system $\mathscr{F}$, any $x \in \mathbb{R}$ is represented by some close value $\mathrm{fl}(x) \in \mathscr{F}$ (the chosen value depends on the rounding strategy: for example round to nearest, stochastic rounding, and others).

Two central properties are related to $\mathscr{F}$. The first one is the *range* of representation of the numbers, that is, the maximum and minimum non-negative numbers in $\mathscr{F}$. The range of a normalized system $\mathscr{F}$ is mainly defined by the maximum and minimum representable exponents $e_{\max}$ and $e_{\min}$; thus, as $\beta^{t-1} \leq m \leq \beta^t - 1$, for all $y \in \mathscr{F} \backslash \{0\}$ we have $\beta^{e_{\min}-1} \leq |y| \leq \beta^{e_{\max}}(1 - \beta^{-t})$. For instance, consider the case of the IEEE fp32 arithmetic using a binary base with 8 bits in the exponent and a -127 bias. As the exponent values -127 and +128 are reserved for special numbers, we have $e_{\min} = -126$ and $e_{\max} = +127$, and so, for all $y \in \mathscr{F}$ we have $10^{-38} \lesssim |y| \lesssim 10^{+38}$. Naturally, the greater the number of bits in the exponent, the larger the range. In the situation where $x \in \mathbb{R}$ is outside of the range of representation of $\mathscr{F}$, the usual way for the computers to handle this is:

- For *underflow*, which is the case where the number to represent is smaller in magnitude than the smallest representable value, $x$ is rounded to 0.
- For *overflow*, which is the case where the number to represent is higher in magnitude than the highest representable value, $x$ is rounded to $\pm$inf.

Underflows can also be handled with the addition of the subnormal numbers in the system $\mathscr{F}$. They are denormalized numbers having the minimum exponent (i.e., $d_1 = 0$ and $e = e_{\min}$) and fill the gap between $\beta^{e_{\min}-1}$ and 0, pushing the minimum positive value further to $\beta^{e_{\min}-t}$. Underflows and overflows can be challenging issues, in particular in cases where the range of $\mathscr{F}$ is small.

The second important property is the *unit roundoff* of $\mathscr{F}$. It corresponds to the maximum relative error made by the representation $\mathrm{fl}(x) \in \mathscr{F}$ of $x \in \mathbb{R}$. The unit roundoff noted $u$ is determined by the number of digits $t$ in the significand and is defined as $u = \beta^{1-t}/2$. It satisfies for all $x \in \mathbb{R}$ lying in the range of $\mathscr{F}$

$$\mathrm{fl}(x) = x(1+\delta), \quad |\delta| \leq u. \tag{2.3}$$

We can interpret the unit roundoff as a measure of the accuracy of a given floating-point arithmetic. For example, in the case of the IEEE fp32 arithmetic, there are 24 bits in the significand (including one implicit bit for the normalization), giving $u = 2^{1-24}/2 \approx 5.96 \times 10^{-8}$. Naturally, a higher number of digits $t$ leads to a smaller $u$ and better accuracy.

*2.1.1.2 Standard error model.* Because the result of an operation between two floating-point numbers $x, y \in \mathscr{F}$ is not necessarily in $\mathscr{F}$, the process of rounding the result to a close element in $\mathscr{F}$ introduces error in the computation. These errors can significantly deteriorate the final result to some degree where the computed result might be unusable.

To be able to study these *rounding errors*, we are using the standard error model of floating-point arithmetic defined by Higham [113, sect. 2.2] and in which $x, y \in \mathscr{F}$ satisfy

$$\mathrm{fl}(x \,\mathrm{op}\, y) = (x \,\mathrm{op}\, y)(1+\delta), \quad |\delta| \leq u, \quad \mathrm{op} \in \{+, -, *, /\}, \tag{2.4}$$

where $u$ is the unit roundoff of $\mathscr{F}$. Throughout the manuscript, we use the notation $\mathrm{fl}(\cdot)$ to denote a given expression's computed value in floating-point arithmetic. We add a "hat" on exact quantities to denote their computed counterpart; for example, if we denote the true solution as "$x$", its computed counterpart is denoted "$\widehat{x}$".

The IEEE 754 standard, published in 1985 [127], defines binary floating-point arithmetics implementations that are portable, reliable, and allow for reproducibility across computers that implement it. The standard specifies in particular floating-point arithmetics systems (e.g., IEEE fp32 and fp64 composed respectively of 32 and 64 bits distributed over the exponent, significand, and the sign), but also rounding rules, basic operations, or handling of exceptions. In particular, the standard error model (2.4) is valid for the arithmetic satisfying the IEEE standard (but is also valid for the rest of the arithmetics we will use in this manuscript).

*2.1.1.3 Other arithmetics.* In this manuscript, we mainly focus our discussion around floating-point arithmetics because this is the number representation on which we based our work. However, note that there exists other number representations, such as fixed-point or posit numbers, that are also commonly used.

### 2.1.2 Commonly available floating-point arithmetics in computers

In Table 2.1, we present different binary floating-point arithmetics that can be found in computers nowadays, either directly supported by the hardware or sometimes emulated in the software. They are ordered by decreasing unit roundoff, and, for convenience, we will often refer to them in this manuscript by their respective symbols (e.g., D for fp64).

Table 2.1: Parameters for floating-point arithmetics: symbol used in this manuscript, number of bits in the significand, number of bits in the exponent, unit roundoff, and range.

| Arithmetic | Symbol | Significand | Exponent | Unit roundoff | Range |
|---|---|---|---|---|---|
| fp128 | Q | 112 (+1) | 15 | $9.63 \times 10^{-35}$ | $10^{\pm 4932}$ |
| double-fp64 | D D | 106 (+1) | 11 | $6.16 \times 10^{-33}$ | $10^{\pm 308}$ |
| fp64 | D | 52 (+1) | 11 | $1.11 \times 10^{-16}$ | $10^{\pm 308}$ |
| fp32 | S | 23 (+1) | 8 | $5.96 \times 10^{-8}$ | $10^{\pm 38}$ |
| tfloat32 | T | 10 (+1) | 8 | $4.88 \times 10^{-4}$ | $10^{\pm 38}$ |
| fp16 | H | 10 (+1) | 5 | $4.88 \times 10^{-4}$ | $10^{\pm 5}$ |
| bfloat16 | B | 7 (+1) | 8 | $3.91 \times 10^{-3}$ | $10^{\pm 38}$ |
| fp8 (E4M3) | R | 3 (+1) | 4 | $6.25 \times 10^{-2}$ | $10^{\pm 2}$ |
| fp8 (E5M2) | R* | 2 (+1) | 5 | $1.25 \times 10^{-1}$ | $10^{\pm 5}$ |

In particular, the fp128 quadruple, the fp64 double, the fp32 single, and the fp16 half precisions are covered by the latest revision (2019) of the IEEE 754 standard [127].

While hardware widely supports fp64 and fp32, fp128 quadruple precision is almost exclusively implemented in software. For instance, we can access it through the GCC compilers using the Quad-Precision Math Library [84] or the Intel Fortran Compiler. Due to the software implementation, the use of this arithmetic is generally extremely expensive: about an order of magnitude slower than double-fp64 precision (see Higham [114]). It can be noted that some hardware, such as the IBM power9 processor [126] and the IBM z13 processor [145], supports fp128 natively.

The double-fp64 arithmetic (or double-double) is an alternative to fp128; it represents a number by a pair of IEEE fp64: one represents the higher order bits of the significand and the other the lower order bits. Thus, it leads to $2 \times 53 + 1$ (implicit) $= 107$ bits for the significand, and double-double is therefore relatively close to the fp128 accuracy. The range of double-double remains identical to fp64 and is substantially smaller than the fp128 one. However, while being less accurate and narrower than fp128, manipulating numbers in this precision reduces to a sequence of fp64 operations supported by the hardware and, so, can run faster than software implemented fp128 (Li et al. [144]).

Interest and support for fp16 arithmetic in scientific computing started relatively recently, during the 2010s, even though earlier hardware implementations of 16-bit floating-point numbers already existed (e.g., Hitachi's HD61810 DSP 1982 [122], GeForce FX 2003 [210]). Interestingly, it was first mainly integrated on GPUs; for example, it began to be supported and accessible through the CUDA toolkit from the NVIDIA Maxwell architectures (2015) and was integrated into the subsequent Pascal, Volta, Turing, Ampere and the upcoming Hopper architectures. It has also been integrated into the AMD GCN and CDNA GPU ar-

chitectures. Its integration and accessibility on Central Processing Units (CPUs) are even more recent. For example, some versions of the ARM architectures support it (ARMv7 and ARMv8 [62]), in particular, Fujitsu's A64FX CPUs that are installed on the Fugaku supercomputer (#2 TOP500 [2] in 2022, Japan) and the Raspberry Pi are based on these ARM architectures. In addition, it is planned to be used in Intel's Sapphire Rapids that will be installed on the Aurora exascale supercomputer (note that it has been available as a storage format from Intel's Ivy Bridge architecture).

Some of the arithmetics in the list are not covered by the IEEE 754 standard and were promoted and implemented by hardware constructors. The bfloat16 arithmetic [1] (also called the "Brain floating-point format") has been designed by Google Brain to be used in Google's TPU AI accelerators (Norrie et al. [165]). Other constructors have also adopted it; for instance, it is present on Intel's Nervana NNP-L1000 AI accelerators, on Intel's Cooper Lake architecture (and is planned to be on Intel's Sapphire Rapids), on the ARM architectures from the ARMv8.6-A revision, and it has been introduced in NVIDIA's GPUs from the Ampere architecture.

The tfloat32 arithmetic [167] has been designed by NVIDIA and has been implemented as part of the tensor core units of the Ampere GPU (Choquette et al. [49]).

The new fp8 (E4M3 or E5M2) arithmetics were announced in 2022 by NVIDIA to be supported as part of the tensor core units of their new Hopper GPU [166].

The arithmetics at the bottom of Table 2.1 (i.e., fp32, tfloat32, fp16, bfloat16, and fp8), using a smaller number of bits and having a reduced accuracy and range, are the so-called low precisions; they play an essential role in the contributions of this manuscript. More is said about their properties, differences, and trade-offs in the next section 2.1.3.

### 2.1.3 Low precision (floating-point) arithmetics

*2.1.3.1 Benefits.* The *low precision (floating-point) arithmetics*, or simply low precisions, refer to arithmetics using a small number of bits to represent numbers. While having fewer bits for the representation of the exponent and the significand leads to a smaller range and a decreased accuracy (i.e., a larger unit roundoff), using low precisions provides however three major benefits:

- Because the storage is proportional to the total number of bits, storing the numbers in low precision decreases the memory consumption.
- Because the speed of computations in processing units can be at least proportional to the total number of bits and because data movements and communications are all proportional to the total number of bits, computing with low precision reduces computation times for both compute- and memory-bound applications.
- Because the power consumption is approximately proportional to the square of the number of significand bits, using low precision decreases the energy consumption. For instance, fp16 and tfloat32 (11 bits in the significand) consume five times less energy than fp32 (24 bits), and bfloat16 (8 bits) consumes nine times less energy than fp32.

Reducing time, memory, and energy consumption are all challenging objectives for the ease of high performance computing. Consequently, as low precisions can significantly decrease these expensive resources, their use is becoming a major avenue of research.

In this manuscript, as many of our targeted applications require fp64 accuracy for the solution of (1.1), we will consider that every precision below fp32 (included) in Table 2.1 is a low precision. We represent these arithmetics in Figure 2.1.

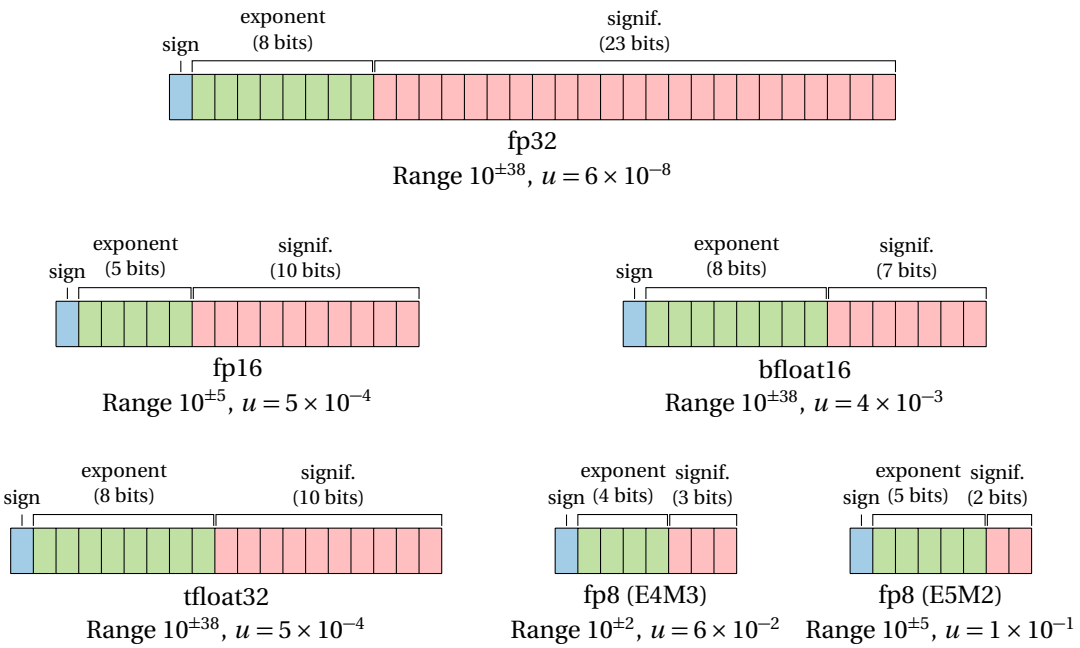



Figure 2.1: Bits distribution on low precision arithmetics.

*2.1.3.2 Challenges.* All these low precisions have different trade-offs between the number of bits, the accuracy, and the range. In particular, both fp16 and bfloat16 have 16 bits, but these bits are distributed differently: bfloat16 has more bits for the exponent and, thus, a better range than fp16; fp16 has more bits for the significand and, thus, a better accuracy. These two design choices illustrate the two different challenges that low precisions face: narrow range and low accuracy.

The loss of accuracy when using low precision is probably the most apparent source of challenges. For some applications, such as machine learning, the low accuracy does little or no deterioration to the quality of the results as said by Dean [56] "*Deep learning models [⋯] are very tolerant of reduced-precision computation*". Therefore, because neural networks can be extremely greedy in resources, the use of low precisions has become prevalent for these applications; a founding work on the topic is from Courbariaux et al. [52]. Because of its massive popularity and efficient combination with low precisions, machine learning has actually been the main driver of the recent increase in support for these arithmetics. However, many scientific computing applications require a high accuracy on the computed solution and, consequently, low precisions cannot be used right out of the box. For example, solving (1.1) entirely in fp16 with Gaussian Elimination only yields a solution with an

error at best of order $10^{-4}$ (cf. Theorem 2.6), which is unsatisfactory in many cases. For this reason, leveraging low precision power for solving scientific computing problems is not straightforward. In particular, using low precisions for solving (1.1) while meeting a reasonable accuracy on the solution is the raison d'être of this manuscript.

Regarding the challenges related to the range, the fp16 case is interesting because, while many linear algebra applications can accommodate the fp32 $10^{\pm 38}$ range for a well-scaled problem, the fp16 $10^{\pm 5}$ range is relatively narrow and can be an issue. In the case of a linear system solution, underflows can cause a severe loss of information and, at worst, could transform a solvable linear system into a singular problem. Overflows are unrecoverable because we cannot produce a satisfactory solution when there are infinities among the entries. Consequently, in general, targeting narrow range precisions requires special care to avoid overflows and limit as much as possible underflows (e.g., adopt a scaling strategy such as in Higham et al. [121]). This is why bfloat16 is designed to prevent this problem by trading off bits in the significand.

The tfloat32 arithmetic opts not to compromise between fp16 and bfloat16 by having a total of 19 bits. It provides the same range as bfloat16 and the same accuracy as fp16 but is twice as slow as fp16 and bfloat16 on NVIDIA GPUs. With the newest fp8 arithmetics (E4M3 and E5M2) that present extremely narrow ranges and low accuracies, the previous issues are even more exacerbated, and leveraging the power of these arithmetics in scientific computing applications will be a serious challenge.

It is important to clarify that the low precisions are not uniformly accessible across the processing units and the memories. In particular, they are generally developed and available on accelerators (e.g., NVIDIA and AMD GPUs or Google's TPUs); sometimes, they are even only available through specific units of these accelerators (e.g., tfloat32 and fp8 are only on NVIDIA GPU's tensor cores). On the other hand, most of the CPUs in supercomputers nowadays cannot target precision lower than fp32, so the use of low precisions on these units is relatively limited for now. Consequently, the choice of precisions in an algorithm is intrinsically linked to the hardware used to run this algorithm, where the hardware used is also highly dependent on the ability of the algorithm to leverage efficiently its computing power.

*2.1.3.3 Mixed precision.* The goal of present-day mixed precision strategies is to fix the loss of accuracy issue of the low precisions while trying to keep its benefits as much as possible. It can be made by observing that not all operations contribute equally to the final accuracy of the solution. Thus, if we can identify inexpensive essential parts of the computation that, if done in high precisions, can recover or preserve the accuracy of the solution, it may be possible to run the high resource-demanding parts of the computations in low precisions without damaging much the quality of the solution. As several different precisions are in play, we call such an algorithm a *mixed precision algorithm*.

Mixed precision algorithms began to flourish in the 2000s when IEEE fp32 single precision computations became effectively at least two times faster than IEEE fp64 double precision ones, the article of Langou et al. [136] is a well-known first representative of this period. In particular, the recent developments around the half fp16/bfloat16/tfloat32 pre-

cisions (and the quarter fp8 E4M3/E5M2 precisions) stimulated even more research in the area. This is why a large amount of effort has been produced on the topic in recent years. The excellent surveys of Higham and Mary [119], Abdelfattah et al. [3] cover most of them. A mixed precision algorithm can have different forms: it can be about doing some computations in one or several other different precisions (e.g., Carson and Higham [45], Gratton et al. [95]); but it can also be about storing and accessing some data in another precision (e.g., Anzt et al. [24], Göbel et al. [85]); or using pieces of hardware that are using internally different precisions (e.g., Haidar et al. [102], Lopez and Mary [152] with GPU tensor cores).

Elaborating mixed precision algorithms requires two main components. First, we need to identify the operations of an algorithm that could run in low or high precisions through rounding error analysis. Second, we need to face implementation questions to effectively leverage low precision advantages, such as availability of the precisions (in hardware and software), copies and access of data in different precisions, or overflow and underflow. Both of these components will be explored in this manuscript.

We now provide some vocabulary around mixed precision that we will often use in this manuscript.

By *working precision* we mean the precision at which the computed solution will be handed back to the user. It often implicitly means that we target a computed solution in working accuracy, so if the working precision is fp64, we expect the forward error and backward error to be a function of the unit roundoff of the fp64 precision. In the case of the solution of (1.1), it means a residual of order $10^{-16}$ (i.e., the fp64 unit roundoff) or at least smaller/better than $10^{-8}$ (i.e., the fp32 unit roundoff). From this definition, we consider that a low precision in a given algorithm is a precision using fewer bits than the working precision. Note that in the literature, its precise meaning might depend on the context and might differ from this definition.

In general, when we present and study a mixed precision algorithm, the different precisions are unspecified independent parameters. Therefore, we refer to a fixed specified set of these precisions as a *combination of precisions*. For instance, if the algorithm has two independent precisions $u_1$ and $u_2$, then $(u_1 = \mathrm{D}, u_2 = \mathrm{S})$ is a combination of precisions of this algorithm.

We say a combination of precisions in an algorithm is *meaningful* if none of the precisions it employs can be reduced without degrading the numerical properties (e.g., convergence or accuracy). For example, if we consider an algorithm with two independent precisions $u_1$ and $u_2$, if switching $u_1$ from fp64 to fp32 with $u_2$ fixed provides the same numerical properties, $u_1$ in fp64 is considered not meaningful. The sense of "meaningfulness" is to always prefer the use of low precisions over high precisions. It should be noted that this concept does not take more hardware or computer properties into account. Instead, it is purely numerical and serves as a first filter to discriminate potential combinations of interest. It means that a meaningful combination of precision is not necessarily relevant in a given applicative context, and further filtering with practical constraints should be considered.

### 2.1.4 Rounding error analysis notations

The inexact representation of the numbers and the related inexact computations lead to intrinsic rounding errors and their potential accumulation throughout the algorithm. In some cases, the final output can be hugely affected and distant from the solution we would obtain with exact arithmetic. To better understand the impact of these errors and the role of each precision in a mixed precision algorithm, an essential part of the work presented in this manuscript is about carrying rounding errors analyses on particular algorithms of interest. We now list the tools, notions, and notations that will be used in that regard.

It is fundamental to quantify the quality of a certain computed solution $\widehat{x}$ of the linear system (1.1). We can use two measurements to define the accuracy of $\widehat{x}$. The first is the *forward error* (or relative error) evaluated as $\|x-\widehat{x}\|/\|x\|$; this is a measure of the distance between the computed solution $\widehat{x}$ and the exact solution $x$. Naturally, in a practical context, $x$ is unknown, and the forward error is not directly computable. The second is the *backward error* which quantifies how much we should perturb the linear system for the computed solution $\widehat{x}$ to be the actual exact solution of this perturbed system. The backward error of (1.1) can therefore be defined as a measure of the smallest perturbations $\Delta A$ and $\Delta b$ such that $(A+\Delta A)\widehat{x} = b+\Delta b$. To measure it, we use the normwise relative formulation of Rigal and Gaches [178]:

$$\min\{\epsilon : (A+\Delta A)\widehat{x} = b+\Delta b,\ \|\Delta A\| \leq \epsilon\|A\|,\ \|\Delta b\| \leq \epsilon\|b\|\} = \frac{\|b-A\widehat{x}\|}{\|A\|\,\|\widehat{x}\|+\|b\|}. \tag{2.5}$$

We will consider an algorithm for solving (1.1) *normwise backward stable* if, for any $A$ and $b$, it is guaranteed to produce a small backward error (2.5). In our context, what we mean by "small" is a backward error of order the unit roundoff of the precision used by the algorithm. When the algorithm is in mixed precision, then, by small, we mean a backward error of order the unit roundoff of the working precision. Also, in this manuscript, when we just refer to the "stability" of an algorithm, we often mean the normwise backward stability of this algorithm.

Another fundamental quantity is the *condition number* or conditioning of (1.1). It carries the sensitivity of the solution to perturbations in the data; thus, the higher the condition number, the more we will generate a high relative error on the computed solution of (1.1). Different measures of the condition number exist; we will use the following in this manuscript. For a given nonsingular square matrix $A \in \mathbb{R}^{n\times n}$, we define the componentwise condition numbers

$$\mathrm{cond}(A,x) = \frac{\||A^{-1}||A||x|\|}{\|x\|}, \quad \mathrm{cond}(A) = \||A^{-1}||A|\|, \tag{2.6}$$

where $|A| = (|a_{ij}|)$, and we define the normwise condition number as $\kappa(A) = \|A^{-1}\|\|A\|$. The matrix $A$ is called *well-conditioned* if its condition number is small or *ill-conditioned* if it is large. What is small or large depends on the context. For a non-square matrix $A \in \mathbb{R}^{m\times n}$, if we note its pseudoinverse $A^{\dagger}$, we define its normwise condition number as $\kappa(A) = \|A^{\dagger}\|\|A\|$.

Note that by definition the backward error of a computed solution $\widehat{x}$ of (1.1) is always

smaller than the forward error, we have the following relation between the two

$$\frac{\|b-A\widehat{x}\|}{\|A\|\|\widehat{x}\|+\|b\|} \leq \frac{\|x-\widehat{x}\|}{\|x\|} \leq \kappa(A)\frac{\|b-A\widehat{x}\|}{\|A\|\|x\|}. \tag{2.7}$$

As different floating-point arithmetics can be used in the same algorithm, we use a subscript on the unit roundoff notation to differentiate them; for example, $u_f$. The working precision is the only precision that does not have a subscript and is simply noted $u$. Moreover, for convenience, we will abusively use the notation $u$ to refer to both the floating-point arithmetic and its unit roundoff, depending on the context.

For any integer $k$ we define

$$\gamma_k = \frac{ku}{1-ku}. \tag{2.8}$$

A superscript on $\gamma$ denotes that $u$ carries that superscript as a subscript; thus $\gamma_k^f = ku_f/(1-ku_f)$, for example. We also use the notation $\widetilde{\gamma}_k = \gamma_{ck}$ to hide modest constants $c$.

The error bounds obtained by our different analyses depend on different constants of the problem, such as its dimension $n$ or the number of iterations in the iterative solver (if used). We will gather these constants into a generic function $f(\cdot)$ in parts of the analyses. In our algorithms, the constants depending on $n$ are known to be pessimistic (Connolly et al. [50], Higham and Mary [116; 117]), as are the other constants that will be gathered in $f(\cdot)$. Therefore, for the sake of readability, we do not always keep track of the precise value of $f(\cdot)$. When we drop constants $f(\cdot)$ from an inequality, we write the inequality using "$\ll$". A convergence condition expressed as "$\kappa(A) \ll \theta$" can be read as "$\kappa(A)$ is sufficiently less than $\theta$". Finally, we also use the notation $\lesssim$ and $\approx$ when dropping negligible second order terms in the error bounds; we note that what makes a second order term negligible depends on the local context: for example, the term $u_f^2$ is not necessarily negligible in the expression $u+u_f^2$ for arbitrary precisions $u$ and $u_f$, but can be safely dropped from the expression $u_f+u_f^2$.

We will also use the notation $\equiv$, which means that we can take the quantity on the left, which is in our control and is not fixed, to be equal to the quantity on the right.

Our error analyses use different matrix norms for $A \in \mathbb{R}^{m\times n}$: the $\infty$-norm, the Frobenius norm, and the 2-norm defined respectively as

$$\|A\|_\infty = \max_{1\leq i\leq m}\sum_j^n |a_{i,j}|, \qquad \|A\|_F = \sqrt{\sum_{i,j=1}^n |a_{i,j}|^2}, \qquad \text{and} \qquad \|A\|_2 = \sigma_{\max}(A), \tag{2.9}$$

where $\sigma_{\max}$ is the largest singular value of matrix $A$. These norms are equivalent and satisfy the following inequalities

$$\|A\|_2 \leq \|A\|_F \leq \sqrt{\operatorname{rank}(A)}\|A\|_2, \tag{2.10a}$$

$$\frac{1}{\sqrt{m}}\|A\|_F \leq \|A\|_\infty \leq \sqrt{n}\|A\|_F, \tag{2.10b}$$

$$\frac{1}{\sqrt{n}}\|A\|_\infty \leq \|A\|_2 \leq \sqrt{m}\|A\|_\infty. \tag{2.10c}$$

In addition, we write $\kappa_\infty(A)$, $\kappa_F(A)$, and $\kappa_2(A)$ the corresponding condition numbers of $A$ based on these respective norms. We will use unsubscripted norms or condition numbers when the constants depending on the problem dimensions have been dropped since the norms are equivalent.

We denote by $p$ the maximum number of nonzeros in any row of $[A\,b]$, $A$, and $b$ from (1.1); thus $p = n+1$ for a dense matrix $A$ and a vector $b$.

This manuscript summarizes the results of rounding error analyses into theorems. In general, they will guarantee bounds on the errors of the computed solution, sometimes under some conditions. As an example, in Theorem 2.1, we present a fundamental result of the rounding error analysis of the classic matrix–vector product kernel $y = Ax$ computed in precision $u$.

**Theorem 2.1** (From [113, eq. (3.11)])**.** *Given a matrix $A \in \mathbb{R}^{m\times n}$ and a vector $x \in \mathbb{R}^m$, the computed matrix–vector product satisfies*

$$\widehat{y} = (A+\Delta A)x, \qquad |\Delta A| \leq \gamma_n |A|. \tag{2.11}$$

Due to rounding errors, the computed $\widehat{y}$ is not exactly $Ax$ but is rather $(A+\Delta A)x$ where $\Delta A$ is a perturbation or error. The theorem ensures that the entries of $\Delta A$ are bounded by the unit roundoff of the precision $u$, the dimension of the problem $n$, and the entries of $A$ if the computation satisfies the standard error model (2.4), which is an implicit assumption all over this manuscript. Furthermore, as we can consider the constant in $n$ pessimistic, this theorem shows that the error $\Delta A$ will stay relatively small compared with the entries of $A$, where small is defined by the unit roundoff of the precision used.

## 2.2 Direct solvers

A *direct solver* for the solution of the general square linear system (1.1) is a solver that reduces the problem to a sequence of systems that are easy to solve (e.g., triangular system). Direct solvers are often considered robust and easy to use because they can achieve backward stability with the addition of some techniques (e.g., good pivoting strategies), their performance is not affected by the spectral properties of the matrix, and their resource consumption is predictable. The main downsides of direct solvers are the high computational cost and memory consumption they require, particularly for the solution of sparse linear systems.

A big part of the work presented in this manuscript is interested in addressing these weaknesses by the use of mixed precision with a specific focus on the sparse case. Therefore, in this section, we recall the essential elements of background on dense and sparse direct solvers that are needed to carry out the different rounding error and performance analyses. More specifically, we mainly focus on the LU direct solver (and to some extent the QR one) on which we recall some classic rounding error stability results in sections 2.2.1 and 2.2.2. Then, we present the multifrontal sparse direct solver and some of its popular numerical approximation techniques that we will use for the parallel solution of large sparse linear systems on supercomputers in sections 2.2.3 and 2.2.4.

### 2.2.1 LU solver

*2.2.1.1 Gaussian elimination and LU factorization.* *Gaussian Elimination (GE)* is the process of transforming (1.1) into a simpler triangular linear system. There are $n-1$ steps beginning with $A^{(1)} = A$ and $b^{(1)} = b$ and finishing with the upper triangular system $A^{(n)}x = b^{(n)}$, where we build iterate $A^{(k)}$ and $b^{(k)}$ by

$$a_{i,j}^{(k+1)} = a_{i,j}^{(k)} - \ell_{i,k} a_{k,j}^{(k)}, \qquad i = k+1:n,\ j = k+1:n, \tag{2.12}$$

$$b_i^{(k+1)} = b_i^{(k)} - \ell_{i,k} b_k^{(k)}, \qquad i = k+1:n, \tag{2.13}$$

with $\ell_{i,k} = a_{i,k}^{(k)} / a_{k,k}^{(k)}$. We call step k the *elimination* of the $k$th variable and $a_{k,k}^{(k)}$ a *pivot*.

The LU direct solver is just the matrix formulation of Gaussian Elimination (GE) for general square linear system. It is about expressing $A$ as a product of a unit lower triangular matrix $L$ and an upper triangular matrix $U$ such that $A = LU$ and where

$$L = \begin{bmatrix} 1 & & & & \\ \ell_{2,1} & \ddots & & & \\ & & 1 & & \\ \vdots & & \ell_{k+1,k} & \ddots & \\ & & \vdots & & \ddots \\ \ell_{n,1} & & \ell_{n,k} & & 1 \end{bmatrix}, \qquad U = A^{(n)}. \tag{2.14}$$

$L$ and $U$ are called the *factors* of $A$, and the process of computing them is called the *factorization*. Computing the solution $x$ of (1.1) is then done in two stages,

$$Ly = b \quad (\textit{forward substitution}), \tag{2.15}$$

$$Ux = y \quad (\textit{backward substitution}), \tag{2.16}$$

that we call the *solve*. For a dense system, the computation of the factorization requires $\mathcal{O}(n^3)$ flops while the solve requires $\mathcal{O}(n^2)$ flops. Moreover, the LU direct solver stores $\mathcal{O}(n^2)$ entries in memory if the factorization is done in-place; that is, $A$ is overwritten, its lower triangular part is replaced by $L$ and its upper triangular part by $U$.

Algorithms 2.1 and 2.2 described the dense point LU factorization and solve. At each step $k$ of the factorization, a new column of $L$ and a new row of $U$ are computed in-place. Note that the diagonal of $L$ is not explicitly stored since $l_{k,k} = 1$ for all $1 \le k \le n$. This algorithm is referred to as "point" because the operations are performed on single matrix entries. It can be reorganized to be performed on blocks of entries instead; it significantly improves the performance due to the better data locality.

Algorithm 2.3 is a formed of blocked LU factorization that we call dense tile LU factorization. This algorithm partitions the matrix $A$ into blocks or tiles stored contiguously in memory; for example, we can partition the matrix $A$ with uniform tiles of size $b \ll n$ composed of $p = n/b$ tiles on each row and column. In particular, expressing the problem this way brings more concurrency between the operations that can be efficiently parallelized

**Algorithm 2.1** Dense point LU factorization (without pivoting)

**Input:** a $p \times p$ block matrix $A$ of order $n$.
**Output:** its computed LU factors.

1: **for** $k = 1 : n-1$ **do**
2: $\quad a_{k+1:n,k} \leftarrow a_{k+1:n,k} / a_{k,k}$
3: $\quad a_{k+1:n,k+1:n} \leftarrow a_{k+1:n,k+1:n} - a_{k+1:n,k} a_{k,k+1:n}$
4: **end for**

**Algorithm 2.2** Dense tile forward substitution (left) and backward substitution (right)

**Input:** a $n \times n$ unit lower triangular matrix $L$ and a right-hand side $b$.
**Output:** a computed solution to $Ly = b$.

1: $y \leftarrow b$
2: **for** $k = 1 : n$ **do**
3:
4: $\quad$ **for** $i = k+1 : n$ **do**
5: $\quad\quad y_i \leftarrow l_{i,k} y_k$
6: $\quad$ **end for**
7: **end for**

**Input:** a $n \times n$ upper triangular matrix $U$ and a right-hand side $y$.
**Output:** a computed solution to $Ux = y$.

1: $x \leftarrow y$
2: **for** $i = n-1 : -1 : 1$ **do**
3: $\quad$ **for** $k = i+1 : n$ **do**
4: $\quad\quad x_i \leftarrow u_{i,k} x_k$
5: $\quad$ **end for**
6: $\quad x_i \leftarrow x_i / u_{i,i}$
7: **end for**

through task-based multithreading. As the computational performance of the dense LU direct solver is not the concern of this manuscript, we refer the reader to the work of Buttari et al. [43], Quintana-Ortí et al. [177] for more information on efficient parallel implementations of tile LU factorization. Note that we refer to this algorithm as "left-looking" to be consistent with the studies we base our contributions on while acknowledging that this is actually a "Crout" variant.

Algorithm 2.3 performs the factorization of a $p \times p$ tiled matrix $A$ by applying the following three steps at each iteration $k$ for the elimination of the $k$th row and column of the tiled matrix: we update the $k$th row and column with the already computed rows and columns, we factor the diagonal tile $A_{k,k}$, and finally, we use the computed factors to apply the different triangular solves on the remaining of the $k$th row and column to finish the elimination. These three steps are represented by Figure 2.2.

*2.2.1.2 Stability.* Naturally, the presence of rounding errors will affect the quality of the computed solution $\widehat{x}$ by the LU direct solver. Higham [113, chap. 9] provides fundamental results that we recall below on the numerical stability of the LU direct solver.

**Theorem 2.2** (From [113, thm. 9.3])**.** *If GE applied to $A \in \mathbb{R}^{m \times n}$ $(m \geq n)$ in precision $u$ runs to completion, then the computed LU factors $\widehat{L} \in \mathbb{R}^{m \times n}$ and $\widehat{U} \in \mathbb{R}^{n \times n}$ satisfy*

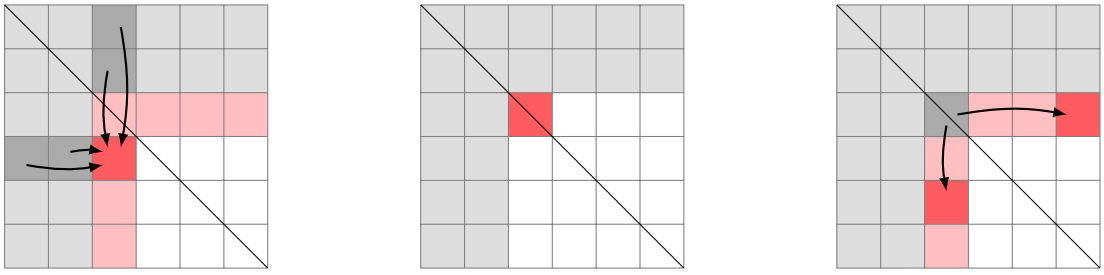
$$\widehat{L}\widehat{U} = A + \Delta A, \qquad |\Delta A| \leq \gamma_n |\widehat{L}||\widehat{U}|. \tag{2.17}$$

**Theorem 2.3** (From [113, thm. 8.5])**.** *Let the triangular system $Tx = b$, where $T \in \mathbb{R}^{n \times n}$ is nonsingular, be solved by backward or forward substitution. Then the computed solution $\widehat{x}$*

**Algorithm 2.3** Dense tile LU (left-looking) factorization (without pivoting)

**Input:** a $p \times p$ block matrix $A$ of order $n$.
**Output:** its computed LU factors.

**for** $k = 1 : p$ **do**
  UPDATE:
    $A_{k,k} \leftarrow A_{k,k} - \sum_{j=1}^{k-1} L_{k,j} U_{j,k}$.
    **for** $i = k+1$ **to** $p$ **do**
      $A_{i,k} \leftarrow A_{i,k} - \sum_{j=1}^{k-1} L_{i,j} U_{j,k}$ and $A_{k,i} \leftarrow A_{k,i} - \sum_{j=1}^{k-1} L_{k,j} U_{j,i}$.
    **end for**
  FACTOR:
    Compute the LU factorization $L_{k,k} U_{k,k} = A_{k,k}$.
  SOLVE:
    **for** $i = k+1 : p$ **do**
      Solve $L_{i,k} U_{k,k} = A_{i,k}$ for $L_{i,k}$ and $L_{k,k} U_{k,i} = A_{k,i}$ for $U_{k,i}$.
    **end for**
**end for**



(a) Update. (b) Factor. (c) Solve.

Figure 2.2: Dense tile LU (left-looking) factorization.

*satisfies*

$$(T + \Delta T)\widehat{x} = b, \qquad |\Delta T| \le \gamma_n |T|. \tag{2.18}$$

**Theorem 2.4** (From [113, thm. 9.4]). *Let $A \in \mathbb{R}^{n\times n}$ and suppose GE applied in precision $u$ produces computed LU factors $\widehat{L}$, $\widehat{U}$, and a computed solution $\widehat{x}$ to $Ax = b$. Then*

$$(A + \Delta A)\widehat{x} = b, \qquad |\Delta A| \le \gamma_{3n} |\widehat{L}||\widehat{U}|. \tag{2.19}$$

What follows from (2.17) and (2.19) is that the size of the error is determined by $|\widehat{L}||\widehat{U}|$ rather than $|A|$, and unfortunately, the ratio $\||L||U|\|/\|A\|$ can be arbitrarily large. This is why GE is not guaranteed to be stable. This instability can be quantified by a term that is called the *growth factor* and which is defined classically as

$$\rho_n = \frac{\max_{i,j,k} |a_{i,j}^{(k)}|}{\max_{i,j} |a_{i,j}|}. \tag{2.20}$$

We can now express the error on the computed solution in terms of $A$ and the growth factor.

**Theorem 2.5** (From [113, lem. 9.6]). *If $A = LU \in \mathbb{R}^{n\times n}$ is an LU factorization produced by GE applied in precision $u$ without pivoting then*

$$|||L||U|||_\infty \leq (1+2(n^2-n)\rho_n)\|A\|_\infty. \tag{2.21}$$

**Theorem 2.6** (From Wilkinson, written in [113, thm. 9.5]). *Let $A \in \mathbb{R}^{n\times n}$ and suppose GE applied in precision $u$ produces a computed solution $\widehat{x}$ to $Ax = b$. Then*

$$(A+\Delta A)\widehat{x} = b, \qquad \|\Delta A\|_\infty \leq n^2\gamma_{3n}\rho_n\|A\|_\infty. \tag{2.22}$$

In order to make GE more stable and to avoid division by zero in (2.12) and (2.13) that would break the whole computation, it is almost mandatory to adopt a pivoting strategy during the factorization. The philosophy of numerical pivoting in GE is to permute the entries in $A$ such that we generate small entries $|a_{i,j}^{(k+1)}|$ in (2.12). It would limit the growth factor and reduce the error generated on the computed solution. A standard strategy to do so is *partial pivoting* which, at the $k$th step of GE, interchanges the $k$th row with the $r$th row, where $r$ is picked such that the new pivot $|a_{r,k}^{(k)}|$ is the maximum of the pivotal column $(|a_{i,k}^{(k)}|)_{k\leq i\leq n}$, that is,

$$r \equiv \underset{k\leq i\leq n}{\arg\max} |a_{i,k}^{(k)}|. \tag{2.23}$$

We name this variant *Gaussian Elimination with Partial Pivoting (GEPP)*. GEPP guarantees that the entries in $L$ are bounded by 1, meaning that for all $1 \leq j \leq i \leq n$ we have $|\ell_{i,j}| \leq 1$, and that the growth factor is bounded by $\rho_n \leq 2^{n-1}$. It is actually possible to build some classes of matrices where this upper bound on $\rho_n$ for GEPP is attained; however, as stated by Higham [113, sect. 9.4]: "*Despite the existence of matrices for which $\rho_n$ is large with partial pivoting, the growth factor is almost invariably small in practice*". This is why we will consider that the growth factor is of order a constant in our different analyses using partial pivoting.

Using Theorem 2.6, we can bound the forward error of the LU direct solver by

$$\frac{\|x-\widehat{x}\|_\infty}{\|x\|_\infty} \leq n^2\gamma_{3n}\rho_n\kappa_\infty(A), \tag{2.24}$$

where we can observe a dependence on the condition number of $A$. Thus, the more $A$ is ill-conditioned, the more the computed solution $\widehat{x}$ obtained by GEPP can differ from the true solution $x$. The backward error does not depend on $\kappa(A)$ and, from (2.7), is always smaller than the forward error.

*2.2.1.3 $LDL^T$ and $LL^T$ solvers.* For the symmetric case, we factorize the matrix $A$ as $LDL^T$, where $L$ is a unit lower triangular matrix and $D$ is diagonal. In addition, when $A$ is positive definite, we can use an even simpler decomposition $LL^T$ referred to as the Cholesky factorization, where $L$ is not unit anymore. Exploiting the symmetry by using $\text{LDL}^\text{T}$ or $\text{LL}^\text{T}$ solver allows storing half the number of entries for the factors and doing half the number of flops during the factorization in comparison to the LU solver.

We will always use the LU factorization for the rounding error analyses in this manuscript.

However, the results can be straightforwardly extended to $\mathrm{LDL}^{\mathrm{T}}$ or $\mathrm{LL}^{\mathrm{T}}$ factorization where we have equivalence for Theorems 2.2 and 2.4 to 2.6 in Higham [113, chap. 10 and 11].

### 2.2.2 Least squares problem and QR solver

*2.2.2.1 Least squares with QR factorization.* Even though computing the solution of the least squares (LS) problem is not the main focus of this manuscript, we will still discuss it because the methods we present for the solution of square linear systems can be naturally extended for this use case.

Let us consider the *least squares problem*

$$r = \min_x \|b - Ax\|_2, \tag{2.25}$$

where $A \in \mathbb{R}^{m\times n}$ $(m \geq n)$ has full rank and $b \in \mathbb{R}^m$. This problem can be seen as the approximation of the solution of an overdetermined system and is used in a wide range of applications from statistical regression, optimal control, or image processing, to cite a few. The solution of the least squares problem (2.25) satisfies the *normal equations*

$$A^T Ax = A^T b, \tag{2.26}$$

which has a unique solution since $A$ is supposed to have full rank. Solving the system (2.26) directly by applying a Cholesky factorization on $A^T A$ presents some numerical issues since the bound (2.24) on the forward error of the computed solution of (2.26) would be a function of $\kappa_2(A^T A) = \kappa_2(A)^2$. Instead, a more stable way to solve (2.26) is to use the QR factorization of $A$

$$A = Q\begin{bmatrix} R \\ 0 \end{bmatrix} \equiv QU, \tag{2.27}$$

where $Q = \begin{bmatrix} Q_1, Q_2 \end{bmatrix} \in \mathbb{R}^{m\times m}$ is an orthogonal matrix with $Q_1 \in \mathbb{R}^{m\times n}$, $Q_2 \in \mathbb{R}^{m\times(m-n)}$, and $R \in \mathbb{R}^{n\times n}$ is upper triangular. The solution can be then expressed as $x = R^{-1}Q_1^T b$.

*2.2.2.2 Stability.* We have the following results on the accuracy of the computed QR factors and the computed LS solution by QR factorization from Higham [113, chap. 19 and 20].

**Theorem 2.7** (From [113, eq. 19.13])**.** *Let $A \in \mathbb{R}^{m\times n}$ $(m \geq n)$ have full rank. The computed $Q$ factor by the Householder QR factorization method in precision $u$ satisfies*

$$\|\widehat{Q} - Q\|_F \leq \sqrt{n}\widetilde{\gamma}_{mn} \tag{2.28}$$

**Theorem 2.8** (From [113, thm. 19.4] (weak form))**.** *If the Householder QR algorithm applied to $A \in \mathbb{R}^{m\times n}$ $(m \geq n)$ in precision $u$ runs to completion then the computed QR factors $\widehat{Q} \in \mathbb{R}^{m\times m}$ and $\widehat{U} \in \mathbb{R}^{m\times n}$ satisfy*

$$\widehat{Q}\widehat{U} = A + \Delta A, \qquad \|\Delta A\|_F \leq \sqrt{n}\widetilde{\gamma}_{mn}\|A\|_F. \tag{2.29}$$

**Theorem 2.9** (From [113, thm. 20.3] (weak form)). *Let $A \in \mathbb{R}^{m\times n}$ ($m \geq n$) have full rank and suppose the LS problem (2.25) is solved using the Householder QR factorization method in precision $u$. The computed solution $\widehat{x}$ is the exact LS solution to*

$$\min_x \|(b+\Delta b)-(A+\Delta A)\widehat{x}\|_2, \qquad \|\Delta A\|_F \leq \widetilde{\gamma}_{mn}\|A\|_F, \qquad \|b\|_2 \leq \widetilde{\gamma}_{mn}\|b\|_2. \tag{2.30}$$

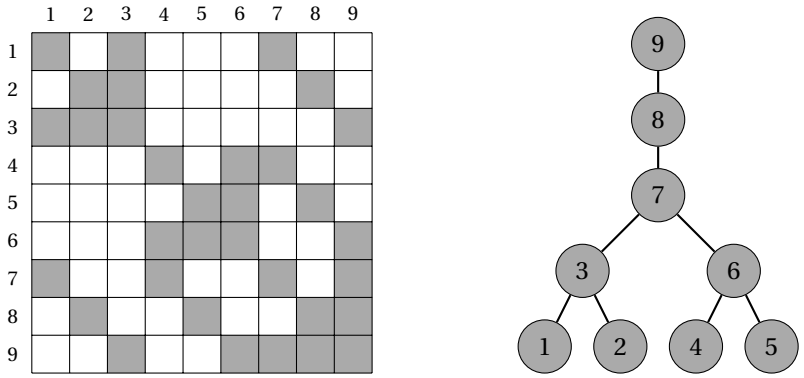
In addition, from Higham [113, sect. 20.2], the accuracy on the computed minimal residual $\widehat{r} = \min_x \|b - A\widehat{x}\|_2$ obtained by QR factorization in working precision $u$ satisfies

$$\|\widehat{r}-r\|_2 \lesssim m\widetilde{\gamma}_{mn}(\||b|+|A||x|\|_2 + \mathrm{cond}_2(A^T)\|r\|_2), \tag{2.31}$$
$$\|\widehat{r}\|_2 \lesssim m\widetilde{\gamma}_{mn}\||b|+|A||x|\|_2 + (1+m\widetilde{\gamma}_{mn}\,\mathrm{cond}_2(A^T))\|r\|_2. \tag{2.32}$$

### 2.2.3 The multifrontal sparse direct solver

Among the different approaches to solve sparse linear systems directly, we will focus in this work on the multifrontal sparse direct solver method (see Schreiber [186], Duff and Reid [69; 70], Liu [149]). Even though most of our conclusions apply to any kind of sparse solver, some results will be specific to the multifrontal approach. We do not cover the method in detail; instead, we provide the main information needed for the proper understanding of this manuscript. More complete descriptions of the method can be found in the books of Duff et al. [72], Davis [54] and in the theses of L'Excellent [140], Mary [158], Buttari [40].

*2.2.3.1 Handling sparsity.* When the matrix $A$ is *sparse*, that is, when there are few interactions between the variables leading to a matrix with mostly zero entries, additional challenges arise for the direct solution of (1.1). Mainly, the algorithms and the data structures need to take advantage of the structural sparsity by performing the computations only on the nonzeros entries carrying the problem's relevant information. In Figure 2.3a, we represent a $9\times 9$ sparse matrix where each row and column is associated with a variable from one to nine and where the gray boxes are the nonzeros entries.




(a) Sparse matrix (b) Elimination tree

Figure 2.3: On the left, a sparse matrix representing a linear system with nine variables. A gray box corresponds to a nonzero entry in the matrix (i.e., an interaction between two variables). On the right, the associated elimination tree.

A key consequence of the structural sparsity of (1.1) is that the elimination of the $k$th variable (2.12) by GE will not necessarily affect all the remaining $(k+1)$th, ..., $n$th rows and columns. For instance, the elimination of variable 1 of matrix 2.3a only affects rows and columns associated with variables 3 and 7 because from (2.12) we need both $a_{i,1} \neq 0$ and $a_{1,j} \neq 0$ for the entry $a_{i,j}^{(2)}$ ($i > 1$ and $j > 1$) to be updated. We call *contributions* of variable $k$ to variable $\ell$ the updates on the $\ell$th row and column that occur during the elimination of variable $k$. Hence, back to our example, we cannot eliminate variable 3 before having computed the contributions of variable 1, but we can eliminate variable 2 because variable 1 does not have any contributions to variable 2.

The dependencies between the elimination of each variable can be expressed in a compact form without redundancy by the *elimination tree* (Schreiber [186]). For example, the elimination tree of matrix 2.3a is represented in Figure 2.3b, where node $k$ corresponds to the elimination of the $k$th variable. The factorization can then be performed by computing all eliminations through a bottom-up traversal of the tree, that is, from leaves to root. The elimination tree is a fundamental structure in sparse direct solvers because it contains the order in which the variables must be eliminated, and it expresses exploitable parallelism by telling which variables can be eliminated concurrently.

Once the factors are computed, the elimination tree can also serve for the solve phase: the forward substitution is computed through a bottom-up traversal of the tree, and the backward substitution is computed through a top-down traversal, that is, from root to leaves.

*2.2.3.2 Fill-in.* The *fill-in* of a sparse factorization is the phenomenon by which the entry $a_{i,j}^{(k)}$ can become a nonzero entry even if $a_{i,j}$ is zero. This occurs when $a_{i,k}^{(k)}$ and $a_{k,j}^{(k)}$ are nonzeros for some $k$ in (2.12). Thus, the factors of $A$ are denser than $A$ itself. As the fill-in can severely increase the computational cost and the memory consumption of the sparse direct solver, reducing it is critical. In Figure 2.4, we display in red the new nonzeros entries that are generated by GE during the factorization of the sparse matrix in Figure 2.3a.

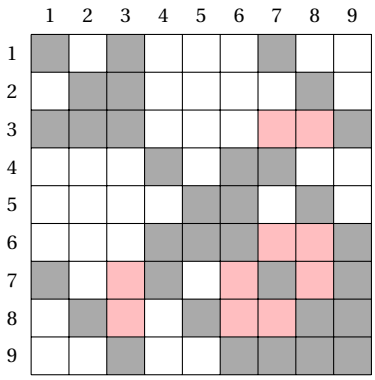



Figure 2.4: Fill-in after factorization of the sparse matrix of Figure 2.3a. A red box corresponds to a new entry in the factors.

The order in which the variables are eliminated in (1.1), or the *ordering*, has a significant

impact on the fill-in. We can change this order by permuting rows and columns in $A$ such that variable 1 might be eliminated after variable 3 (say). Unfortunately, finding the ordering that minimizes the fill-in is an NP-complete problem as demonstrated by Yannakakis [216], and we can only use heuristics to limit it. Some are based on the use of local information on the variables (e.g., Liu [148], Amestoy et al. [11], Rothberg and Eisenstat [179], Ng and Raghavan [163]), or some apply global partitioning strategies on the variables (e.g., George [79]). For the experiments of this manuscript, we use the METIS library (Karypis and Kumar [133]) for the ordering which adopts a hybrid strategy using both global and local heuristics.

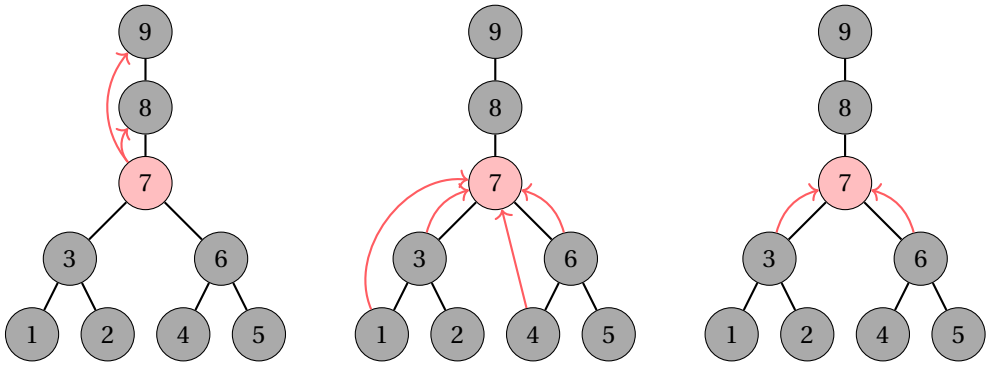
*2.2.3.3 Analysis phase.* To exploit the structural sparsity, sparse direct solvers require an *analysis* phase done prior to the factorization and the solve. This phase has two primary roles. First, it should preprocess the operations to improve the solver's performance. It includes, in particular, computing an ordering to limit the fill-in (see previous section 2.2.3.2). Second, it should perform the symbolic factorization simulating the eliminations to predict the fill-in and determine the elimination tree. Based on this information, the solver can allocate and prepare the different structures necessary for the factorization, distribute the workload, and proceed to other operations meant to prepare and optimize the factorization.

The analysis does not make use of the values of the nonzero entries in $A$ and is essentially composed of symbolic operations. Thus, in our context, it means that this part of the sparse direct solver cannot be accelerated through low precisions, and only the factorization and the solve phases are potentially concerned by mixed precision improvements.

*2.2.3.4 The multifrontal method.* We distinguish three main classes of sparse direct solvers, all scheduling differently the computation and the application of the contributions; we represent them in Figure 2.5.

- In the *right-looking* approach (Figure 2.5a), the contributions of variable 7 to variables 8 and 9 are computed and applied right after the elimination of variable 7; they are computed as soon as possible.
- In the *left-looking* approach (Figure 2.5b), the contributions of variables 1, 3, 4, and 6 to variable 7 are computed and applied when we eliminate variable 7; they are computed as late as possible.
- In the *multifrontal* approach (Figure 2.5c), the contributions of variables 1, 3, 4, and 6 to variable 7 are computed as soon as possible but not applied immediately. Instead, the contribution of variable 1 to variable 7 is computed during its elimination, carried along during the elimination of variable 3, and applied when it is time to eliminate variable 7.

In the multifrontal sparse direct solver, we associate with each node in the elimination tree a dense matrix called a *frontal matrix* (or *front*) on which a partial factorization is performed. It yields both the elimination of the variable and the computation of all its contributions. For example, the partial factorization on node 7 in the tree 2.5c eliminates




(a) Right-looking approach (b) Left-looking approach (c) Multifrontal approach

Figure 2.5: Right-looking, left-looking, and multifrontal approaches. The red arrows represent the different contributions sent when variable 7 is eliminated.

variable 7 by computing the associated row and column in the factors L and U and computes all its contributions to variables 8 and 9 in the form of a Schur complement.

Variables with columns having the same structures can be grouped together in the same frontal matrix of the elimination tree and can be eliminated at once. This process is called *amalgamation*; it further transforms the elimination tree in an *assembly tree* illustrated by Figure 2.6. The sparse factorization of the multifrontal solver is therefore expressed as a sequence of efficient BLAS 3 partial dense factorizations whose dependencies are described by the tree structure. In Figure 2.6, the red parts are the computed factors entries, and the green parts are the temporary contribution blocks carried over the tree. This temporary data is called the *active memory*. Due to this active memory, the peak memory consumption of the multifrontal factorization is higher than the factors in memory; we call the difference between the two the *active memory overhead*.

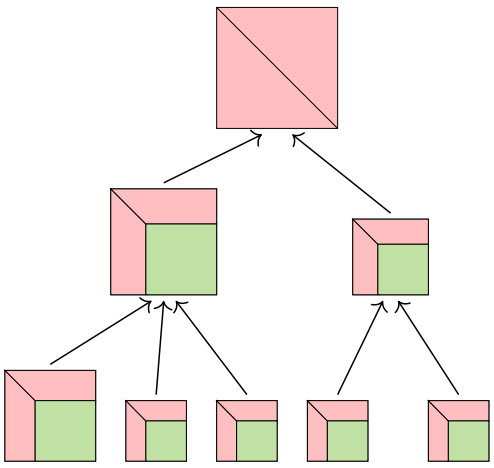


Figure 2.6: An example of an assembly tree of a multifrontal factorization.

The right- and left-looking approaches (Figures 2.5a and 2.5b) are generally referred to as *supernodal* methods. Even though this manuscript is focused on the multifrontal

method, most of the conclusions related to its improvement with mixed precision apply to supernodal methods as well. However, there will be one significant difference related to memory consumption. Because the supernodal methods do not need to carry the contributions over the tree, they do not generate as much temporary data as the multifrontal method and, specifically, do not have active memory. Necessarily, any improvement related to reducing the active memory consumption will not concern supernodal methods.

*2.2.3.5 Operational complexity.* George [79] showed that it is possible to derive complexities for linear systems arising from a discretization on a regular grid when the nested dissection ordering is applied. We call $d$ the dimension of the regular grid and $N$ the number of points on each dimension of the grid (e.g., for a 2D grid $d = 2$ and the total number of points is $n = N^2$, for a 3D grid $d = 3$ and $n = N^3$).

Nested dissection is an ordering where the $l$th level of the elimination tree contains $(2^d)^l$ frontal matrices of dimension $m_l = (N/2^l)^{d-1}$. Since every partial factorization of the fronts is of order $\mathcal{O}(m_l^3)$ flops and the fronts have $\mathcal{O}(m_l^2)$ entries, the total number of flops of the sparse factorization, the solve, and the number of entries in the factors can be determined. We display these different complexities in Table 2.2.

Table 2.2: Theoretical complexities in flops and memory of the multifrontal solver for sparse problems coming from 2D and 3D regular grids.

| | Flops factorization | Flops solve | Entries in $LU$ |
|---|---|---|---|
| Dense | $\mathcal{O}(n^3)$ | $\mathcal{O}(n^2)$ | $\mathcal{O}(n^2)$ |
| 2D | $\mathcal{O}(n^{3/2})$ | $\mathcal{O}(n \log n)$ | $\mathcal{O}(n \log n)$ |
| 3D | $\mathcal{O}(n^2)$ | $\mathcal{O}(n^{4/3})$ | $\mathcal{O}(n^{4/3})$ |

For 3D problems the ratio between the flops complexities of the factorization and the solve operations is $n^{2/3}$, whereas for 2D problems it is $\sqrt{n}/\log n$. A resulting effect of this is that the solve operation is, relative to the factorization, less costly in 3D than in 2D. This difference has some consequences on algorithms that require the application of multiple solves from one factorization; in particular, it concerns the iterative refinement algorithm that we are interested in (see chapter 4).

*2.2.3.6 Exploiting parallelism.* In a multifrontal solver, we have two sources of parallelism. The structure of the assembly tree offers high-level parallelism which lets us process concurrently different partial dense factorizations of the tree at the same time; we call this *tree parallelism*. On the other hand, we can also parallelize the partial dense factorization of a given front; we call this *node parallelism*. We face two issues when exploiting these two kinds of parallelism. Tree parallelism decreases near the root, whereas node parallelism generally increases because frontal matrices tend to be bigger. Conversely, tree parallelism increases near the leaves, and node parallelism becomes less efficient because the fronts tend to be smaller.

In the case of the multifrontal sparse direct solver, increasing the tree parallelism will increase the number of contribution blocks stored concurrently in memory. It can there-

fore increase the active memory overhead and, so, the global memory consumption of the factorization.

*2.2.3.7 Pivoting in sparse solvers.* In the case of unsymmetric or symmetric indefinite problems, pivoting must be used to reduce element growth and make the algorithm backward stable, as stated in section 2.2.1.2. However, pivoting reduces the efficiency and scalability of the factorization on parallel computers for both dense and sparse systems because it requires communication and synchronization. In the case of sparse factorizations, pivoting has the additional drawback of introducing more fill-in. Moreover, because this fill-in depends on the unfolding of the factorization, it cannot be accurately predicted beforehand, so pivoting requires the use of dynamic data structures and may lead to load unbalance in a parallel setting. For this reason, few sparse direct solvers employ robust pivoting techniques such as partial pivoting. Although the overhead imposed by pivoting can be modest in many cases, when targeting large scale parallel computers and/or numerically difficult problems, performance may be severely affected.

This is why more scalable pivoting strategies are often employed in sparse direct solvers that meet different trade-offs between robustness and computation efficiency. For instance, *threshold partial pivoting*, proposed by Duff et al. [72], relaxed the choice of pivot by accepting pivot $a_{r,k}^{(k)}$ slightly lower than the maximum of the pivotal column $(|a_{i,k}^{(k)}|)_{k\leq i\leq n}$. We accept $a_{r,k}^{(k)}$ with $k \leq r \leq n$ if

$$|a_{r,k}^{(k)}| \geq \tau_p \max_{k\leq i\leq n} |a_{i,k}^{(k)}|, \tag{2.33}$$

for a certain threshold $\tau_p \leq 1$. Another example is *static pivoting* proposed by Li and Demmel [142] that we will further review in section 2.2.4.2.

The drawbacks of these techniques are that they cannot generally guarantee the backward stability of the direct solver as for partial pivoting and can worsen the accuracy of the solution.

*2.2.3.8 Software.* For our different experiments with the multifrontal sparse direct solver, we rely on the MUltifrontal Massively Parallel sparse direct Solver (MUMPS) package (Amestoy et al. [12]) for the solution of unsymmetric, symmetric positive definite, or general symmetric linear system of equations, exploiting both MPI and OpenMP parallelization on distributed memory computers.

However, many other sparse direct solvers are available. For example, UMFPACK (Davis [53]) and WSMP (Gupta [98]) are other implementations of the multifrontal method. SuperLU (Li and Demmel [143]) and its variants SuperLU_MT and SuperLU_DIST designed for sequential shared memory and distributed memory systems, PaStiX (Hénon et al. [106]), and PARDISO (Schenk et al. [185]) are implementations of the right- and left-looking methods.

## 2.2.4 Numerical approximations in sparse factorization

In order to improve performance and/or reduce complexity, sparse direct solvers often compute approximate factorizations. By approximate factorization, we refer to techniques

that make a numerical approximation at the algorithm level, independent of any floating-point arithmetic. We are interested in two approximate factorization techniques in this manuscript: BLR and static pivoting.

*2.2.4.1 Block Low-Rank.* In several applications, we can exploit the presence of redundant or unimportant data, referred to as the data sparsity, by partitioning dense matrices (e.g., those appearing in the assembly tree of the multifrontal method, see Figure 2.6) into blocks of low numerical rank. These blocks can be suitably compressed with a reliably controlled loss of accuracy, for example, through a truncated SVD decomposition. Sparse direct solvers exploiting this property to accelerate the computations and reduce memory consumption have been proposed and shown to be highly effective in a variety of applications (Amestoy et al. [15], Ghysels et al. [80], Shantsev et al. [187], Pichon et al. [176], Amestoy et al. [17]).

Figure 2.7 offers an illustrative view of how the compression works. We approximate a given block $B_i \in \mathbb{R}^{b_i \times b_i}$, carrying the interactions between the subdomains $\sigma$ and $\pi$, by a product of rectangular matrices $X_i Y_i^T$ with an accuracy $\tau_b$, where $X_i, Y_i \in \mathbb{R}^{b_i \times k_i}$. We call it a low-rank approximation because $B_i$ is at most of rank $b_i$ and $X_i Y_i^T$ is at most of rank $k_i \leq b_i$. The weaker the interactions between the subdomains the more the compression is efficient. If $k_i \ll b_i$, using $X_i Y_i^T$ instead of $B_i$ reduces flops and memory and can greatly improve the general performance of the solver.

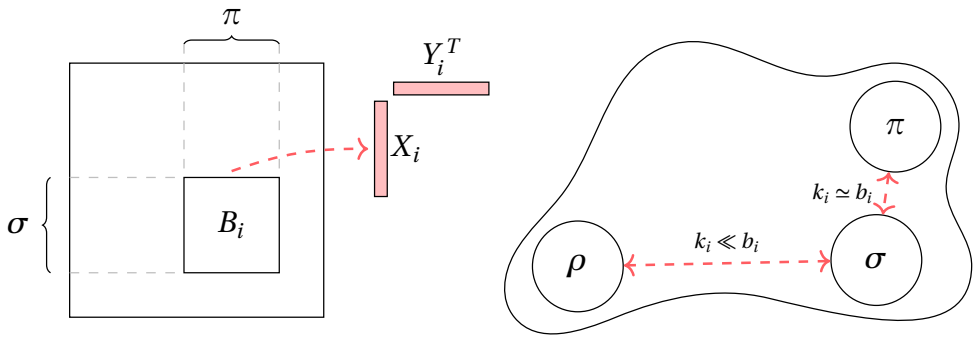



Figure 2.7: Compression by block of a dense matrix. Strong or weak interactions between subdomains lead to a high- or low-rank block, respectively.

The block low-rank (BLR) format (Amestoy et al. [14; 16; 17]) is based on a flat block partitioning of the matrix into low-rank blocks as opposed to other formats that employ hierarchical partitioning. The LU factorization of a BLR matrix can be efficiently computed by adapting the usual tile LU factorization, represented by Algorithm 2.3, to take advantage of the low-rank property of the blocks. For a detailed description of the BLR LU factorization algorithms, we refer to, for example, Amestoy et al. [17] or Mary [158]; in this manuscript, we specifically use the left-looking UCF+LUA variant described by Amestoy et al. [17] and represented by Algorithm 2.4. UCF refers to the order in which the different operations are carried: Update, Compress, and Factor+Solve. The compression of the $k$th row and column is made before their elimination; thus, the update and solve steps benefit from the reduction of the number of operations.

**Algorithm 2.4** BLR LU factorization (without pivoting)

**Input:** a $p \times p$ block matrix $A$.
**Output:** its BLR LU factors $\widetilde{L}$ and $\widetilde{U}$.

**for** $k = 1$ **to** $p$ **do**
  UPDATE:
    $A_{k,k} \leftarrow A_{k,k} - \sum_{j=1}^{k-1} \widetilde{L}_{k,j}\widetilde{U}_{j,k}$.
    **for** $i = k+1$ **to** $p$ **do**
      $A_{i,k} \leftarrow A_{i,k} - \sum_{j=1}^{k-1} \widetilde{L}_{i,j}\widetilde{U}_{j,k}$ and $A_{k,i} \leftarrow A_{k,i} - \sum_{j=1}^{k-1} \widetilde{L}_{k,j}\widetilde{U}_{j,i}$.
    **end for**
  COMPRESS:
    **for** $i = k+1$ **to** $p$ **do**
      Compute LR approximations $\widetilde{A}_{i,k} \approx A_{i,k}$ and $\widetilde{A}_{k,i} \approx A_{k,i}$.
    **end for**
  FACTOR+SOLVE:
    Compute the LU factorization $\widetilde{L}_{k,k}\widetilde{U}_{k,k} = A_{k,k}$.
    **for** $i = k+1$ **to** $p$ **do**
      Solve $\widetilde{L}_{i,k}\widetilde{U}_{k,k} = \widetilde{A}_{i,k}$ for $\widetilde{L}_{i,k}$ and $\widetilde{L}_{k,k}\widetilde{U}_{k,i} = \widetilde{A}_{k,i}$ for $\widetilde{U}_{k,i}$.
    **end for**
**end for**

Once the partial dense BLR LU factorization is finished, the $L$ and $U$ factors are compressed and the contribution block is still full-rank. In the context of the multifrontal solver, it is possible to compress the contribution blocks; however, compressing them does not lead to a global reduction of the flops but instead increases them. It is because, for each partial factorization, we need to decompress the contributions to apply them before the elimination and, at the end of the partial factorization, we need to compress the generated contribution. On the other hand, the communications related to the transfer of the contribution blocks across processes can be reduced and the active memory consumption lowered, leading to a global reduction of the memory consumption during the factorization. The choice of compressing the contribution blocks is therefore made when lower memory consumption is preferred over reduced execution time.

Amestoy et al. [16] determined the operational complexities of the BLR multifrontal method. They showed that the complexities are significantly reduced compared with the ones of the full-rank multifrontal method (see section 2.2.3.5); for example, we can at best decrease the complexity from $\mathcal{O}(n^2)$ flops on a 3D grid to $\mathcal{O}(n^{4/3})$. The rest of the complexities are displayed in Table 2.3. The constants hidden in the big $\mathcal{O}$ depend on the ranks of the blocks. These ranks are determined by a threshold, $\tau_b$ in this manuscript, that controls the accuracy of the approximations. A larger threshold leads to lower memory and operational costs but also lower accuracy.

Note that for simplicity, Algorithm 2.4 describes UCF without numerical pivoting. However, the numerical experiments of this manuscript using the UCF BLR factorization will use pivoting.

Higham and Mary [118] proved the backward stability of the BLR LU factorization. In their work, they provided equivalent results as for Theorems 2.2 to 2.4 for the case of the

Table 2.3: Theoretical complexities in flops and memory of the BLR multifrontal solver for sparse problems coming from 2D and 3D regular grid. The full-rank complexities are recalled for comparison.

| | Flops factorization | Flops solve | Entries in $LU$ |
|---|---|---|---|
| full-rank 2D | $\mathcal{O}(n^{3/2})$ | $\mathcal{O}(n\log n)$ | $\mathcal{O}(n\log n)$ |
| BLR 2D | $\mathcal{O}(n\log n)$ | $\mathcal{O}(n)$ | $\mathcal{O}(n)$ |
| full-rank 3D | $\mathcal{O}(n^2)$ | $\mathcal{O}(n^{4/3})$ | $\mathcal{O}(n^{4/3})$ |
| BLR 3D | $\mathcal{O}(n^{4/3})$ | $\mathcal{O}(n\log n)$ | $\mathcal{O}(n\log n)$ |

BLR LU factorization. We recall their results below and specialize them to the UCF variant.

**Theorem 2.10** (From [118, thms. 4.2 and 4.3])**.** *Let $A\in\mathbb{R}^{n\times n}$ be a nonsingular matrix partitioned into $p^2$ blocks of order $b$. If a UCF algorithm is used and runs to completion, it produces BLR LU factors of $A$ satisfying*

$$A=\widehat{L}\widehat{U}+\Delta A,\quad \|\Delta A\|_F\le(f(p)\tau_b+\gamma_p)\|A\|_F+\gamma_c\|\widehat{L}\|_F\|\widehat{U}\|_F+O(u\tau_b),\tag{2.34}$$

*where $c=b+2r^{3/2}+p$, and where $f(p)$ is a function of $p$.*

**Theorem 2.11** (From [118, thm. 4.4])**.** *Let $\widehat{T}\in\mathbb{R}^{n\times n}$ be a triangular BLR matrix partitioned into $p^2$ LR blocks $\widehat{T}_{ij}\in\mathbb{R}^{b\times b}$ and let $v\in\mathbb{R}^n$. If the solution to the system $\widehat{T}x=v$ is computed by solving the triangular system $T_{ii}x_i=v_i-\sum_{j=1}^{i-1}\widehat{T}_{ij}x_j$ for each block $x_i=x((i-1)b+1:ib)$, the computed solution $\widehat{x}$ satisfies*

$$(\widehat{T}+\Delta\widehat{T})\widehat{x}=v+\Delta v,\quad \|\Delta\widehat{T}\|_F\le\gamma_c\|\widehat{T}\|_F,\quad \|\Delta v\|_2\le\gamma_p\|v\|_2,\tag{2.35}$$

*where $c=b+r^{3/2}+p$.*

**Theorem 2.12** (From [118, thm. 4.5])**.** *Let $\widehat{A}\in\mathbb{R}^{n\times n}$ be a $pb\times pb$ BLR matrix and let $v\in\mathbb{R}^n$. If the linear system $\widehat{A}x=v$ is solved by solving the triangular systems $\widehat{L}y=v$ and $\widehat{U}x=y$, where $\widehat{L}$ and $\widehat{U}$ are the BLR LU factors computed by the UCF algorithm, then the computed solution $\widehat{x}$ satisfies*

$$\begin{aligned}&(A+\Delta A)\widehat{x}=v+\Delta v,\\ &\|\Delta A\|_F\le(f(p)\tau_b+\gamma_p)\|A\|_F+\gamma_{3c}\|\widehat{L}\|_F\|\widehat{U}\|_F+O(u\tau_b),\\ &\|\Delta v\|_2\le\gamma_p(\|v\|_2+\|\widehat{L}\|_F\|\widehat{U}\|_F\|\widehat{x}\|_2)+O(u^2),\end{aligned}\tag{2.36}$$

*where $c=b+2r^{3/2}+p$, and where $f(p)$ is a function of $p$.*

A recent extension of this analysis handling the case of mixed precision inside BLR has been made by Amestoy et al. [10].

*2.2.4.2 Static pivoting.* Unlike partial pivoting (see section 2.2.1.2), static pivoting, first proposed by Li and Demmel [142], does not apply permutations on the rows or columns of the sparse matrix. Instead, when a pivot is found to be too small with respect to a prescribed

threshold $\tau_s\|A\|_\infty$ (i.e., if at the $k$th step $|a_{k,k}^{(k)}| < \tau_s\|A\|_\infty$), it is replaced with $\tau_s\|A\|_\infty$. However, with such a strategy, depending on the chosen parameter $\tau_s$, the growth factor can be far more important and consequently static pivoting guarantees lower numerical stability. Actually, static pivoting has a twofold effect on the accuracy and stability of the factorization. A small value for $\tau_s$ introduces a smaller error but might lead to a large growth factor, while a large value controls element growth but reduces the accuracy of the factorization.

However, what is lost in stability is traded for substantial computational improvements. Indeed, static pivoting improves the use of BLAS 3 operations and improves parallelism compared with partial pivoting, whose scalability suffers from the communications needed to identify the pivots at each elimination stage. Moreover, the use of static pivoting in a sparse direct solver keeps the original ordering and does not introduce additional fill-in as partial pivoting does (see the last paragraph of section 2.2.3.2). Consequently, static pivoting is less prone to load unbalance, additional flops, and increased memory consumption.

## 2.3 Iterative solvers

An *iterative solver* for the solution of the general square linear system (1.1) is a solver that computes a sequence $x_1, x_2, \ldots, x_q$ converging toward the true solution $x$. Compared with direct solvers, iterative solvers generally have a significantly lower computational cost and memory consumption if the convergence is quick or the iterations are stopped quickly. For example, considering a sparse problem coming from the discretization on a 3D regular grid, an iterative solver would generally do about $\mathcal{O}(n)$ flops and stores $\mathcal{O}(n)$ entries (if the number of iterations is far lower than the dimension of the problem $n$ and without considering preconditioner) compared with $\mathcal{O}(n^2)$ flops and $\mathcal{O}(n^{4/3})$ entries for the multifrontal solver (see section 2.2.3.5). In particular, their better scalability made them very popular for solving high-dimensional problems too big to be processed with direct solvers. However, their efficiency is very sensitive to the number of iterations which depends strongly on the internal parameters of the solver and the matrix properties (e.g., the spectrum). Therefore, the resource consumption of iterative solvers can be quite unpredictable and inconsistent across problems; in addition, many well-known iterative solver are not guaranteed to be backward stable.

In this manuscript, we will mainly focus on the GMRES iterative solver that we will extensively combine with direct solvers and other various preconditioners in a mixed precision fashion. Therefore, we first introduce the GMRES algorithm in section 2.3.1 and its MGS-GMRES variant on which we based our rounding error analyses. Then, in section 2.3.2, we will explain what a preconditioner is and present a few of them that we will use or discuss later on in the manuscript. Finally, we will quickly review other iterative solvers for the solution of (1.1) in section 2.3.3.

### 2.3.1 GMRES

The *GMRES* algorithm proposed by Saad and Schultz [183] for the solution of general square linear systems is a Krylov subspace based iterative solver that computes at each iteration

$k$ better and better approximations to the true solution $x$.

*2.3.1.1 The GMRES algorithm.* We consider the *Krylov subspaces* that are of the form

$$K_k(A, v) = \text{span}\{v, Av, A^2 v, \ldots, A^{k-1} v\}, \tag{2.37}$$

where $A \in \mathbb{R}^{n\times n}$ and $v \in \mathbb{R}^n$, and we consider $x_0$ and $r_0$, respectively, a first guess of the solution of (1.1) and its associated residual. The GMRES algorithm for the solution of general unsymmetric square linear systems consists in minimizing the residual norm of (1.1) over all vectors in $x_0 + K_k(A, r_0/\|r_0\|_2)$. For the sake of readability, we denote from now on $K_k \equiv K_k(A, r_0/\|r_0\|_2)$ if there is no ambiguity. Note that for $k \equiv n$, $K_k$ spans $\mathbb{R}^n$ and GMRES would deliver the exact solution $x$ in exact arithmetic.

To achieve this, GMRES uses the *Arnoldi* algorithm proposed by Arnoldi [28] to iteratively build orthonormal bases for increasing subspaces $K_j$ at a given iteration $j$. The procedure is as follows. For $j = 1$, we choose $v_1 = r_0/\|r_0\|_2$ where we verify naturally $K_1 = \text{span}\{v_1\}$. To build the next vector $v_{j+1}$ at step $j+1$, we multiply the previous Arnoldi vector $v_j$ by $A$, and we use a Gram-Schmidt process to orthonormalize the resulting vector against the already computed previous vectors $v_1, \ldots, v_j$. It can be shown that $K_{j+1} = \text{span}\{v_1, \ldots, v_{j+1}\}$.

**Algorithm 2.5** Arnoldi with MGS

**Input:** a matrix $A \in \mathbb{R}^{n\times n}$, a first vector $v_1 \in \mathbb{R}^n$, and a number of iteration $k$.
**Output:** a basis $V_k \in \mathbb{R}^{n\times k}$ for $K_k$ and a Hessenberg matrix $\bar{H}_k \in \mathbb{R}^{(k+1)\times k}$.

```
v_1 = v_1/||v_1||
for j = 1 : k do
    w_j = A v_j
    for l = 1 : j do
        h_{l,j} = v_l^T w_j
        w_j = w_j − h_{l,j} v_l
    end for
    h_{j+1,j} = ||w_j||
    v_{j+1} = w_j / h_{j+1,j}
end for
V_k = [v_1, ..., v_k],   H̄_k = {h_{i,j}}_{1≤i≤k+1; 1≤j≤k}
```

A variant of the Arnoldi algorithm using Modified Gram-Schmidt orthogonalization is described in Algorithm 2.5. The completion of the algorithm delivers a basis $V_k \in \mathbb{R}^{n\times k}$ and a Hessenberg matrix $\bar{H}_k \in \mathbb{R}^{(k+1)\times k}$ which verify by construction

$$AV_k = V_{k+1}\bar{H}_k. \tag{2.38}$$

Since for any vector $x$ in $x_0 + K_k$ it exists $y \in \mathbb{R}^n$ such that $x = x_0 + V_k y$, we have

$$b - Ax = b - A(x_0 + V_k y) = \beta v_1 - V_{k+1}\bar{H}_k y = V_{k+1}(\beta e_1 - \bar{H}_k y), \tag{2.39}$$

where $\beta = \|r_0\|_2$ and $e_1 = [1, 0, \ldots, 0]^T$. In addition, since $V_{k+1}$ is orthonormal we have $\|b -$

$Ax\|_2 = \|\beta e_1 - \bar{H}_k y\|_2$.

**Algorithm 2.6** MGS-GMRES

**Input:** an $n \times n$ matrix $A$ a right-hand side $b$, and a number of iteration $k$.
**Output:** a computed solution to $Ax = b$.

1: Initialize $x_0$
2: $r_0 = Ax_0 - b$
3: $\beta = \|r_0\|, \quad v_1 = r_0/\beta$
4: **for** $j = 1 : k$ **do**
5: $\quad w_j = Av_j$
6: $\quad$ **for** $l = 1 : j$ **do**
7: $\quad\quad h_{l,j} = v_l^T w_j$
8: $\quad\quad w_j = w_j - h_{l,j} v_l$
9: $\quad$ **end for**
10: $\quad h_{j+1,j} = \|w_j\|$
11: $\quad v_{j+1} = w_j / h_{j+1,j}$
12: **end for**
13: $y_k = \arg\min_y \|\beta e_1 - H_k y\|$
14: $x_k = x + V_k y_k$

Following the Arnoldi process, GMRES computes its approximation $x_k$ of the solution of (1.1) as the unique vector of $x_0 + K_k$ which minimizes the norm of the residual, that is

$$x_k = x_0 + V_k y_k, \quad \text{where} \tag{2.40a}$$
$$y_k = \arg\min_y \|\beta e_1 - \bar{H}_k y\|_2. \tag{2.40b}$$

We described in Algorithm 2.6 the *Modified Gram-Schmidt Generalized Minimal RESidual (MGS-GMRES)* variant of GMRES using Modified Gram-Schmidt orthonormalization. If $k \ll n$, the computation of the minimum (2.40b) is relatively inexpensive to compute; therefore, the most costly parts of Algorithm 2.6 are the matrix–vector product kernel with $A$ (step 5) and the orthogonalization process (steps 6 to 9).

*2.3.1.2 Orthonormalization and stability.* There are three popular ways to orthonormalize the vectors of the basis in the Arnoldi process that are often preferred over *Classical Gram-Schmidt (CGS)*: either the *Modified Gram-Schmidt (MGS)* algorithm, the *Householder* algorithm, or the *Classical Gram-Schmidt with reorthogonalization (CGS2)* algorithm, which consists in reapplying the orthonormalization after a first application of CGS. The reason for this is that CGS is very sensitive to the rounding errors; it generates a loss of orthogonality of order $\kappa(A)^2 u$, against $\kappa(A)u$ for MGS and $u$ for Householder and CGS2 (see Giraud et al. [82]). We describe in Algorithm 2.7 both the CGS and MGS algorithms; the difference being that, with MGS, we orthogonalize against the emerging set of vectors instead of the original set as for CGS. The Householder and CGS2 algorithms are the most reliable regarding the quality of the orthogonalization in inexact arithmetic. However, both algorithms are more expensive in terms of flops than the two other approaches. Throughout this manuscript, we will use the MGS-GMRES variant because this is the one among the four which is the

most often used for the solution of large sparse linear systems. However, GMRES can be combined with any of these four orthonormalization algorithms; for instance, it has been first combined with the Householder algorithm by Walker [209].

**Algorithm 2.7** CGS (left) and MGS (right)

**Input:** a set of vectors $\{a_j\}_{1\leq j\leq n}$.
**Output:** the orthonormalized set of vectors $\{q_j\}_{1\leq j\leq n}$ from $\{a_j\}_{1\leq j\leq n}$.

1: **for** $j = 1 : n$ **do**
2: $a_j^{(1)} = a_j$
3: **for** $l = 1 : j-1$ **do**
4: $a_j^{(1)} = a_j^{(1)} - (a_j, q_l)q_l$
5: **end for**
6: $q_j = a_j^{(1)} / \|a_j^{(1)}\|$
7: **end for**

1: **for** $j = 1 : n$ **do**
2: $a_j^{(1)} = a_j$
3: **for** $l = 1 : j-1$ **do**
4: $a_j^{(1)} = a_j^{(1)} - (a_j^{(1)}, q_l)q_l$
5: **end for**
6: $q_j = a_j^{(1)} / \|a_j^{(1)}\|$
7: **end for**

GMRES has been proven backward stable with both the Householder and the MGS orthonormalization. It means that there exists an iteration $k \leq n$ such that GMRES applied in working precision $u$ delivers an approximate solution $x_k$ whose backward error is of order the unit roundoff $u$. The proof for the Householder orthonormalization was made by Drkošová et al. [68], and the one with MGS orthonormalization, which is sensibly more difficult due to the higher sensitivity of MGS to rounding errors, was proposed a decade later by Paige et al. [175].

*2.3.1.3 Restarted GMRES.* When the number of iteration $k$ is large, GMRES becomes impracticable since it needs to store $\mathcal{O}(kn)$ entries and since step 13 of Algorithm 2.6 requires the solution of a dense minimization problem of dimension $k$. The issue is even more amplified for very high-dimensional problems where only a few dense vectors can be stored, so we are restricted to just a few iterations. To remedy this issue, we can use a restarting strategy that allows GMRES to cumulate more iterations while bounding the practical cost by keeping the basis to a small acceptable dimension.

**Algorithm 2.8** Restarted GMRES

**Input:** an $n \times n$ matrix $A$, a right-hand side $b$, and maximum number of iterations $k$.
**Output:** a computed solution to $Ax = b$.

1: Initialize $x_0$
2: **repeat**
3: Compute the approximate solution $x_k$ of $Ax = b$ by $k$ iterations of GMRES with the first guess $x_0$
4: $x_0 = x_k$
5: **until** convergence

We describe the so-called *restarted Generalized Minimal RESidual (restarted GMRES)* in Algorithm 2.8. The principle is simple: we start a GMRES for the solution of (1.1) and when we reach the $k$th iteration, we stop the process, get the approximate solution $x_k$, erase the

basis, and restart GMRES with the first guess initialized to $x_0 = x_k$; we repeat the process until we are satisfied with the approximate solution. Note that restarted GMRES with a maximum fixed number of iterations $k < n$ per GMRES call, as described in Algorithm 2.8, is not guaranteed backward stable; the solution might stagnate and not converge to a satisfactory accuracy.

### 2.3.2 Preconditioners

GMRES might converge too slowly for a wide range of real-life and industrial applications; therefore, it would be unusable as it is on these problems. This is why we often combine GMRES (and Krylov subspace based iterative solvers in general) with a preconditioner aiming to remedy this issue. A preconditioner transforms the original linear system into an easier one to solve, possessing the same solution. Preconditioning is a fundamental component of iterative solvers. Actually, the robustness and the computing performance often depend more on the quality of the preconditioner than on the chosen iterative solver itself and the setup of its internal parameters.

*2.3.2.1 Basics.* A preconditioner $M$ of the matrix $A$ is a matrix that approximates $A$ and should be both relatively inexpensive to compute and apply. We can use $M$ to define a preconditioned system that shall be solved instead of the original one (1.1) in three main ways:

- The *left preconditioning* approach where the system becomes

$$M^{-1}Ax = M^{-1}b, \tag{2.41}$$

  and which consists in applying the preconditioner to the "left".

- The *right preconditioning* approach where the system becomes

$$AM^{-1}u = b, \qquad x = M^{-1}u, \tag{2.42}$$

  and which consists in applying the preconditioner to the "right".

- The *split preconditioning* approach where the system becomes

$$M = M_1M_2, \qquad M_1^{-1}AM_2^{-1}u = M_1^{-1}b, \qquad x = M_2^{-1}u, \tag{2.43}$$

  and which consists in a mix between left and right preconditioning where the preconditioner is split in two; the first part is applied to the left and the second part is applied to the right.

If $M$ is a good preconditioner, the resulting preconditioned matrix $\widetilde{A} = M^{-1}A$ or $AM^{-1}$ or $M_1^{-1}AM_2^{-1}$ has a lower condition number than the original matrix $A$, reducing potentially the number of iterations. We describe in Algorithm 2.9 left- and right-preconditioned MGS-GMRES. Note that step 1 is "Optional" in the sense that not every preconditioner requires

some kind of precomputation. For instance, this step is necessary for preconditioners based on some (approximate) factorization of the matrix $A$, such as the preconditioners we will review in the following sections 2.3.2.2 to 2.3.2.4.

**Algorithm 2.9** Left- (left) and right- (right) preconditioned MGS-GMRES

**Input:** an $n \times n$ matrix $A$ and a preconditioner $M$, a right-hand side $b$, and a number of iteration $k$.

**Output:** a computed solution to $Ax = b$.

| | |
|---|---|
| 1: Compute $M$ (Optional) | 1: Compute $M$ (Optional) |
| 2: Initialize $x_0$ | 2: Initialize $x_0$ |
| 3: $\boldsymbol{r_0 = M^{-1}(Ax_0 - b)}$ | 3: $r_0 = Ax_0 - b$ |
| 4: $\beta = \|r_0\|, \quad v_1 = r_0/\beta$ | 4: $\beta = \|r_0\|, \quad v_1 = r_0/\beta$ |
| 5: **for** $j = 1 : m$ **do** | 5: **for** $j = 1 : m$ **do** |
| 6: $\boldsymbol{w_j = M^{-1}Av_j}$ | 6: $\boldsymbol{w_j = AM^{-1}v_j}$ |
| 7: **for** $l = 1 : j$ **do** | 7: **for** $l = 1 : j$ **do** |
| 8: $h_{l,j} = v_l^T w_j$ | 8: $h_{l,j} = v_l^T w_j$ |
| 9: $w_j = w_j - h_{l,j} v_l$ | 9: $w_j = w_j - h_{l,j} v_l$ |
| 10: **end for** | 10: **end for** |
| 11: $h_{j+1,j} = \|w_j\|$ | 11: $h_{j+1,j} = \|w_j\|$ |
| 12: $v_{j+1} = w_j / h_{j+1,j}$ | 12: $v_{j+1} = w_j / h_{j+1,j}$ |
| 13: **end for** | 13: **end for** |
| 14: $y_m = \arg\min_y \|\beta e_1 - H_m y\|$ | 14: $y_m = \arg\min_y \|\beta e_1 - H_m y\|$ |
| 15: $x_m = x + V_m y_m$ | 15: $\boldsymbol{x_m = x + M^{-1}V_m y_m}$ |

The preconditioned problem does not have to be explicitly formed; rather, solving the preconditioned system (2.41), (2.42), or (2.43) instead of the original one (1.1) requires one application of $M^{-1}$ to a vector at each Arnoldi iteration (step 6 of Algorithm 2.9). While the preconditioner should reduce the number of iterations of GMRES, if its application is not sufficiently cheap, we might not obtain any computational improvements. On the other hand, in general, the better the preconditioner is, that is, the more it reduces the number of iterations, the more it is demanding in terms of computing resources. This is why finding the ideal preconditioner that maximizes computing performance is difficult.

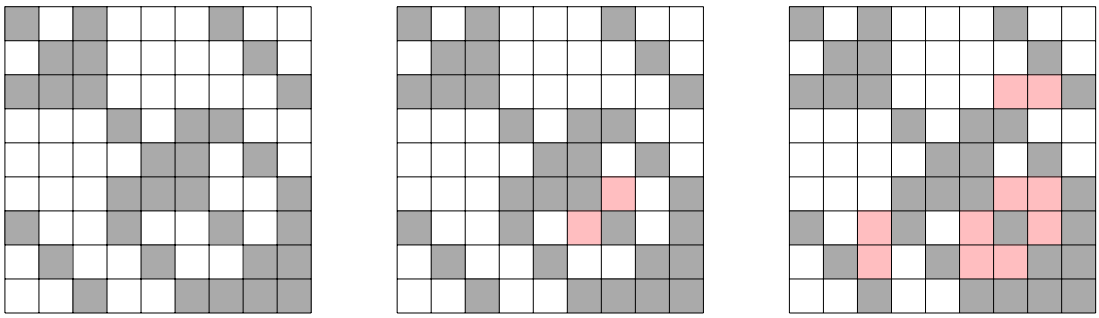
Note that the right-preconditioned GMRES can be formulated in a "flexible" way to accommodate nonconstant preconditioners, we call this algorithm *Flexible Generalized Minimal RESidual (FGMRES)* (Saad [182]). This version stores an additional set of $j$ vectors at each iteration $j$.

*2.3.2.2 LU preconditioner.* The LU (or $\mathrm{LDL}^{\mathrm{T}}/\mathrm{LL}^{\mathrm{T}}$) preconditioner consists simply in choosing $M = LU$ where $L$ and $U$ are the LU factors of $A$. The factors should be computed once before the start of the GMRES iterations; the application of $M^{-1}$ at each iteration is an LU solve.

It might seem strange at first to be willing to use a full direct solver as a preconditioner. Indeed, with GEPP, GMRES would instantly provide a satisfactory solution in one iteration; that is, after one LU solve application. However, it becomes interesting to use an LU direct

solver as a preconditioner when the LU solver does not offer satisfactory accuracy on the solution. It can happen, for example, when the LU solver uses an unstable pivoting strategy such as static pivoting (this setting has been explored by Arioli et al. [27]) or when low precision is used for the factorization (examples of this setting can be found in the papers of Arioli and Duff [25] and Carson and Higham [45]). In these cases, the GMRES iterations can enhance the accuracy of the solution of the LU solver. In addition, if the number of iterations to improve the solution is small and the computational performance improvements obtained with the inaccurate direct solver are substantial, we might be able to solve (1.1) with less computational resources and as accurately as a stable LU direct solver.

*2.3.2.3 ILU preconditioner.* The Incomplete LU (ILU) preconditioner for the solution of sparse linear systems consists of performing the LU factorization of $A$ where some entries of the sparse factors are dropped following a certain strategy. The underlying philosophy of ILU is to contain and limit the fill-in during the factorization, see section 2.2.3.2. The advantage of doing so is that the complexity of the factorization and the number of entries in the factors can be reduced and, therefore, ILU can improve time and memory performance compared with a full standard LU factorization. Figure 2.8 illustrates the different number of entries in $A$ and its ILU or full LU factors.



(a) A (b) ILU (c) LU

Figure 2.8: Entries in $A$ and in the ILU and full LU factors.

Of course, dropping entries during the factorization is not numerically harmless, and the resulting performance gains come at the cost of inaccurate LU factors. Intuitively, it can be expected that the more entries we drop, the worse the preconditioner is. However, the more we drop, the more the resources needed for the computation and application of the preconditioner are reduced. Hence, a good setting is a trade-off that depends on the problem.

A prevalent choice of ILU preconditioner is the zero fill-in ILU or ILU(0). As its name suggests, it consists in avoiding fill-in by dropping any nonzero entries in the LU factors that are not nonzero in $A$. Hence, the ILU(0) factors have the same number of nonzero entries as $A$, the effect of the fill-in is canceled, and the time and memory consumption in computing and applying the factors are drastically cheaper than that of a full sparse LU factorization. However, it often leads to an approximation that is too crude and might require too many iterations to converge.

The ILU preconditioner that we will use in this manuscript is Incomplete LU with threshold (ILUT). With ILUT, we drop the entries of the factors that are lower than a certain prescribed threshold $\tau_{\text{ilu}}$. This strategy comes from the intuition that small elements carry less information than larger ones, so keeping the large values will produce a better quality preconditioner than dropping entries regardless of their magnitude. Dropping elements by their magnitude and not by position allows, in particular, better control of the numerics. The downside is that there is no control on the level of fill-in, which can be arbitrarily large.

*2.3.2.4 Block Jacobi preconditioner.* The block Jacobi preconditioner consists in choosing the preconditioner $M$ as a block diagonal part of $A$ as illustrated in Figure 2.9. Depending on the application, the block can be chosen of the same size or not and can overlap or not. For the preconditioner to be well-defined, we need each block (i.e., $D_i$ for $i = 1, \cdots, 4$ in the figure) to be nonsingular.

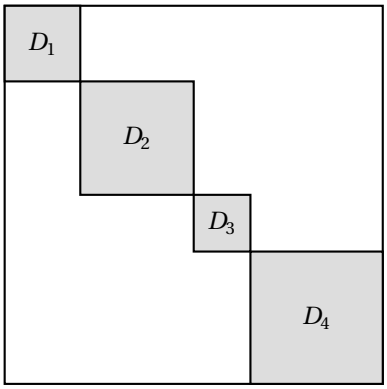



Figure 2.9: A block diagonal decomposition of a matrix.

The application of $M^{-1}$ is therefore reduced to the successive applications of $D_i^{-1}$ for $i = 1, \cdots, 4$. $D_i^{-1}$ is generally computed by mean of a factorization or by explicitely forming the inverse. If the blocks are relatively small, the computation and application of $M^{-1}$ should be relatively cheap; for instance, far cheaper than computing and applying the full factors of $A$.

### 2.3.3 Other iterative solvers

This manuscript mainly focuses on GMRES because it has backward stable forms and can compute solutions of general square linear systems. However, we believe that many of our ideas developed for GMRES can be efficiently applied to other Krylov subspace based iterative solvers.

Other solvers for the solution of symmetric linear systems are, for example, MINimal RESidual (MINRES) by Paige and Saunders [173] or Conjugate Gradient (CG) by Hestenes and Stiefel [107] (using the Lanczos process instead of the Arnoldi one).

The BiConjugate Gradient (BiCG) method is a generalization of CG for unsymmetric matrices (Fletcher [75], Sonneveld [194], van der Vorst [207]). Full Orthogonalization Method (FOM) by Saad [181] can also be used for the solution of unsymmetric square systems.

Finally, LSQR proposed by Paige and Saunders [174] is a method for the solution of rectangular systems.

All these methods can make use of preconditioners and present different trade-offs. The best choice of method depends strongly on the application.

# 3 Iterative refinement history

Iterative refinement is an old and altogether simple algorithm. It has been in use for more than 70 years and has been the object of many theoretical and practical analyses, making it one of the most well-understood and predictable linear algebra algorithms. Though, it is fascinating how decades later, we are still interested and able to produce new knowledge on it. The key reason for this observation is that iterative refinement has constantly been evolving over time, repeatedly reconsidered according to trends, researcher's interests, and hardware specifications, as well as the computing challenges of each computing era. In this chapter, we are interested in providing a brief historical survey on iterative refinement, answering both questions: *how was it used at some time* and *why ?* This chapter is not interested in providing technical details on the method which is the topic of chapter 4.

This chapter is also dedicated to providing a list of the research around iterative refinement, to put these different pieces of work in relation, and to clarify their respective contributions. Therefore, we hope that it can be used as a survey and might interest anybody looking for an overview of this specific algorithm.

## 3.1 Newton's method (17th century)

The roots of iterative refinement date back to the 17th century with the emergence of Newton's method, from which it is a specialization. *Newton's method* consists in iteratively building approximations $x_i \in \mathbb{R}^n$ converging, under suitable conditions (e.g., good starting point), toward a zero $x$ of a given continuously differentiable function $f : \mathbb{R}^n \to \mathbb{R}^n$ :

$$x_{i+1} = x_i - (\nabla f(x_i))^{-1} f(x_i), \tag{3.1}$$

where $\nabla f$ is the Jacobian matrix of $f$. We can build correction methods from Newton's method that express a problem under the form of a residual $f(x)$ that we want to minimize.

In doing so, we can "refine" a first guess of the solution $x_0$ to progressively obtain a better solution, that is, a solution that provides a smaller residual for the problem.

The *iterative refinement* procedure is the application of Newton's method to the function $f(x) = b - Ax$ and, thus, it is aiming at reducing the residual $b - Ax$ of (1.1). The Newton iterative procedure becomes in this case

$$x_{i+1} = x_i + A^{-1}(b - Ax_i), \tag{3.2}$$

and gives the solution $x = A^{-1}b$ in one iteration in the absence of computing errors, for instance if we use exact arithmetic, no matter the initial guess $x_0$.

In (3.2) three steps can be identified:

$$\text{(1) Computing the residual:} \qquad r_i = b - Ax_i, \tag{3.3}$$

$$\text{(2) Solving the correction equation:} \qquad Ad_i = r_i, \tag{3.4}$$

$$\text{(3) Updating the solution:} \qquad x_{i+1} = x_i + d_i, \tag{3.5}$$

where each of these steps are subject to rounding errors if they are computed on a computer in inexact arithmetic. These rounding errors prevent iterative refinement from converging in one iteration.

## 3.2 From the 40s to the 70s

The primary official appearance of iterative refinement is attributed to Wilkinson, who programmed it on the Automatic Computing Engine (ACE) in the 40s during his collaboration with Turing (1948, p. 111). It was part of a broader study reporting the use of linear algebra methods with the Pilot ACE and was probably one of the first programs to be written for it. It is argued that in earlier work, we could already observe embryos of what will be iterative refinement, such as in the work of Mallock [1933], who seems to present an electrical circuit implementing it. However, it is actually unclear how old is this algorithm because it may have been used with earlier computing units or with paper-and-pencil computing.

The inexact representations of numbers on computers and the resulting rounding errors they generate have been an early concern. It gave birth to modern numerical analysis as a domain of mathematics; in particular, the article of von Neumann and Goldstine [1947] is considered one of the first in this field. In addition, it also raised the interest in algorithms able to recover this lost accuracy, such as, naturally, iterative refinement. When looking more precisely at the context of the use of iterative refinement in this period, we need to consider three essential facts.

First, a fair amount of bits were already put into the numbers, allowing for reasonably accurate computations. For example, the ACE on which Wilkinson experimented with iterative refinement could already process single or double precision floating-point numbers (see Copeland [2005, p. 95]). In addition, the IBM 700's computers series (commercialized in the 50s) were using 36 bits single precision; 72 bits double precision format came later on the 7094 model (see [Wikipedia]). Generally, the only two formats proposed on computers

of this generation were single or double precision; however, for the same arithmetic name (e.g., single or double precision), there were possibly significant differences in the number of bits between computers. This situation was overcome by the establishment of the IEEE 754 standard a few decades later in 1985 (see IEEE Computer Society [2019] for the last revision of the standard).

Second, the use of direct methods with stable pivoting strategies for the solution of linear systems (e.g., partial pivoting for GE, see section 2.2.1.2) was ubiquitous. In addition, because the factors of the matrix can be computed once and applied multiple times throughout the iterations, the refinement procedure can stay relatively cheap, making iterative refinement particularly suited to improve direct methods. For instance, the studies documented in the ACE's progress report (Wilkinson [1948]) and another contemporary study by Fox et al. [1948], as well as all the other studies cited later in this section, use stable direct methods. We did not find any documented use of iterative refinement for other kinds of solvers during this period.

Third, almost all computers of this era were computing the exact scalar product in precision $u^2$ of two numbers in precision $u$. To take even more advantage of this feature, most computers supported the accumulation of the inner products' unitary products in precision $u^2$; this technique is referred to as the "inner-product accumulation" and is described by von Neumann and Goldstine [1947, sect. 2.3]. As said by Wilkinson [1963, p. 6], this strategy was actually a costless access to higher precision $u^2$ for certain kind of computations:

> *"We may say that a double-precision result is obtained although the time taken by the computation is essentially that which we associate with single-precision work"* (1963).

A direct consequence of this is that it was affordable to accumulate a matrix–vector product in precision $u^2$ and, so, the computation of the residual (3.3) of the refinement process could be done more accurately than the other operations. In other words, there could be up to two different precisions used in iterative refinement at that time. It shows that the interest in mixing precisions is not particularly recent and that the first mixed precision algorithm is almost as old as the first computers.

For these different reasons, the primary usage of iterative refinement was to improve normwise backward stable direct solvers run in reasonably accurate arithmetics by computing the residual (3.3) with extra-precision. The use of extra-precision on the residual is interesting because, if we take the example of GEPP, it has been remarked that it removes the dependence of the forward error on the condition number (see bound (2.24)); the forward error of the solution is therefore cheaply improved, reduced to the unit roundoff of the working precision $u$. In addition we can note other secondary use cases for iterative refinement, for instance as evoked by Fox et al. [1948], it has also been used to roughly evaluate the forward error on the solution by looking at the changing digits between $x_{i+1}$ and $x_i$:

> *"The second solution is worth obtaining [...] in those cases when accuracy of a given order is required and there is doubt about the number of correct figures*

> *in the first approximation, of which the residuals do not always give a reliable indication"* (1948).

Moreover, as stated by Golub and Wilkinson [1966], it could be used for performance concerns:

> *"For large systems a single-precision factorization plus iterative refinement has the advantage over double-precision factorization without refinement that it requires less storage [...], is faster"* (1966).

Besides Pilot ACE, iterative refinement has been implemented on other computers of this time such as the ILLIAC (see Snyder [1955]), the IBM 7090/94 with Fortran (see Kahan [1965], Moler [1964]), or the KDF9 with Algol (see [Wikipedia], Martin et al. [1971], Bowdler et al. [1966; 1971]).

It was in the 60s that the first rounding error analyses of iterative refinement were proposed. All the analyses of the period cover direct stable solvers based on LU or QR factorization, respectively, for the solution of a square linear system and least squares problem.

The first rounding error analysis of iterative refinement is also attributed to Wilkinson and appeared in his famous book "*Rounding errors in algebraic processes*" (Wilkinson [1963, p. 121]), which established a standard for rounding error analysts. This analysis is based on a block floating-point representation of the numbers also defined in his book; however, despite the word "floating", block floating-point is closer to fixed-point than floating-point arithmetic. In particular, this analysis reveals a clear connection between the convergence of the forward error and the conditioning of the problem and is, in a way, the first draft of an analysis that will be updated and upgraded over time by many researchers. Indeed, the two main ingredients of the classic iterative refinement analysis are already there: a *convergence condition* guaranteeing the computed forward error of the solution to converge to a certain *limiting accuracy*. In this analysis, the correction equation (3.4) and the update (3.5) are computed in working precision $u$, while the residual (3.3) is computed in extra-precision $u^2$.

The second main iterative refinement analysis is attributed to Moler [1967] which adapted the analysis of Wilkinson to floating-point arithmetic. This analysis is particularly important because floating-point representation is the real number format that has become the most used in scientific computing; therefore, this result is still directly applicable today. Two major add-ons make it particularly interesting. First, the condition for convergence becomes $\kappa(A)u \ll 1$, which is exactly the condition as we know it today, up to constants. Second, this analysis includes different choices for the precision at which the residual (3.3) is computed: computed in working precision $u$ or computed in extra-precision $u^2$. Yet, the case where it is computed in working precision, as for the correction equation and the update, was rarely considered relevant because, as stated by Moler [1967],

> *"In this case, $x_m$ is often no more accurate than $x_1$"* (1967).

It means that the solution at the $m$th iteration would not be much improved compared with the first iterate. Specifically, the forward error remains dependent on the conditioning of the problem without the use of extra-precision. For this reason, the recommended

configuration was to use extra-precision on the residual. Interestingly, the fixed precision approach, using the same precision for all the refinement procedures (3.3)–(3.5), will be reconsidered a decade later while the interest and the ability to use extra-precision will fade; more is said about this in section 3.3.

In the 60s, Golub and Björck both worked on the application of iterative refinement for the improvement of the computed solution of the LS problem (2.25). At first, Golub applied the iterative refinement process directly to the overdetermined system to improve the computed solution $\hat{x}$ obtained through the QR decomposition of $A$ with Householder transformations (see Golub [1965]). This approach has also been used by Bauer [1971], and an Algol implementation was proposed by Businger and Golub [1965]. However, as pointed out by Golub and Wilkinson [1966], the iterative refinement process might not work well for overdetermined systems. Namely, if the system is not consistent or nearly consistent, the residual will not converge to working precision, and the forward error can be arbitrarily large; this is something that we further explain in section 4.5.2.

To overcome this issue, Björck [1967] proposed to apply the iterative refinement to the augmented system whose solution reveals the one of the least squares problem. As the augmented system is a square linear system, iterative refinement is guaranteed to converge regardless of the overdetermination. An Algol implementation was proposed by Björck and Golub [1968]. Later, Fletcher [1975], Björck [1978] discussed the use of the Cholesky factorization combined with iterative refinement for solving the least squares problem. They remarked that while this approach is less robust regarding the conditioning of the problem in comparison with the Householder decomposition, it has a lower flops complexity.

Good surveys of the period about iterative refinement applied on linear systems and least squares problems can also be found in the books of Golub [1969] and Stewart [1973]. In particular, the analysis drawn in the book of Stewart [1973] has been reworked and quoted in many subsequent analyses.

## 3.3 From the 70s to the 2000s

For a long time, iterative refinement has only been used in fixed-precision, meaning that all the operations (3.3)–(3.5) were computed in working precision $u$. Even today, this is still a widely used form of iterative refinement. However, at first sight, it is not obvious what motivated the change from using extra-precision $u^2$ for the residual computation, that we covered in the previous section 3.2, to simply applying it in precision $u$ as for all the other operations. As previously stated, downgrading the precision on the residual from $u^2$ to $u$ could only provide, at best, a small improvement on the forward error of a solution obtained with a stable solver (e.g., GEPP). Mainly, it is not able to remove the dependency of the forward error on the conditioning of the problem (see bound (2.24) on the forward error of GEPP). These reasons even led the community to consider for a while that iterative refinement was not worth doing with no extra-precision on the residual computation.

At the end of the 70s, two articles played an essential role in the reconsideration of fixed precision iterative refinement. They highlighted two of its major benefits.

The first benefit of fixed precision iterative refinement, as shown by Jankowski and

Woźniakowski [1977], is that it guarantees normwise backward stability of any linear solver, direct or iterative, more or less stable, under some conditions. Such a result appeared in a period where propositions for solving linear systems diversified, iterative methods were already on the rise, and direct methods were adapting to parallel computing and sparse data structures. Many of these new methods were not normwise backward stable, and the ability of iterative refinement to recover this stability is both theoretically and practically very appealing and useful. In their article Jankowski and Woźniakowski [1977] used the Chebyshev iteration to solve the correction equation (3.4), showing by this way that iterative refinement is not constrained to be used only with direct methods and can be implemented on top of any linear solver. To the best of our knowledge, this is the first documented use that combines iterative refinement with an iterative solver. They recall this technique in Jankowski and Woźniakowski [1985] for the accurate solutions of elliptic problems in single precision arithmetic. Higham [1988; 1990] used iterative refinement with some fast direct methods for the solutions of Vandermonde-like systems. Iterative refinement has also been extensively used to correct instability arising from sparse direct solver techniques. For example, it has been combined with more scalable but less stable pivoting strategies (see Li and Demmel [1998], Dongarra et al. [2001]) or simply without numerical pivoting at all (see Gill et al. [1996]), and with non-zeros drop strategies (see Arioli et al. [1989], Zlatev [1982]) that aim to reduce fill-in. Other applications of iterative refinement are worth mentioning, such as its use in block elimination methods (see Govaerts and Pryce [1990], Govaerts [2000]) or in the accurate computing of matrix eigenpairs (see Dongarra et al. [1983]).

The second benefit of fixed precision iterative refinement is that it guarantees componentwise backward stability of every linear solver under some conditions. In particular, it is well-known that GEPP is a normwise backward stable solver, but it is not componentwise backward stable; that is, the computed solution $\widehat{x}$ may not exactly satisfy a linear system with each coefficient slightly different from those of the original one. A scaling solution was proposed to make GEPP stable in that sense (see Skeel [1979]), but it is not very helpful in practice because it requires estimates of the solution components $x$. However, when combined with iterative refinement, GEPP becomes componentwise backward stable after one iteration as demonstrated by Skeel [1980] and is, consequently, stable in a stronger sense. In practice, the improvement provided by fixed precision iterative refinement on top of GEPP is generally not essential for most applications. In more detail, it transforms the dependence of the forward error on $\kappa(A)$ (see bound (2.24)) to dependence on $\text{cond}(A,x)$. However, even though $\text{cond}(A,x)$ can be arbitrarily smaller than $\kappa(A)$, we know that the two quantities are close with proper scaling of $A$. While it might have a reduced interest in practice, the theoretical guarantee of componentwise stability is a very appealing property that can serve other rounding error analyses. For example, this result is crucial for the demonstration of the stability of the sparse GEPP to guarantee that we are solving a perturbed system carrying the same sparse structure as the original one; it is explained by Arioli et al. [1989]:

> *"When solving systems of $n$ linear equations $Ax = b$ by means of Gaussian elimination with pivoting, a classical analysis shows that we should expect to get the exact solution $\hat{x}$ of a slightly different linear system $(A+\Delta A)\hat{x} = b+\Delta b$ [...].*

> *This classical view permits any entry of $\Delta A$ or $\Delta b$ to be equally large, and in particular $A + \Delta A$ may be dense even if $A$ is quite sparse."* (1989).

Following these two results, Higham worked on their compilation to provide the most general analysis of iterative refinement at that time. It was done in two stages. At first, Higham [1991] proved that fixed precision iterative refinement applied on an arbitrary solver is stable in the componentwise sense under some conditions, merging both the analysis of Skeel [1980] specialized to GEPP and of Jankowski and Woźniakowski [1977] specialized on the normwise stability. In addition, he extended his results to the least squares case following the idea of Björck [1967] by applying iterative refinement on the augmented system (4.29). In a second time, Higham [1997] extended his analysis to allow the residual (3.3) to be computed in extra-precision, taking special care in considering lapack implementations and making sure his analysis covered them.

Fixed precision iterative refinement has also been considered in fault-tolerant computing by Boley et al. [1995]. They considered LU or QR factorization for the solution of (1.1) followed by one step of fixed precision iterative refinement. If the solution does not satisfy its expected accuracy after the correction, it can be concluded that a transient error occurred in the hardware. On the other hand, if it does satisfy the expected accuracy, it means that either there were no errors or some errors of modest impact were silently corrected. In that sense, we can consider the method tolerant to the error.

Other than the work of Arioli et al. [1989], which is more theory focused, the sparse case has also been tackled by Zlatev [1982] in a more performance-oriented way. In particular, his work highlighted one main asset of sparse iterative refinement compared with the dense case: as the original matrix $A$ has far fewer nonzeros than its factors due to fill-in (see section 2.2.3.2), the memory overhead for storing a copy of $A$ in order to compute the residual is low. In addition, he showed that using drop tolerance strategies with fixed precision iterative refinement can even lead to memory savings compared with a standard direct solver.

As we have explained previously, the rising interest of fixed precision iterative refinement from the 70s can be partly explained by the use of more unstable solvers, where the increasing dimension of the problems led computer scientists to trade stability off for scalability. However, the loss of extra-precison on the computation of the residual (3.3) is mostly due to hardware evolutions. Mainly, the inner-product accumulation capability, allowing cheap extra-precision computation, progressively disappeared from the mid-60s. Thus, using extra-precision for the residual became more or less complicated, slow, or even impossible, depending on the computers. This inconsistency of the inner-product accumulation support between machines made the application of the residual in extra-precision less common, as stated by Golub and van Loan [1996]:

> *"The primary drawback of mixed precision iterative improvement is that its implementation is somewhat machine-dependent. This discourages its use in software that is intended for wide distribution."* (1996).

Fixed precision iterative refinement was and is still widely used in the linear algebra landscape. Most of the books of this period that focused on linear algebra and direct solvers

for solving linear systems mention it at least. Probably the most exhaustive is the book of Higham [2002, p. 231], but dedicated sections and chapters about fixed precision iterative refinement can also be found in the following books by Demmel [1997, p. 60], Golub and van Loan [1996, p. 126], and Duff et al. [2017, p. 80]. Also, the most used direct solvers and linear algebra libraries often embed fixed precision iterative refinement. For the solution of a dense linear system, lapack ([Netlib]) implements such routines (routines whose names end in `-rfs` and called by the expert drivers whose names end in `-svx`). For the sparse case, for instance, the sparse direct solvers MUMPS (Amestoy et al. [2001]), PaStiX (Hénon et al. [1999]), and SuperLU (Demmel et al. [1999]) embed fixed precision iterative refinement to improve the accuracy of the solution (see Amestoy et al. [2001]).

Even though most of the studies and applications of iterative refinement between the 70s and the 2000s were focused on fixed precision, some work took different approaches by exploring the use of multiple precisions in the computations. The following paragraphs present these efforts.

Kiełbasiński [1981] proposed to increase gradually, at each iteration, the number of bits in the mantissa of the precision with which the residual (3.3) and the update (3.5) are computed. This gradual increase should be done smartly in order to optimize the theoretical number of bit operations to reach a given targeted accuracy on the solution. Smoktunowicz and Sokolnicka [1984; 1990] amended this previous work and adapted it to the least squares problem (2.25). We call such approaches "dynamic precision" strategies because the precisions at which the operations (3.3)–(3.5) are computed can change from one iteration to another. Of course, implementing it as described would require some variable floating-point arithmetics, and considering that there were generally only two arithmetics available on computers at this time (single or double precision), such a dynamic precision method was not relevant in practice. Interestingly, the paper of Kiełbasiński [1981] is the first documented study we know about that computes the correction equation with lower precision than the update. Actually, this setting of precisions, where the correction equation is computed less accurately to reduce resource consumption, began to raise much attention only in the mid-2000s. The two following paragraphs will also present pioneer studies using this setting.

The contribution of Douglas et al. [1990] is quite unique as well in the landscape of iterative refinement. Indeed, they tackled the iterative refinement algorithm from an electrical engineering standpoint where they used a fast but inaccurate analog linear solver for the computation for the correction equation (3.4) and recovered a good accuracy by processing the residual (3.3) and the update (3.5) with slower digital circuitry. Namely, they are in a configuration where the correction equation is computed with lower precision than the working precision. Hence, the hybrid analog/digital circuit is faster than the full digital circuit for computing the solution of (1.1).

The work of Turner and Walker [1992], similarly to Jankowski and Woźniakowski [1977], employed a form of iterative refinement using an iterative solver to solve the correction equation (3.4). In their case, they used GMRES, and similarly to Douglas et al. [1990], they applied this iterative solver in single precision while the residual (3.3) and the update (3.5) are processed in double precision. This method can provide a solution with double precision

accuracy while performing the iterative solver in single precision (say), and can potentially accelerate the computation.

The paper by Gulliksson [1994] proposed to extend the analysis of Björck [1967] to the constrained and weighted linear least squares case. Congruently to Björck [1967] and the other studies on the least squares case of the 60s, he worked with extra-precision on the residual (3.3).

## 3.4 From the 2000s to the 2010s

While, before the 2000s, iterative refinement was primarily used to improve the accuracy and stability of linear solvers, after the 2000s, it was increasingly considered as a way to improve the computational performance reliably. The core idea is to use an inaccurate but less resource-demanding linear solver deliberately; this solver would provide a poor solution for the original linear system (1.1) that can be refined cheaply to the targeted accuracy with iterative refinement. This is achieved by using the low precision linear solver for the solution of the correction equation (3.4) while computing the residual (3.3) and the update (3.5) in working precision.

Prior to the 2000s, the workload was essentially on the processing unit and not on the memory subsystem leading to little advantages of low precision data motion. Adding the fact that on many processing units, single precision computations were not faster than double precision ones, low precision arithmetic could not offer acceleration. However, in the early 2000s, two main hardware trends changed how we performed linear algebra computations: the widening gap between the speed of processors and the speed of memories and the widespread availability of short vector SIMD processing units. Because Single Instruction Multiple Data (SIMD) vector units, which started appearing in the mid-1990s, can accommodate twice as many single precision values as double precision ones, the speed of compute-bound algorithms can be potentially doubled by using single precision arithmetic. Identically, memory-bound algorithms can also be accelerated by a potential factor of two because twice as many single precision values can be transferred simultaneously from the memory as double precision ones. However, as explained by Buttari et al. [2007]:

> *"Although short vector, SIMD processors have been around for over a decade, the concept of using those extensions to utilize the advantages of single precision performance in scientific computing did not come to fruition until recently, due to the fact that most scientific computing problems require double precision accuracy"* (2007).

Since most scientific problems require a double precision accuracy solution, low precision arithmetics cannot be used alone. Thus, the observation that iterative refinement can recover a reasonable accuracy from a low precision solver opened new potential applications and stimulated the research interests for this algorithm.

It is a set of papers from Jack Dongarra's team in Tennessee in the mid-2000s that participated extensively in the reconsideration of iterative refinement as a way to improve the resource consumption of a linear solver. In Langou et al. [2006], Buttari et al. [2007], the

authors proposed both: a readaptation of the normwise analysis of iterative refinement with the LU direct solver applied in low precision and performance studies on different architectures (e.g., Intel Pentium III, Intel Pentium IV, AMD Opteron, Cray X1, IBM SP Power3, and others). These performance studies successfully highlighted time gains for the solution of dense linear systems when using single precision factorization instead of double, getting closer to a factor of 2 on different hardware configurations. Some specific hardware, such as the Cell processor, could even lead to a factor of 10. Buttari et al. [2008] proposed to extend this approach to sparse linear systems. In addition to a performance study of iterative refinement applied to the sparse case, they proposed a larger discussion on the acceleration of iterative methods with the preconditioner applied with lower precision. Dongarra's team finally summarized their results in the survey by Baboulin et al. [2009].

Actually, the work by Langou et al. [2006] is not precisely the first to implement iterative refinement over a low precision LU direct solver. Indeed, a previous paper of Geddes and Zheng [2003] already assessed this configuration, but with some slight differences. First, the LU factors are computed in double precision, and the residual and update are processed in software-emulated high precision arithmetic. Second, they did not provide a rounding error analysis.

The previously presented efforts mainly addressed implementations of iterative refinement over direct solvers; however, iterative methods can also be efficiently combined with iterative refinement. For example, in the work of Strzodka and Göddeke [2006;], the authors used a low precision CG to solve the correction equation (3.4). One can note that this is strictly the approach of Turner and Walker [1992] reviewed in the previous section 3.3, the difference being that the solver is not GMRES in this case. Few other papers adopted and studied this setting (see Anzt et al. [2012; 2011]). Moreover, it can be quite natural to compare iterative methods to iterative refinement, mainly when the preconditioner is based on a direct method. For example, Arioli et al. [2007] compared iterative refinement and GMRES/FGMRES over a perturbed static pivoting factorization, and Arioli and Duff [2008], Hogg and Scott [2010] compared iterative refinement and FGMRES using single precision LU factorization while providing double accuracy solution.

Multigrid solvers are another class of methods dedicated to solving linear systems arising from PDEs. They were first introduced in the 1970s and have grown since then. Their ability to efficiently target high-dimensional problems made them very popular from the 1990s; therefore, it does not come as a surprise that these methods have also been efficiently combined with iterative refinement. Equivalently to what has been done in the previous studies using GMRES and CG, it is about applying the multigrid solver in lower precision for the solution of (3.4) while the refinement operations (3.3) and (3.5) are operated in higher. Göddeke et al. [2007] were the first to address such an algorithm and actually compared it with CG-based iterative refinement. They completed this study in Goddeke and Strzodka [2011]. Sumiyoshi et al. [2014], Glimberg et al. [2012] also adopted this approach.

All the previous contributions that we just presented are almost always meant to take advantage of the time and memory saving of single precision over double precision that most of the CPU of that time exhibited. As explained earlier, single precision could be twice as fast as double precision. On some architecture, the single precision speedup could

even be of order a factor 10, such as on the Cell processor, making iterative refinement approaches even more relevant (see Kurzak and Dongarra [2007]). In addition, at that exact moment, other kinds of processor units were introduced into the scientific computing landscape:

- The Field-Programmable Gate Array (FPGA) architectures can, contrary to microprocessors, operate efficiently with arbitrary-width number formats, making this technology particularly relevant for mixed precision algorithms. Implementations of LU-based iterative refinement on these units were done by Sun et al. [2008], Lee and Peterson [2011], and a CG-based implementation was proposed by Strzodka and Göddeke [2006].

- Graphics Processing Unit (GPU) are many-core devices designed for highly parallelizable computations. They were invented to target computer graphics and image processing but became a more generalized computing device. Nowadays, computing linear algebra problems with GPUs seems natural and widespread; however, back in the 2000s, its use in scientific computing was more at the experimental stage. Interestingly, iterative refinement has been quickly considered and combined with GPU computation (e.g. Göddeke et al. [2005], Goddeke and Strzodka [2011], Sumiyoshi et al. [2014], Glimberg et al. [2012]). One natural reason is that for some time GPUs were only supporting single precision and no access to higher double precision was possible. When double precision was finally available, it was more than twice slower than single precision.

Parallel implementations of iterative refinement using low precision LU factorization are available in the PLASMA library Agullo et al. [2009]; for example, in the routine `dsgesv` for single precision LU factorization followed by double precision refinement.

Meanwhile, during the same period, the old approach using extra-precision computation for the residual (3.3), which has been progressively avoided from the 70s (see section 3.3), is being reintroduced. The main problem was the inconsistency in hardware support for extra-precision, which complicated the portability of any implementation. Interestingly, from the 2000s, reliable software solutions for access to extra-precision were proposed. It unlocked the possibility of computing the residual in higher precision than the working precision $u$.

In 2002, following the BLAS Technical Forum meetings, which began in November 1995 at the University of Tennessee, a new BLAS standard was established (see Dongarra [2002]). Among other things, it addressed the extended and mixed precision version of the BLAS routines in Dongarra [2002, chap. 4]. With XBLAS, Li et al. [2002] proposed an implementation of this standard, allowing computation of the residual in extra-precision in a portable and efficient way. As most of the hardware does not support extra-precision natively (e.g., fp128 quadruple precision), they used the double-double arithmetic (see section 2.1.2), which is a good trade-off between high accuracy and performance. There are other examples of extra-precision implementations, such as the Quad-Precision Math Library GNU (2010) or the Intel Fortran Compiler that both support fp128. As stated by Higham [2002, p. 241]:

> *"However, with the release of the Extended and Mixed Precision BLAS (see §27.1O) and the portable reference implementation for IEEE arithmetic, and with the ubiquity of IEEE arithmetic, portable mixed precision iterative refinement is now achievable"* (2002).

These software developments made the use of extra-precision on the residual (3.3) possible again when the working precision $u$ is already the most precise arithmetic supported by the hardware, generally fp64.

Demmel et al. [2006] revisited extra-precision iterative refinement to develop a set of practical stopping criteria for this algorithm; they are described in section 4.6. These criteria aim to quickly detect the convergence of the solution to its limiting accuracy to ensure both: that we stop when the refinement has converged and that we make as few iterations as possible to detect the convergence. Later, Demmel et al. [2009] extended this work to the least squares problems (2.25) by using the augmented system formulation of Björck [1967]; it presents some new challenges compared with the square linear system case.

The availability of extra-precision with XBLAS also renewed the interest in using mixed precision in Newton's method, where the residual (3.3) computed in extra-precision can bring improved accuracy. Tisseur [2001] developed a general rounding error analysis for Newton's method, which, consequently, also applies to iterative refinement since it is a specialization of Newton's method for the solution of linear systems (see section 3.1). The analysis of iterative refinement being well-known, it then serves as a check to verify if the more general analysis is consistent.

Implementations of iterative refinement using extra-precision on the residual are available in the NAG library Ltd [2005]; for example, in the routines `f01abf`, `f04abf`, `f04aef`, `f04amf` and others.

## 3.5 From the 2010s to 2022

The fastly growing presence of very low precisions since the mid-10s offered new opportunities for scientific computing and boosted the research interests over mixed precision algorithms. Most of our supercomputers can now use more than four different precisions; for example, half, single, double, and quadruple precision. Because of this, in recent years, much effort has been made to upgrade iterative refinement into a more versatile algorithm regarding the choice of precision for computing its operations (3.3)–(3.5). These efforts came to fruition under a new form of iterative refinement that allows setting independently the precisions at which each operation (i.e., (3.3), (3.4), or (3.5)) is computed and, therefore, bridges the three main previously evoked uses together: extra-precision for the computation of the residual (3.3) to further improve the forward error accuracy, the same fixed precision for all the operations (3.3)–(3.5) which can bring stability for unstable solvers, and low precision for the solution of the correction equation (3.4) which improves the computational performance.

During the 2010s, much work has been done on the hardware to accommodate efficient half precision arithmetics (i.e., fp16, bfloat16, tfloat32 in Table 2.1), which consume significantly less storage, data movement, and energy, and allow more flops per clock. More details

are given on these low precisions in sections 2.1.2 and 2.1.3. Naturally, with such impressive benefits, much attention has been directed towards leveraging these arithmetics in scientific computing applications. Moreover, the recent announcement of the fp8 arithmetic keeps fueling the need for research in this area.

The interest of clearly separating the precision inside iterative refinement in three distinct precisions, one for each operation (3.3)–(3.5), stems from the emergence of these new arithmetics. Because, first, accounting for quadruple precision in software, our current supercomputers can generally accommodate at least four different precisions. In such a heterogenous arithmetic ecosystem, we need to shape our algorithms toward a highly-mixed-precision scheme to optimize the hardware performance with the application requirements as much as possible. It is the main reason for the current formulation of iterative refinement, which allows for a more versatile set of precisions and is, therefore, more suited for the current needs of applications running on such heterogeneous architectures. Second, while still many scientific applications can handle single precision accuracy, a full half precision solution accuracy is, in most cases, not enough; therefore, it is essential to have a strategy to recover the accuracy when working with these precisions. For these two reasons, the interest over iterative refinement skyrocketed in the late 2010s.

It is the work of Carson and Higham [2017; 2018] in the late 2010s that greatly participated for this revision of iterative refinement.

At first Carson and Higham [2017] revisited the analysis that Higham proposed in the 90s, which was at this time the most general analysis (it can be found in Higham [1997] and in Higham [2002, chap. 12]). This older analysis requires at least $\kappa(A) \leq u$ for the convergence to be guaranteed, regardless of the solver used for the correction equation (3.4). With changes in the solver assumptions and the introduction of a new quantity better capturing the sharpness of an inequality, Carson and Higham [2017] proposed a new convergence condition that can potentially allow $\kappa(A) > u$. In particular, they showed that using iterative refinement with a left-preconditioned GMRES by the LU factors instead of a more classical LU solver carries a less restrictive convergence condition than $\kappa(A) < u$. In this way, this approach can significantly improve the robustness with respect to the condition number of the problem.

A year after, Carson and Higham [2018] addressed the use of low precision for the computation of the correction equation solution (3.4). This last work can be seen somehow as the combination of the analyses of Higham [1997] and Langou et al. [2006] accounting for the robustness improvement brought previously in Carson and Higham [2017]. One of the main add-ons is the possibility of solving the correction equation with a low precision solver in addition to using extra-precision for the computation of the residual (3.3). With this scheme, the authors could use a fast low precision solver, using, for example, half precision, while recovering full double precision accuracy on the forward error. They propose two different specializations for the solver, both using a formulation in three precisions: either a GEPP solver, which is the classical/historical choice of solver, or the more robust LU left-preconditioned GMRES strategy, inherited from their previous work. The robustness consideration actually became even more critical in this study because when low precisions are in play, the convergence conditions become more restrictive. Therefore, the

left-preconditioned GMRES approach, which can overcome at some point this restrictive condition, makes even more sense.

Finally, Carson et al. [2020] tackled the least squares case. The method, as most of the iterative refinement algorithms on least squares problems, relies on the Björck [1967] augmented system approach, which transforms the least squares problem into a square linear system. The previous results of Carson and Higham [2018] are then applicable to this square linear system, and the authors derived analyses for specializations with QR direct solver and left-preconditioned GMRES solver providing equivalent results as in Carson and Higham [2018]. On the other hand, Higham and Pranesh [2021] also proposed to tackle the LS problem, but, by solving the normal equation (2.26) with the left-preconditioned GMRES approach where the Cholesky factors are used as a preconditioner. They explained that the GMRES solver could be naturally exchanged with CG; while it would break the theoretical convergence guarantees because preconditioned CG is not backward stable, it seems to have little effect in practice.

Besides, a few improvements were proposed for left-preconditioned GMRES-based iterative refinement. First, Carson and Khan [2022] explored the use of the approximate inverse preconditioner instead of using the LU or QR factors. Second, Oktay and Carson [2022] proposed to use a recycling strategy inside the GMRES solver that can reduce the cumulated number of GMRES iterations significantly. Recycling is a well-known approach for Krylov subspace based solvers where we reuse information from previous solves on systems that share the same matrix.

Motivated by these new theoretical findings, different performance analyses assessed the sustainability and the benefits of using these new forms of iterative refinement on different kinds of solvers.

For the solution of dense linear systems, Haidar et al. [2017; 2018] especially targeted half precision factorization through GPU and investigated the LU and left-preconditioned GMRES solvers for iterative refinement, as well as other forms of iterative methods using LU preconditioning. Haidar et al. [2020; 2018] extended these studies by exploiting the GPU tensor cores which can deliver even higher performances; error analyses of LU solver with GPU tensor cores have been carried out by Blanchard et al. [2020], Lopez and Mary [2020]. The different implementations resulting from these efforts were made available in the MAGMA library [Magma]; for example, with the `dhgesv_iteref_gpu` or `dsgesv_iteref_gpu` routines, and the cuSolver library NVIDIA [2019]. The iterative refinement using the LU left-preconditioned GMRES is actually the algorithm used by the new HPL-AI Mixed-Precision Benchmark [HPL-AI], which solves a dense system to double precision accuracy using an fp16 factorization; #1 on this benchmark in 2022 is the Frontier supercomputer (#1 TOP500 top [2022] in 2022, USA).

While performance studies on dense linear systems using the recent forms of iterative refinement were quickly tackled, less content has been produced for the sparse case. It may be because, on many supercomputers, the only way to access efficient half precision is on GPUs. However, the use of these units in sparse solvers is less straightforward and attractive than for the dense case due to the lower granularity of the computation. Zounon et al. [2022] investigated these techniques' potential benefits and pitfalls on popular direct

sparse solvers: MUMPS, PARDISO, and SuperLU. They consider multi-core parallelism, acceleration with single precision factorization, and incomplete factors preconditioners.

The papers of Loe et al. [2021;] proposed performance studies of iterative refinement with GMRES for the solution of large sparse linear systems using GPUs. In the same fashion as Turner and Walker [1992], the GMRES solver is applied in single precision for the solution of the correction equation (3.4), while the residual (3.3) and the update (3.5) are in double. Different kinds of preconditioners are used (block Jacobi and polynomial); the study compares its iterative refinement approach with a restarted GMRES applied first in single precision and then fully switched to double precision. This form of iterative refinement has also been studied by Lindquist et al. [2020; 2022].

The use of GPUs for accelerating multigrid solvers has already been studied in the 2000s. An update of this approach is provided by Oo and Vogel [2020], who employ GPU's half precision instead of single precision as previously done. In this study, the multigrid solver uses either full half precision or progressively increases or decreases the precisions as the grids get coarser. The relevance of this last approach was confirmed by the error analysis of McCormick et al. [2021], Tamstorf et al. [2021], which combines mixed precision multigrid with iterative refinement; they concluded that

> *"[...] V-cycles and FMG are capable of leveraging progressive precision by using increasingly lower precision on levels that are increasingly coarser, and thus decreasingly accurate"* (2020).

Interestingly, the increasing number of computer arithmetics has also renewed the interest in dynamic strategies. In the 80s, the papers of Kiełbasiński [1981], Smoktunowicz and Sokolnicka [1984] proposed to progressively increase the number of bits of the precision at which the residual (3.3) and the update (3.5) are computed. At this time, the hardware could not process arbitrary precisions, and the idea could not be implemented. However, the recent access to more arithmetics in standard supercomputers or the use of FPGAs, which can directly supply for arbitrary arithmetic formats, makes this strategy realizable by now. A direct successor of these studies is the papers of Lee et al. [2018; 2020], who proposed to rely on the number of cancellation bits in the residual to know when we should increase the precision. On the other hand, as proposed by Oktay and Carson [2022], we can also dynamically change the precision at which the correction equation (3.4) is solved, which affects the algorithm's convergence. The idea is to start with the least accurate solver and switch to more accurate ones if we observe that the solution cannot converge.

The general trend over mixed precision techniques in the 2010s raised interest around iterative refinement, the algorithm we mainly focus on in this manuscript. However, research on mixed precision algorithms is going far beyond iterative refinement: Higham and Mary [2022], Abdelfattah et al. [2021] are two more general surveys covering the use of mixed precision in linear algebra.

## 3.6 Summary

We have seen that iterative refinement has been used in four main different ways through the different periods, each of them is achieved by a specific setting of precisions for the computation of the residual (3.3), the correction equation (3.4), and the update (3.5). We summarize below these different settings in chronological order of appearance:

- 1: Get a better forward error with extra-precision on the residual (3.3). Using extra-precision for the residual computation removes the dependence of the forward error of the solution on the condition number. This technique is particularly relevant for applications requiring good accuracy on the forward error and dealing with ill-conditioned problems. It is the first form of iterative refinement. Most of the studies on it are covered in section 3.2.

- 2: Recover the stability with fixed precision. When the factorization is not done with a stable algorithm, such as GEPP, iterative refinement applying all its operations (3.3)–(3.5) in the same precision can recover the backward stability. Note that this property is also valid for extra-precision iterative refinement. Most of the studies on fixed precision are covered in section 3.3.

- 3: Accelerate the solution of linear systems with low precision on the correction equation (3.4). Applying the costly solver for the solution of the correction equation in a lower precision and using the working precision for the computation of the residual and the update allows taking advantage of the low precision benefits while refining the solution to working precision accuracy cheaply. Most studies about using low precision for the correction equation are covered in section 3.4.

- 4: All previous strategies combined. Stability, performance, and high accuracy can all be achieved simultaneously by using extra-precision on the residual (3.3), low precision on the correction equation (3.4), and working precision on the update (3.5). It leads to implementations of iterative refinement using more than three different precisions. Most of the studies for this setting are covered in section 3.5.

We classify in Table 3.1 all the references of this chapter based on the linear solver used on the correction equation and the previously presented iterative refinement setting employed (numbered from 1 to 4). We also use a color code to refer to the contribution period. One reference can appear in different cells because it has contributions in each of these cells. Also, for some particular papers, it is sometimes unclear which cell the contribution belongs to, so we made some arbitrary choices.

Table 3.1: Summary of existing scientific papers about iterative refinement classified by type of combination of precision and by linear solver used. The bold references are the articles related to the contributions of this manuscript. In red are represented the references from the 40s to the 70s (section 3.2), in blue from the 70s to the 00s (section 3.3), in green from the 00s to the 10s (section 3.4), in purple from the 10s to 2022 (section 3.5).

| | 1 | 2 | 3 | 4 | Dynamic |
|---|---|---|---|---|---|
| Main rounding error analyses | Wilkinson [1963]<br>Golub [1965]<br>Moler [1967]<br>Björck [1967]<br>Higham [1997]<br>Carson and Higham [2017] | Jankowski and Woźniakowski [1977]<br>Skeel [1980]<br>Higham [1991]<br>Higham [1997] | Douglas et al. [1990]<br>Buttari et al. [2007] | Carson and Higham [2018]<br>Carson et al. [2020]<br>**Amestoy et al. [2021]**<br>Carson and Khan [2022] | Kiełbasiński [1981] |
| $LU/LDL^T/LL^T$ solvers | Wilkinson [1948]<br>Fox et al. [1948]<br>Snyder [1955]<br>Wilkinson [1963]<br>Moler [1964]<br>Kahan [1965]<br>Bowdler et al. [1966]<br>Golub [1969]<br>Martin et al. [1971]<br>Bowdler et al. [1971]<br>Uhlig [1972]<br>Stewart [1973]<br>Zlatev [1982]<br>Dongarra et al. [1983]<br>Golub and van Loan [1996]<br>Higham [1997]<br>Demmel [1997]<br>Higham [2002]<br>Demmel et al. [2006]<br>Carson and Higham [2017] | Skeel [1980]<br>Higham [1988]<br>Arioli et al. [1989]<br>Higham [1990]<br>Govaerts and Pryce [1990]<br>Higham [1991]<br>Boley et al. [1995]<br>Gill et al. [1996]<br>Demmel [1997]<br>Higham [1997]<br>Li and Demmel [1998]<br>Dongarra et al. [2001]<br>Higham [2002] | Geddes and Zheng [2003]<br>Langou et al. [2006]<br>Buttari et al. [2007]<br>Arioli et al. [2007]<br>Kurzak and Dongarra [2007]<br>Arioli and Duff [2008]<br>Sun et al. [2008]<br>Buttari et al. [2008]<br>Baboulin et al. [2009]<br>Hogg and Scott [2010]<br>Lee and Peterson [2011]<br>Haidar et al. [2017; 2018;; 2020]<br>Zounon et al. [2022] | Carson and Higham [2018]<br>Higham and Pranesh [2021]<br>**Amestoy et al. [2021]**<br>Oktay and Carson [2022]<br>**Amestoy et al. [2022]** | Kiełbasiński [1981]<br>Lee et al. [2018]<br>Lee et al. [2020]<br>Oktay and Carson [2022] |

| | | | | | |
|---|---|---|---|---|---|
| QR solvers | Golub [1965]<br>Businger and Golub [1965]<br>Golub and Wilkinson [1966]<br>Björck [1967]<br>Björck and Golub [1968]<br>Golub [1969]<br>Bauer [1971]<br>Stewart [1973]<br>Fletcher [1975]<br>Gullikkson [1994]<br>Demmel et al. [2009] | Higham [1991]<br>Boley et al. [1995]<br>Higham [2002] | | Carson et al. [2020] | Smoktunowicz and Sokolnicka [1990] |
| Iterative solvers | Carson and Higham [2017] | Jankowski and Woźniakowski [1977]<br>Jankowski and Woźniakowski [1985] | Turner and Walker [1992]<br>Göddeke et al. [2005]<br>Strzodka and Göddeke [2006;]<br>Anzt et al. [2011]<br>Anzt et al. [2012]<br>Haidar et al. [2017; 2018;; 2020]<br>Lindquist et al. [2020]<br>Loe et al. [2021;]<br>Lindquist et al. [2022] | Carson and Higham [2018]<br>Carson et al. [2020]<br>Higham and Pranesh [2021]<br>**Amestoy et al. [2021]**<br>Oktay and Carson [2022]<br>**Amestoy et al. [2022]**<br>Carson and Khan [2022] | Oktay and Carson [2022] |
| Multigrid solvers | | | Göddeke et al. [2007]<br>Goddeke and Strzodka [2011]<br>Glimberg et al. [2012]<br>Sumiyoshi et al. [2014]<br>Oo and Vogel [2020] | McCormick et al. [2021]<br>Tamstorf et al. [2021] | |
| Arbitrary solvers | Higham [1997]<br>Tisseur [2001]<br>Higham [2002]<br>Carson and Higham [2017] | Jankowski and Woźniakowski [1977]<br>Higham [1991]<br>Higham [1997]<br>Higham [2002] | Douglas et al. [1990] | Carson and Higham [2018] | Smoktunowicz and Sokolnicka [1984]<br>Lee et al. [2018] |

# 4 State-of-the-art iterative refinement

Mixed precision iterative refinement is definitely the cornerstone of this Ph.D. thesis. Roughly, most of the work presented in this manuscript will be about improving linear solvers with different iterative refinement flavors. In chapter 3, we developed a complete survey gathering and explaining most of the bibliography on this algorithm. In this chapter, we provide an in-depth technical presentation of the most advanced iterative refinement algorithms, containing all the theoretical tools we will need to develop the results of this manuscript.

To this end, we will present and develop in section 4.2 what we call the generalized form of iterative refinement from which we can derive rounding error analyses for many iterative refinement variants. Subsequently, we will specifically focus on two of these variants, which aim at improving direct solvers, using either an LU solver (LU-IR3) or a GMRES preconditioned by the LU factors (LU-GMRES-IR3) in sections 4.3 and 4.4. Naturally, these variants can be straightforwardly extended to any kind of linear system solution based on a factorization of the matrix $A$: $LDL^T$, $LL^T$, QR, etc. We will also explain how these two variants can be adapted to the solution of the least squares problem through QR factorization in section 4.5, discuss stopping criteria in section 4.6, and summarize the main results of the different error analyses in section 4.8.

## 4.1 On the understanding of refining a linear system

As explained in section 3.1, iterative refinement is the application of Newton's method on a linear system in inexact arithmetic. It consists in repeating the three following steps

$$\text{(1) Computing the residual:} \qquad r_i = b - Ax_i, \tag{4.1}$$

$$\text{(2) Solving the correction equation:} \qquad Ad_i = r_i, \tag{4.2}$$

$$\text{(3) Updating the solution:} \qquad x_{i+1} = x_i + d_i, \tag{4.3}$$

to refine the solution in order to improve its accuracy to some extent. In this section, we aim to illustrate the refinement process and intuitively explain the role of each of these steps.

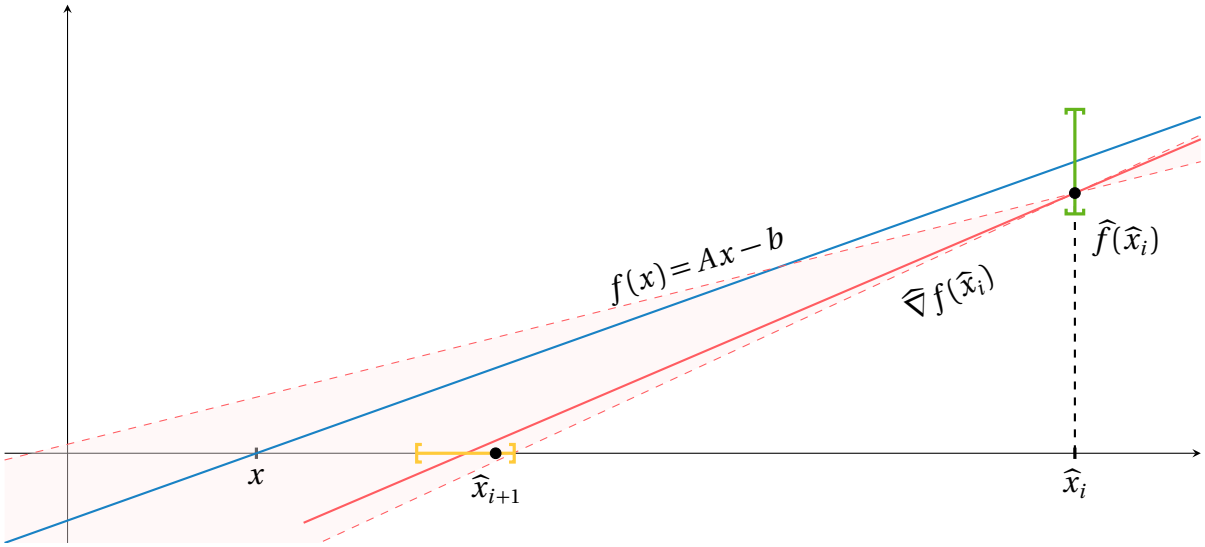



Figure 4.1: The $i$th iteration of Newton's method applied to $f(x) = Ax - b$ with $n = 1$ in the presence of rounding errors. The "green interval" represents the error made on the computation of the residual (step (4.1)), the "red zone" represents the error made on the solution of the correction equation (step (4.2)), and the "yellow interval" represents the error made on the computation of the update (step (4.3)).

Applying Newton's method in inexact arithmetic creates errors that have two types of consequences. To illustrate this point, we use Figure 4.1 which is a geometrical interpretation of Newton's method; it shows how we can build successive $x_i$ converging to the true solution $x$ by using red tangent lines to the blue curve $f(x) = Ax - b$. For the procedure to be representable on a 2D plane, the linear system is of dimension 1 (i.e., $n = 1$), so $A$, $x$, and $b$ are scalars, and the residual is an affine function. In the figure, we have the errors on the computation of the residual and the update (steps (4.1) and (4.3)) represented in green and yellow, and the error on the computation and application of the Jacobian matrix (step (4.2)) represented in red.

The green error comes from the inexact computation of $f(\widehat{x}_i)$. Because we are not able to compute the exact quantity $f(\widehat{x}_i)$, we instead provide a computed quantity $\widehat{f}(\widehat{x}_i)$ contained in the (vertical) green interval, which represents the error range of the residual computation. The yellow error comes from the inexact computation of the new iterate $x_{i+1} = \widehat{x}_i + \widehat{d}_i$. Because of the error we are making in adding the computed correction $\widehat{d}_i$, we instead provide the computed quantity $\widehat{x}_{i+1}$ contained in the (horizontal) yellow interval, which represents the error range for the update computation. The green and yellow errors are not expected to affect much the convergence quality, but they describe the accuracy we can make our successive $x_i$ to converge to. Indeed, the errors on these operations prevent the computed solution from being improved indefinitely by the successive corrections, and the size of these errors shall define an accuracy interval on which the iterates $x_i$ stagnate

(from a certain $i$). This "accuracy interval" can be interpreted as the limiting accuracy for the iterative refinement iterates, that is, the accuracy at which we guarantee the iterates to converge.

The red error comes from the inexact computation and application of $\nabla f(\widehat{x}_i)$; it affects the slope of the red tangent curve to $f(x)$ at $(\widehat{x}_i, \widehat{f}(\widehat{x}_i))$. The computed tangent is guaranteed to be contained in the red zone representing the error range of the computation of the correction equation solution. Because of this error, the red and blue curves are not parallel. Naturally, in exact arithmetic, the tangent to $f(x) = Ax - b$ is trivially $f$ at any $x$, the red and blue curves should, in that case, overlap and $x$ is obtained after one iteration. This error affects the convergence speed; the more the red curve is parallel to the blue one, the faster we converge to the final limiting accuracy. The opposite consequence is that if the red curve is not sufficiently parallel to the blue one, we might take more iterations to converge or simply diverge. Therefore, the red error shall describe both the convergence speed and the condition under which we can converge.

The purpose of Figure 4.1 and its interpretation is only to intuitively understand the iterative refinement process and what are the impacts of the different rounding errors. In the following sections, we will develop more formally these different observations.

## 4.2 Generalized iterative refinement

We now present the generalized form of iterative refinement proposed by Carson and Higham [45] and the main conclusions resulting from their error analysis. The tools developed in their article are the basis for almost every latest analysis of iterative refinement, including the analyses carried out in this manuscript. Therefore, due to their importance for this work, we carefully recall, contextualize, and explain how to use them.

### 4.2.1 Preliminaries

The *generalized iterative refinement* is represented by Algorithm 4.1, it consists in *"refining"* a first approximation $x_0$ of the solution of (1.1) to improve its accuracy (as described by steps (4.1)–(4.3)). In a *refinement step*, we have the computation of the linear system residual in precision $u_r$ (step 3), the computation of the correction equation in precision $u_s$ (step 4), and finally the computation of the update in precision $u$ (step 5). We call it "generalized" because both the solver used at step 4 and the set of precision $u_r$, $u_s$ and $u$ are generic. Therefore, Algorithm 4.1 covers almost all iterative refinement algorithms proposed in the literature so far, except for dynamic change of the precisions (see Kiełbasiński [134], Oktay and Carson [170]). The advantage of working on Algorithm 4.1 is that any results derived for this generic algorithm hold for any of its specialization, that is, given a certain solver at step 4 and a certain combination of precisions $(u, u_r, u_s)$.

In generalized iterative refinement, the solver at step 4 for the solution of the correction equation is subject to the following conditions on the returned computed solution $\widehat{d}_i$ :

$$\widehat{d}_i = (I + u_s E_i) d_i, \quad u_s \|E_i\|_\infty < 1, \tag{4.4}$$

**Algorithm 4.1** Generalized iterative refinement

**Input:** an $n \times n$ matrix $A$ and a right-hand side $b$.
**Output:** an approximate solution to $Ax = b$.

1: Initialize $x_0$.
2: **while not** converged **do**
3: Compute $r_i = b - Ax_i$. $(u_r)$
4: Solve $Ad_i = r_i$. $(u_s)$
5: Compute $x_{i+1} = x_i + d_i$. $(u)$
6: **end while**

$$\|\widehat{r}_i - A\widehat{d}_i\|_\infty \leq u_s(c_1\|A\|_\infty\|\widehat{d}_i\|_\infty + c_2\|\widehat{r}_i\|_\infty), \tag{4.5}$$
$$|\widehat{r}_i - A\widehat{d}_i| \leq u_s G_i|\widehat{d}_i|, \tag{4.6}$$

where $E_i$, $c_1$, $c_2$, and $G_i$ are functions of $n$, $A$, $\widehat{r}_i$ and $u_s$ and have nonnegative entries. Few comments on this model:

- $u_s$ is the accuracy at which the solver delivers the computed solution $\widehat{d}_i$ and is not necessarily directly related to the unit roundoff of a given precision (despite the notation rules of section 2.1.4). Though, throughout the manuscript, we will abusively refer to $u_s$ as a precision for convenience and readability.
- In the same philosophy as the work of Jankowski and Woźniakowski [129], this solver model is generic enough to cover many kinds of linear solvers: direct or iterative, backward stable or not. The potential instability of the solver is carried by the terms $E_i$, $c_1$, $c_2$, and $G_i$.
- Conditions (4.4) and (4.5) are required, respectively, for the forward error convergence and the normwise backward error convergence. In contrast, condition (4.6) is required for the componentwise backward error convergence that we do not cover in this manuscript but which has been tackled by Carson and Higham [45, sect. 5].

In addition, we will make two assumptions on the precisions:

- We need the unit roundoff of the working precision $u$ to be smaller than the accuracy at which the solver at step 4 of Algorithm 4.1 delivers $\widehat{d}_i$, i.e.,

$$u \leq u_s\|E_i\|_\infty. \tag{4.7}$$

  It is relatively natural since $\widehat{d}_i$ is cast in precision $u$ at step 5; thus, $\widehat{d}_i$ will not be more accurate than $u$ when added to $\widehat{x}_i$. This assumption is essentially cosmetic and is made to avoid terms of the form $\max(u, u_s\|E_i\|_\infty)$ in the coming bounds (4.9) and (4.16).
- Specifically for the normwise backward error analysis, we need to ensure that the problem is not singular at the solver precision, that is,

$$c_1\kappa_\infty(A)u_s < 1. \tag{4.8}$$

### 4.2.2 Forward and backward errors analyses

We now recall two major results from Carson and Higham [45] that guarantee Algorithm 4.1 to provide a computed solution $\widehat{x}$ of (1.1) whose forward and backward errors will meet well-defined *limiting accuracies* under well-defined *convergence conditions*. There is one theorem each for the forward and backward errors which are a direct rewriting of [45, Cor. 3.3] and [45, Cor. 4.2].

**Theorem 4.1** (Forward error convergence)**.** *Let Algorithm 4.1 be applied to (1.1) where $A \in \mathbb{R}^{n\times n}$ is nonsingular, and assume the solver used at step 4 satisfies (4.4) and (4.7). As long as*

$$\phi_i = 2u_s \min(\text{cond}(A), \kappa_\infty(A)\mu_i) + u_s\|E_i\|_\infty \tag{4.9}$$

*is sufficiently less than 1, the forward error is reduced on the i th iteration by a factor approximatively $\phi_i$ until an iterate $\widehat{x}_i$ is produced for which*

$$\frac{\|x-\widehat{x}\|_\infty}{\|x\|_\infty} \lesssim 4p\,u_r\,\text{cond}(A,x) + u. \tag{4.10}$$

*Proof.* We will only present the core ideas of the proof; the full details can be found in [45, sect. 3]. The proof consists mainly in expressing the quantities $\phi_i$ and $\varepsilon_i$ such that $\|x-\widehat{x}_{i+1}\|_\infty \leq \phi_i\|x-\widehat{x}_i\|_\infty + \varepsilon_i$. Therefore, if $\phi_i < 1$, the error $\|x-\widehat{x}_i\|_\infty$ will reduce as we iterate until it reaches $\varepsilon_i$ where, actually, the requirement $\phi_i < 1$ corresponds to the convergence condition.

Bounding $\|x-\widehat{x}_{i+1}\|_\infty$ in terms of $\|x-\widehat{x}_i\|_\infty$ can be made by observing that

$$\widehat{x}_{i+1} = x + A^{-1}\Delta r_i + (\widehat{d}_i - A^{-1}\widehat{r}_i) + \Delta x_i, \tag{4.11}$$

where $\Delta r_i$ is the error made on the computation of the residual at step 3 in precision $u_r$, $\Delta x_i$ is the error made on the computation of the update at step 5 in working precision $u$, and $\widehat{d}_i - A^{-1}\widehat{r}_i$ is the forward error made by the solver at step 4 and is defined by condition (4.4). To bound these different errors, we need to introduce the scalar $\mu_i$ defined such that

$$\|A(x-\widehat{x}_i)\|_\infty = \mu_i\|A\|_\infty\|x-\widehat{x}_i\|_\infty, \tag{4.12}$$

and satisfying by definition

$$\frac{\|A(x-\widehat{x}_i)\|_\infty}{\|A\|_\infty\|x\|_\infty} = \mu_i\frac{\|x-\widehat{x}_i\|_\infty}{\|x\|_\infty}. \tag{4.13}$$

It can therefore be interpreted as the ratio between the normwise backward error and the forward error of the original linear system (1.1) for the iterate $\widehat{x}_i$. This ratio is known to be at least of order $\kappa_\infty(A)^{-1}$ (see (2.7)), giving

$$\kappa_\infty(A)^{-1} \leq \mu_i \leq 1. \tag{4.14}$$

With this quantity, we can bound $\|b-A\widehat{x}_i\|_\infty = \|A(x-\widehat{x}_i)\|_\infty$ in terms of $\|x-\widehat{x}_i\|_\infty$. We do not provide the details on bounding these errors, but they can be found in [45, sect. 3].

Using (4.11) with the bounds on $\Delta r_i$, $\Delta x_i$, and $\widehat{d}_i - A^{-1}\widehat{r}_i$ gives

$$\|x - \widehat{x}_{i+1}\|_\infty \lesssim (2u_s \min(\mathrm{cond}(A), \kappa_\infty(A)\mu_i) + u_s\|E_i\|_\infty)\|x - \widehat{x}_i\|_\infty \tag{4.15a}$$

$$+ 2pu_r \,\mathrm{cond}(A,x)(\|x\|_\infty + \|\widehat{x}_i\|_\infty) + u\|\widehat{x}_{i+1}\|_\infty, \tag{4.15b}$$

where we can identify $\phi_i$ and $\varepsilon_i$. The limiting accuracy (4.10) is obtained from $\varepsilon_i$ and by assuming that from a certain iteration $i$ we have $\|x_i\|_\infty \approx \|x_{i+1}\|_\infty \approx \|x\|_\infty$. □

The convergence condition (4.9) is made of the two terms $2u_s \min(\mathrm{cond}(A), \kappa_\infty(A)\mu_i)$ and $u_s\|E_i\|_\infty$, where the first one depends on the quantity $\mu_i$ previously introduced. In practice, we observed that $\mu_i$ is constant over the iterations, near its lower bound $\kappa(A)^{-1}$ (see (4.14)), until the backward error converges. Once the backward error has converged to double precision accuracy (say), the ratio between the backward and the forward error will increase if the forward error shall also converge to double precision accuracy. Therefore, near the end of the iterations, $\mu_i$, which quantifies this ratio, is expected to increase. Another interpretation of this quantity is proposed in [44, sect. 2.1]. In particular, it means that $\phi_i$ will most probably be dominated by the term $u_s\|E_i\|_\infty$ rather than by $2u_s \min(\mathrm{cond}(A), \kappa_\infty(A)\mu_i)$, at least, until the backward error converges. Thus, we will generally focus on the term $u_s\|E_i\|_\infty$ when working on bounding $\phi_i$, and we will ignore the other.

Turning our attention to the limiting accuracy (4.10), we can observe that the term in $u_r$ carries the conditioning of the problem $\mathrm{cond}(A,x)$. Consequently, when we choose $u_r = u$, the forward error of the computed solution should be of order $u\,\mathrm{cond}(A,x)$. It is possible to remove this dependence on the conditioning by choosing $u_r \ll u$, such that $u_r\,\mathrm{cond}(A,x) \leq u$. In doing so, we can obtain a forward error of order $u$. It is not meaningful to set $u_r > u$.

**Theorem 4.2** (Backward error convergence)**.** *Let Algorithm 4.1 be applied to* (1.1) *where* $A \in \mathbb{R}^{n\times n}$ *is nonsingular and satisfies* $c_1\kappa_\infty(A)u_s < 1$*, and assume the solver used at step 4 satisfies* (4.5) *and* (4.7)*. As long as*

$$\phi = u_s(c_1\kappa_\infty(A) + c_2) \tag{4.16}$$

*is sufficiently less than* 1*, the residual is reduced on each iteration by a factor approximatively* $\phi$ *until an iterate* $\widehat{x}_i$ *is produced for which*

$$\|b - A\widehat{x}_i\|_\infty \lesssim \gamma_p^r(\|b\|_\infty + \|A\|_\infty\|\widehat{x}_{i-1}\|_\infty) + u\|A\|_\infty\|x_i\|_\infty. \tag{4.17}$$

*Proof.* The proof follows the same philosophy as the proof of Theorem 4.1. We want to express the quantity $\phi$ and $\varepsilon_i$ such that $\|b - A\widehat{x}_{i+1}\|_\infty \leq \phi\|b - A\widehat{x}_i\|_\infty + \varepsilon_i$. Bounding $\|b - A\widehat{x}_{i+1}\|_\infty$ in terms of $\|b - A\widehat{x}_i\|_\infty$ can be made by observing that

$$A\widehat{x}_{i+1} - b = \Delta r_i + A\widehat{d}_i - \widehat{r}_i + A\Delta x_i, \tag{4.18}$$

where $\Delta r_i$ and $\Delta x_i$ are the same as in the proof of Theorem 4.1. It remains to bound $A\widehat{d}_i - \widehat{r}_i$,

with condition (4.5) we have

$$\|A\widehat{d}_i - \widehat{r}_i\|_\infty \leq u_s(c_1\|A\|_\infty\|\widehat{d}_i\|_\infty + c_2\|\widehat{r}_i\|_\infty) \tag{4.19a}$$
$$\leq u_s(c_1\|A\|_\infty\|A^{-1}\|_\infty(\|\widehat{r}_i\|_\infty + \|A\widehat{d}_i - \widehat{r}_i\|_\infty) + c_2\|\widehat{r}_i\|_\infty), \tag{4.19b}$$

where, under the assumption (4.8), we obtain

$$\|A\widehat{d}_i - \widehat{r}_i\|_\infty \leq u_s \frac{c_1\kappa_\infty(A) + c_2}{1 - c_1\kappa_\infty(A)u_s}\|\widehat{r}_i\|_\infty. \tag{4.20}$$

Using (4.18) with the bounds on $\Delta r_i$, $\Delta x_i$, and $A\widehat{d}_i - \widehat{r}_i$ gives

$$\|b - A\widehat{x}_{i+1}\|_\infty \lesssim u_s(c_1\kappa_\infty(A) + c_2)\|b - A\widehat{x}_i\|_\infty \tag{4.21a}$$
$$+ \gamma_p^r(\|b\|_\infty + \|A\|_\infty\|\widehat{x}_i\|_\infty) + u\|A\|_\infty\|x_{i+1}\|_\infty, \tag{4.21b}$$

where we can identify $\phi$ and $\varepsilon_i$. We skipped the details for obtaining this last bound, but they can be found in [45, sect. 4]. □

Making the reasonable assumption that $\|\widehat{x}_{i-1}\|_\infty \approx \|\widehat{x}_i\|_\infty$, we can conclude from Theorem 4.2 that

$$\frac{\|b - A\widehat{x}_i\|_\infty}{\|b\|_\infty + \|A\|_\infty\|x_i\|_\infty} \lesssim p(u + u_r). \tag{4.22}$$

Therefore, if $u_r = u$, generalized iterative refinement is guaranteed to offer a backward stable solution for $Ax = b$ to the working precision $u$. An important aspect of this result is that, considering the fixed precision case where $u_r = u = u_s$, iterative refinement can transform a potentially unstable solver (under condition (4.5)) into a backward stable one. In particular, if we consider a direct LU solver, the presence of large element growth during the factorization is a well-known issue that prevents the stability (see sections 2.2.1.2 and 2.2.3.7), but this can be rectified by the use of iterative refinement.

### 4.2.3 Various practical comments

Theorems 4.1 and 4.2 are the current standard tools to guarantee stability and convergence of mixed precision iterative refinement algorithms. They have been extensively used on most of the recent iterative refinement analyses: Carson and Higham [45], Carson et al. [47], Amestoy et al. [18], Oktay and Carson [170], Amestoy et al. [19], Carson and Khan [46], including the contributions that will be presented in this manuscript. We make some comments on these major results:

- To derive stability results for a given specialized iterative refinement algorithm, we just need to determine $E_i$, $c_1$, $c_2$ and $u_s$ for the chosen linear solver used at step 4. The theorems will then guarantee the convergence of the errors to the limiting accuracies (4.10) and (4.17) at the convergence rates (4.9) and (4.16).

- A central property stated by these theorems is that the limiting accuracies are only determined by the precision $u$ and $u_r$ and are independent of the solver used at step 4.

On the other hand, the convergence properties (rate and ability) are only determined by the accuracy at which the solver computes $\widehat{d}_i$. It is consistent with the intuitions developed in section 4.1 from the illustration Figure 4.1.

- Since the backward error is guaranteed to be always lower than the forward error (see (2.7)), Theorem 4.1 implicitely covers the convergence of the normwise backward error of (1.1) as well. What makes Theorem 4.2 different is that it guarantees a stronger limiting accuracy for the backward error. Particularly, we do not have dependence on the system's conditioning $\text{cond}(A, x)$ even when $u_r = u$.

### 4.2.4 Targeting low precisions

Iterative refinement has become very attractive nowadays because it can potentially improve the computing performance of many kinds of linear solvers for the solution of (1.1). The strategy consists in using low resource demanding low precision(s) (see section 2.1.3) in the solver at step 4 while refining in higher precision with $u, u_r \ll u_s$ for step 3 and 5. As the application of the solver should be the most computationally demanding part, the use of low precision(s) can potentially bring substantial performance improvements for the computation of the solution. This strategy was popularized in the 2000s (see section 3.4) and covers different kinds of linear solvers; for example, direct solvers with Langou et al. [136], iterative solvers with Strzodka and Göddeke [197], and multigrid solvers with Göddeke et al. [87].

## 4.3 LU-IR3

When we specialize the solver at step 4 of Algorithm 4.1 to be an LU solver applied in precision $u_f$ (see section 2.2.1), we call the algorithm *LU-based iterative refinement in three precisions (LU-IR3)* (Carson and Higham [45]). The "three precisions" stands for the ability to independently set the precisions $u_f$, $u$, and $u_r$. We describe LU-IR3 in Algorithm 4.2, where it should be noted that, as the computed LU factors $\widehat{L}$ and $\widehat{U}$ stay identical over the iterations, we rather factorize once at step 1 and use the same computed factors throughout all the iterations. As iterative refinement has been first used with GE, LU-based iterative refinement is generally considered as the traditional form of iterative refinement.

**Algorithm 4.2** LU-IR3

**Input:** an $n \times n$ matrix $A$ and a right-hand side $b$.
**Output:** an approximate solution to $Ax = b$.

1: Factorize $A = \widehat{L}\widehat{U}$. $(u_f)$
2: Solve $\widehat{L}\widehat{U}x_0 = b$ for $x_0$. $(u_f)$
3: **while not** converged **do**
4: Compute $r_i = b - Ax_i$. $(u_r)$
5: Solve $\widehat{L}\widehat{U}d_i = r_i$ for $d_i$. $(u_f)$
6: Compute $x_{i+1} = x_i + d_i$. $(u)$
7: **end while**

### 4.3.1 Error analysis

Using the generalized iterative refinement theorems of section 4.2, we can derive the following result for LU-IR3.

**Theorem 4.3** (Convergence of LU-IR3)**.** *Let* (1.1) *be solved by LU-IR3 (Algorithm 4.2) using GEPP at step 1. If* $\kappa(A)u_f \geq u$*, then the forward and backward errors will reach their respective limiting accuracies*

$$p\,u_r\,\mathrm{cond}(A,x)+u \text{ (forward)} \qquad \text{and} \qquad p\,u_r+u \text{ (backward)}, \tag{4.23}$$

*provided that*

$$u_f\kappa(A) \ll 1 \text{ (forward and backward)}. \tag{4.24}$$

*Proof.* Solving the correction equation at step 5 with a GEPP applied in precision $u_f$ will deliver a computed solution $\widehat{d}_i$ such that

$$\frac{\|\widehat{r}_i - A\widehat{d}_i\|_\infty}{\|A\|_\infty\|\widehat{d}_i\|_\infty} \leq f(n,\rho_n)u_f, \qquad \frac{\|d_i - \widehat{d}_i\|_\infty}{\|d_i\|_\infty} \leq f(n,\rho_n)\kappa_\infty(A)u_f, \tag{4.25}$$

which is obtained from Theorem 2.6, and where $f(n,\rho_n)$ is a function of $n$ and $\rho_n$. We can identify $u_s \equiv u_f$, $\|E_i\|_\infty \equiv f(n,\rho_n)\kappa_\infty(A)$, $c_1 \equiv f(n,\rho_n)$, and $c_2 \equiv f(n,\rho_n)$. As stated in section 2.2.1.2, with GEPP we expect $\rho_n$ to be small, therefore, dropping $f(n,\rho_n)$, applying Theorems 4.1 and 4.2, and dropping the remaining constants in the convergence condition ends the proof. Note that condition $\kappa(A)u_f \geq u$ translates (4.7), and that condition (4.8) is implicitly included in the convergence condition (4.24). □

### 4.3.2 Targeting low precisions

As the factorization is expected to be the dominant operation ($\mathcal{O}(n^3)$ flops for dense systems), a well-known strategy to accelerate an LU direct solver is to set $u_f$ to low precision. While the solution $x_0$ computed at step 2 will have a low accuracy, the refinement steps in high precision (i.e., $u, u_r \leq u_f$), supposed negligible compared with the factorization in $u_f$ ($\mathcal{O}(n^2)$ flops for dense systems), will improve the solution to its limiting accuracies at a low cost.

Necessarily, this strategy is only applicable if the method can converge. When looking at the convergence condition (4.24), we observe that the ability of LU-IR3 to refine the solution to higher accuracy depends only on the capacity of $u_f \ll 1$ to compensate the term $\kappa(A) \gg 1$. It means that if we are using low precision to compute the factorization, we will be constrained to process problems with small condition numbers (e.g., with $u_f = \text{fp16}$ condition (4.24) becomes $\kappa(A) \ll 2\times 10^3$).

## 4.4 LU-GMRES-IR3

The restrictive convergence condition of LU-IR3 when very low precision factorization is employed motivated the conception of a new iterative refinement algorithm called *LU pre-*

*conditioned GMRES-based iterative refinement in three precisions (LU-GMRES-IR3)*. With LU-GMRES-IR3, we specialize the solver at step 4 of Algorithm 4.1 to be a left-preconditioned GMRES by the LU factors computed in precision $u_f$ (see section 2.3.1), where this GMRES applies all its operations in precision $u$ except the preconditioned matrix–vector product applied in a higher precision $u^2$. We describe LU-GMRES-IR3 in Algorithm 4.3, where it should be noted that on line 5 $\widetilde{A}$ is not explicitly formed, but its action on a vector is obtained with a matrix multiplication followed by two triangular solves. To avoid ambiguity, we will often refer to the iterations of iterative refinement as the outer iterations and the iterations of GMRES as the inner iterations.

**Algorithm 4.3** LU-GMRES-IR3

**Input:** an $n \times n$ matrix $A$ and a right-hand side $b$.
**Output:** an approximate solution to $Ax = b$.
1: Factorize $A = \widehat{L}\widehat{U}$. $(u_f)$
2: Solve $\widehat{L}\widehat{U}x_0 = b$ for $x_0$. $(u_f)$
3: **while not** converged **do**
4: Compute $r_i = b - Ax_i$. $(u_r)$
5: Solve $\widehat{U}^{-1}\widehat{L}^{-1}Ad_i = \widehat{U}^{-1}\widehat{L}^{-1}r_i$ by GMRES at precision $u$ with matrix–vector products with $\widetilde{A} = \widehat{U}^{-1}\widehat{L}^{-1}A$ computed at precision $u^2$.
6: Compute $x_{i+1} = x_i + d_i$. $(u)$
7: **end while**

### 4.4.1 Error analysis

Using the generalized iterative refinement's theorems of section 4.2, we can derive the following result for LU-GMRES-IR3.

**Theorem 4.4** (Convergence of LU-GMRES-IR3)**.** *Let* (1.1) *be solved by LU-GMRES-IR3 (Algorithm 4.3) using MGS-GMRES at step 5 and using GEPP at step 1. If* $\kappa(A)u \leq 1$*, the forward and backward errors will reach their respective limiting accuracies*

$$p\,u_r\,\mathrm{cond}(A,x) + u \text{ (forward)} \qquad \text{and} \qquad p\,u_r + u \text{ (backward)}, \tag{4.26}$$

*provided that*

$$u_f\,u^{1/2}\kappa(A) \ll 1 \text{ (forward)} \qquad \text{and} \qquad u_f^{1/2}u^{1/2}\kappa(A) \ll 1 \text{ (backward)}. \tag{4.27}$$

*Proof.* As LU-GMRES-IR3 is a specialization of LU-GMRES-IR5 that we will introduce and study in chapter 5 of this manuscript, using Theorem 5.1 with the precision parameters $u_g$ and $u_p$ (introduced in this chapter as well) set to $u_p = u^2$ and $u_g = u$ gives the result directly.

For the original proof, we refer the reader to Carson and Higham [44, sect. 3] and Carson and Higham [45, sect. 8]. We will not attempt to rewrite it because it would be long and redundant with the proof in chapter 5, we instead provide the essential ideas in the following.

As for the proof of Theorem 4.3, we need to identify $u_s$, $\|E_i\|_\infty$, $c_1$, and $c_2$ to make use of Theorems 4.1 and 4.2. To do so, we must determine bounds for the forward and backward errors of the computed solution of the correction equation $Ad_i = \widehat{r}_i$. As MGS-GMRES actually solves the preconditioned system $\widetilde{A}d_i = s_i$, where $\widetilde{A} = \widehat{U}^{-1}\widehat{L}^{-1}A$ and $s_i = \widehat{U}^{-1}\widehat{L}^{-1}\widehat{r}_i$, we proceed as follows: we first bound the error $s_i - \widehat{s}_i$ in forming the preconditioned right-hand side, we show that the backward stability property of MGS-GMRES proved by Paige et al. [175] for the solution of unpreconditioned systems holds for the solution of the preconditioned system $\widetilde{A}d_i = \widehat{s}_i$ where $\widetilde{A}$ is applied in precision $u^2$ and the rest of the operations in precision $u$, we use these previous results to determine bounds on the forward and backward error of the original system $Ad_i = \widehat{r}_i$, from these bounds we identify the different terms as $u_s \equiv u$, $\|E_i\|_\infty \equiv f(n,k,\rho_n)\kappa_\infty(\widetilde{A})$, $c_1 \equiv f(n,k,\rho_n)\|\widetilde{A}\|_\infty$, and $c_2 \equiv f(n,k,\rho_n)\kappa_\infty(A)$ and conclude on the convergence conditions of LU-GMRES-IR3. The constant $k$ corresponds to the maximum number of GMRES inner iterations over the outer iterations.

Note that condition (4.7) is met and (4.8) is implicitly included in the normwise backward convergence condition (4.27). The condition $\kappa(A)u \leq 1$ allows for simplifications of the form $\kappa(A)u^2 \leq u$ and is used especially by Carson and Higham [44] to show that the backward stability result of Paige et al. [175] still holds.

Actually, the condition on the backward error is stricter than what has been proposed in Carson and Higham [45] (i.e., $u\kappa(A) \ll 1$). The reason for this discrepancy is that they assumed that $c_1 \equiv 1$ in (4.16), which is too optimistic in general. □

### 4.4.2 LU-GMRES-IR3 vs LU-IR3

LU-GMRES-IR3 is more resilient on the conditioning of the problem since conditions (4.27) (for LU-GMRES-IR3) are looser than condition (4.24) (for LU-IR3) (e.g., with $u_f$ = fp16 and $u$ = fp64 we obtain the condition $\kappa(A) \ll 2 \times 10^{11}$ on the forward error, which is far more permissive than the previous condition $\kappa(A) \ll 2 \times 10^3$). It means both that the method can achieve convergence for more ill-conditioned problems and that it can converge at a faster rate (i.e., less outer iterations). In Table 4.1, we compare the convergence conditions of the forward error of LU-IR3 and LU-GMRES-IR3 for different precision $u_f$.

Table 4.1: Condition on $\kappa(A)$ for the convergence of the forward error with LU-IR3 and LU-GMRES-IR3 for different precision $u_f$ and with $u$ = D fixed. The symbols D, S, H, B, and R refer to the arithmetics of Table 2.1.

| $u_f$ | LU-IR3 | LU-GMRES-IR3 |
|---|---|---|
| R | $2 \times 10^1$ | $2 \times 10^9$ |
| B | $3 \times 10^2$ | $2 \times 10^{10}$ |
| H | $2 \times 10^3$ | $2 \times 10^{11}$ |
| S | $2 \times 10^7$ | $2 \times 10^{15}$ |

One major difference between LU-GMRES-IR3 and LU-IR3 is that the LU factors are applied in a high precision $u^2$ instead of the low precision $u_f$. Specifically, it means that for the same set of precision $u_f$, $u$, and $u_r$ an outer iteration of LU-GMRES-IR3 is more

expensive in execution time and possibly in memory consumption than an iteration of LU-IR3. It is because, first, the application of the LU factors in precision $u^2$ is expected to be more costly in time than the application of these factors in precision $u_f$. Second, the factors computed in precision $u_f$ will need to be cast at some point in precision $u^2$, and, depending on if we choose to store the factors in precision $u^2$ fully, LU-GMRES-IR3 can increase the memory consumption. However, LU-GMRES-IR3 will do less outer iterations than LU-IR3; therefore, it is not straightforward to predict if LU-GMRES-IR3 will be more expensive than LU-IR3 as a whole, and if it is the case, to estimate how much more expensive it is. Comparisons between practical implementations of LU-GMRES-IR3 and LU-IR3 have been made for dense systems by Haidar et al. [101; 102; 103], a comparison for sparse systems is the topic of chapter 6 of this manuscript.

### 4.4.3 GMRES-IR

We refer more generally to the class of iterative refinement algorithms that use a GMRES solver at step 4 of Algorithm 4.1 by *GMRES-based iterative refinement (GMRES-IR)*. Hence, LU-GMRES-IR3, which uses a left-preconditioned GMRES by the LU factors, is a representative of this class, but it is not the only one. For instance, there are studies using GMRES-IR with other various kinds of preconditioners: ILU (Lindquist et al. [147]), polynomial (Loe et al. [151]), or approximate inverse (Carson and Khan [46]).

Actually, as we will explain in more detail in chapter 7, restarted GMRES can be seen as a form of GMRES-IR. Therefore, GMRES-IR is a relatively old approach; from our knowledge, its first mixed precision implementation has been proposed by Turner and Walker [205] who used $u = u_r \leq u_s$.

## 4.5 Extension to least squares problem

Mixed precision iterative refinement can be extended to improve the solution of the least squares problem (2.25) as well. We will review three main ways to do it, where the last one is probably the most versatile and robust.

### 4.5.1 Iterative refinement on the normal equations

One of the most straightforward ways is to use iterative refinement for the solution of the normal equations (2.26), that is, using LU-IR3 or LU-GMRES-IR3 on the system $A^T Ax = A^T b$ by replacing the LU factors with the Cholesky factors. However, this is also one of the least robust approaches since the conditioning of this system satisfies $\kappa(A)^2 \leq f(n)\kappa(A^T A)$, which strongly restrict the respective convergence conditions (4.24) and (4.27). For example, with $u_f = \text{fp16}$ and $u = \text{fp64}$, condition (4.24) becomes $\kappa(A) \ll 5 \times 10^1$ and condition (4.27) becomes $\kappa(A) \ll 4 \times 10^5$. Actually, in this configuration, LU-IR3 becomes almost unusable with a low precision factorization. The use of LU-GMRES-IR3 for the solution of (2.25) with the normal equation has been explored by Higham and Pranesh [120].

### 4.5.2 Iterative refinement on the overdetermined system

Another early approach to solve (2.25) through iterative refinement was proposed by Golub [88] and used by Bauer [31]. It is described and extended to three precisions in Algorithm 4.4; it can be noted that the original algorithm is in two precisions $u_f = u \geq u_r$. This approach can be viewed as the application of LU-IR3 on the overdetermined system $Ax = b$, where we replace the LU factorization with the QR one. For this reason, we call the method *QR-based iterative refinement in three precisions (QR-IR3)*.

**Algorithm 4.4** QR-IR3

**Input:** an $m \times n$ matrix $A$ and a vector $b$.
**Output:** an approximate solution to $\min_x \|b - Ax\|_2$.

1: Factorize $A = \begin{bmatrix} \widehat{Q}_1 & \widehat{Q}_2 \end{bmatrix} \begin{bmatrix} \widehat{R} \\ 0 \end{bmatrix}$. $(u_f)$
2: Solve $\widehat{R}\, x_0 = \widehat{Q}_1^T b$ for $x_0$. $(u_f)$
3: **while not** converged **do**
4: Compute $r_i = b - Ax_i$. $(u_r)$
5: Solve $\min_{d_i} \|r_i - Ad_i\|_2$ (eq. Solve $\widehat{R}\, d_i = \widehat{Q}_1^T r_i$ for $d_i$). $(u_f)$
6: Compute $x_{i+1} = x_i + d_i$. $(u)$
7: **end while**

While this proposition is simple and computationally efficient, it does not actually work for the general case. If we define $f(x) = Ax - b$, there might not be any $x$ such that $f(x) = 0$ because the overdetermined system $Ax = b$ does not have a solution in general. Therefore, QR-IR3 can be interpreted as the application of Newton's method for finding a zero of a function $f$ that does not have a zero. While it might sound absurd at first, doing so has some sense, and in the following we will try to provide a rough intuition of why.

(4.18) and (4.11) tell us that the errors on the $(i+1)$th iterate $\widehat{x}_{i+1}$ depend on the terms $A^{-1}\Delta r_i$, $\Delta x_i$, and $A\widehat{d}_i - \widehat{r}_i$ (or $\widehat{d}_i - A^{-1}\widehat{r}_i$). While the evaluation of the first two terms does not change in comparison with the square linear system case, the last term is now bounded by (2.32), that is,

$$\begin{aligned} \|\widehat{r}_i - A\widehat{d}_i\|_\infty \leq f(n,m)(\||\widehat{r}_i| + |A||d_i|\|_\infty + \mathrm{cond}_2(A^T)\|\widehat{r}_i - Ad_i\|_\infty)u_f \\ + \|\widehat{r}_i - Ad_i\|_\infty. \end{aligned} \tag{4.28}$$

In particular, this bound shows that the quality of the computed correction $\widehat{d}_i$ might be of order $\|\widehat{r}_i - Ad_i\|_\infty = \|b - A(\widehat{x}_i + d_i)\|_\infty = \min_x \|b - Ax\|_\infty$, and thus, if $\min_x \|b - Ax\|_\infty \gg 1$, the computed correction $\widehat{d}_i$ will be too poor, and the method will not converge. On the other hand, for nearly consistent overdetermined systems (i.e., $\min_x \|b - Ax\|_2$ close to 0), the term $\||\widehat{r}_i| + |A||d_i|\|_\infty u_f$ might become dominant. In this case, iterative refinement will improve the solution. This observation has been previously made by Golub and Wilkinson [91].

### 4.5.3 Iterative refinement on the augmented system

The two previous approaches described in sections 4.5.1 and 4.5.2 present major downsides: either we are restricted to working with well-conditioned systems or with consistent overdetermined ones. To relax these restrictions, Björck [32] proposed to use iterative refinement for the solution of the augmented system

$$A_+ z = b_+, \quad A_+ = \begin{bmatrix} I & A \\ A^T & 0 \end{bmatrix}, \quad z = \begin{bmatrix} r \\ x \end{bmatrix}, \quad b_+ = \begin{bmatrix} b \\ 0 \end{bmatrix}, \tag{4.29}$$

which is, in exact arithmetic, an equivalent problem to the normal equations (2.26). As the augmented system is a square linear system, iterative refinement can be applied "as usual" and will refine the solution $[r^T\, x^T]^T$. We call the method *QR-based iterative refinement on the augmented system in three precisions (AQR-IR3)*, and it is described in Algorithm 4.5.

**Algorithm 4.5** AQR-IR3

**Input:** an $m \times n$ matrix $A$ and a right-hand side $b$.
**Output:** an approximate solution to $\min_x \|b - Ax\|_2$.

1: Factorize $A = \begin{bmatrix} \widehat{Q}_1 & \widehat{Q}_2 \end{bmatrix} \begin{bmatrix} \widehat{R} \\ 0 \end{bmatrix}$. $(u_f)$
2: Solve $\widehat{R}\, x_0 = \widehat{Q}_1^T b$ for $x_0$. $(u_f)$
3: **while not** converged **do**
4: Compute $\begin{bmatrix} h_i \\ g_i \end{bmatrix} = \begin{bmatrix} b - r_i - Ax_i \\ -A^T r_i \end{bmatrix}$. $(u_r)$
5: Solve $\begin{bmatrix} I & A \\ A^T & 0 \end{bmatrix} \begin{bmatrix} d_{r_i} \\ d_{x_i} \end{bmatrix} = \begin{bmatrix} h_i \\ g_i \end{bmatrix}$ for $\begin{bmatrix} d_{r_i} \\ d_{x_i} \end{bmatrix}$. $(u_f)$
6: Compute $\begin{bmatrix} r_{i+1} \\ x_{i+1} \end{bmatrix} = \begin{bmatrix} r_i \\ x_i \end{bmatrix} + \begin{bmatrix} d_{r_i} \\ d_{x_i} \end{bmatrix}$. $(u)$
7: **end while**

Note that, at step 1, we are not doing an LU factorization of the matrix of the augmented system itself (4.29), but we rather compute the QR factorization of $A$. The solve at step 5 is readily affected, and to solve the correction equation

$$\begin{bmatrix} I & A \\ A^T & 0 \end{bmatrix} \begin{bmatrix} d_{r_i} \\ d_{x_i} \end{bmatrix} = \begin{bmatrix} h_i \\ g_i \end{bmatrix} \tag{4.30}$$

we proceed as follows:

$$f = R^{-T} g_i, \qquad \begin{bmatrix} k_1 \\ k_2 \end{bmatrix} = Q^T h_i, \tag{4.31a}$$

$$d_{r_i} = Q \begin{bmatrix} f \\ k_2 \end{bmatrix}, \qquad d_{x_i} = R^{-1}(k_1 - f). \tag{4.31b}$$

In particular, we apply the factors $R$ and $Q$ twice, which makes step 5 of Algorithm 4.5 two times more costly than Algorithm 4.4 and should increase the time cost of a refine-

ment step. While the factorization at step 1 is still expected to be dominant compared with a refinement step, the execution time might become more sensitive to the number of iterations.

As (4.29) is a square linear system, the results in section 4.2 apply, and from the analysis in Carson et al. [47] we can derive the following theorem for AQR-IR3.

**Theorem 4.5** (Convergence of AQR-IR3)**.** *Let* (1.1) *be solved by AQR-IR3 (Algorithm 4.5) using the Householder method at step 1. Then the forward and backward errors of the augmented system* (4.29) *will reach their respective limiting accuracies*

$$p\,u_r\,\mathrm{cond}(A_+,z)+u \text{ (forward)} \qquad \text{and} \qquad p\,u_r+u \text{ (backward)}, \tag{4.32}$$

*provided that*

$$u_f\,\kappa(A_+)\ll 1 \text{ (forward and backward)}. \tag{4.33}$$

*Proof.* The proof is relatively similar to the one of Theorem 4.3. The main change in the analysis is located at step 5 for the application of the solve where we need to use [47, Thm. 2.1]. □

Carson et al. [47] stated that with the following scaling on the augmented matrix

$$A_{+,\alpha}\equiv\begin{bmatrix}\alpha I & A\\ A^T & 0\end{bmatrix} \tag{4.34}$$

using a proper parameter $\alpha=\sigma_{\min}(A)$, $\kappa(A_{+,\alpha})$ is the same order of magnitude as $\kappa(A)$ and, therefore, the convergence condition (4.33) can rather be interpreted in terms of $\kappa(A)$.

As for the square linear system case, this technique can improve the accuracy of the computed solution of the least squares problem substantially in comparison with a direct solver based on a QR factorization (see section 2.2.2). For the latter, the forward error on the computed residual $\widehat{r}$ satisfies (2.31) and depends on the precision $u_f$ at which the factorization has been computed and on the condition number of $A$. From (4.32), we know that AQR-IR3 instead provides an accuracy depending on the working precision $u$, and if we choose $u_r\ll u$, we can even remove the dependence on the condition number.

Carson et al. [47] also proposed to adapt LU-GMRES-IR3 for the LS case. The new method is represented by Algorithm 4.6 and uses the augmented system strategy; we call it *QR preconditioned GMRES-based iterative refinement on the augmented system in three precisions (AQR-GMRES-IR3)*. As we are working with the augmented system, we cannot simply use a preconditioner of the form $M=\widehat{Q}\widehat{U}$ in the same manner as for LU-GMRES-IR3. Therefore, Carson et al. [47] proposed the following preconditioner built from the computed QR factors

$$M=\begin{bmatrix}\alpha I & \widehat{Q}_1\widehat{R}\\ \widehat{R}^T\widehat{Q}_1^T & 0\end{bmatrix},\quad M^{-1}=\begin{bmatrix}\frac{1}{\alpha}(I-\widehat{Q}_1\widehat{Q}_1^T) & \widehat{Q}_1\widehat{R}^{-T}\\ \widehat{R}^{-1}\widehat{Q}_1^T & -\alpha\widehat{R}^{-1}\widehat{R}^{-T}\end{bmatrix}. \tag{4.35}$$

If the factors are computed in exact arithmetic, we have $M^{-1}A_{+,\alpha}=I$.

**Algorithm 4.6** AQR-GMRES-IR3

**Input:** an $m \times n$ matrix $A$ and a right-hand side $b$.
**Output:** an approximate solution to $\min_x \|b - Ax\|_2$.

1: Factorize $A = \begin{bmatrix} \widehat{Q}_1 & \widehat{Q}_2 \end{bmatrix} \begin{bmatrix} \widehat{R} \\ 0 \end{bmatrix}$. $(u_f)$
2: Solve $\widehat{R} x_0 = \widehat{Q}_1^T b$ for $x_0$. $(u_f)$
3: **while not** converged **do**
4: Compute $\begin{bmatrix} h_i \\ g_i \end{bmatrix} = \begin{bmatrix} b - r_i - Ax_i \\ -A^T r_i \end{bmatrix}$. $(u_r)$
5: Solve $M^{-1} A_{+,\alpha} \begin{bmatrix} d_{r_i} \\ d_{x_i} \end{bmatrix} = M^{-1} \begin{bmatrix} h_i \\ g_i \end{bmatrix}$ by GMRES at precision $u$ with matrix–vector products with $\widetilde{A}_+ = M^{-1} A_{+,\alpha}$ computed at precision $u^2$.
6: Compute $\begin{bmatrix} r_{i+1} \\ x_{i+1} \end{bmatrix} = \begin{bmatrix} r_i \\ x_i \end{bmatrix} + \begin{bmatrix} d_{r_i} \\ d_{x_i} \end{bmatrix}$. $(u)$
7: **end while**

As (4.29) is a square linear system, the results in section 4.2 applies, and we can derive the following theorem for AQR-GMRES-IR3.

**Theorem 4.6** (Convergence of AQR-GMRES-IR3)**.** *Let* (1.1) *be solved by AQR-GMRES-IR3 (Algorithm 4.6) using MGS-GMRES at step 5 and using the Householder method at step 1. If* $\kappa(A)u < 1$*, the forward and backward errors will reach their respective limiting accuracies*

$$p\,u_r \operatorname{cond}(A_+, z) + u \text{ (forward)} \qquad \text{and} \qquad p\,u_r + u \text{ (backward)}, \tag{4.36}$$

*provided that*

$$u_f\, u^{1/2} \kappa(A) \ll 1 \text{ (forward)} \qquad \text{and} \qquad u_f^{1/2} u^{1/2} \kappa(A) \ll 1 \text{ (backward)}. \tag{4.37}$$

*Proof.* As AQR-GMRES-IR3 is a specialization of AQR-GMRES-IR5 that we will introduce and study in chapter 5 of this manuscript, using Theorem 5.4 with the precision parameters $u_g$ and $u_p$ (introduced in this chapter as well) set to $u_p = u^2$ and $u_g = u$ gives the result directly.

For the original proof, we refer the reader to Carson et al. [47, sect. 3.1]. It is relatively similar to the proof of Theorem 4.4, the main difficulty being the more complex application of the preconditioned matrix $\widetilde{A}_+$. □

In Theorem 4.6, the convergence conditions (4.37) are directly expressed with $\kappa(A)$ rather than $\kappa(A_+)$ because we suppose that we apply the scaling (4.34).

While the preconditioner defined by (4.35) has interesting theoretical properties guaranteeing the validity of Theorem 4.6, it has, unfortunately, poor practical interests. Indeed, we need to apply four times $\widehat{Q}_1$ and four times $\widehat{R}^{-1}$ at each iteration of GMRES to use it, which would increase the cost of the refinement steps severely. That is why Carson et al.

[47] rather encourage using a cheaper preconditioner for Algorithm 4.6

$$M_1 M_2 \equiv \begin{bmatrix} \sqrt{\alpha} I & 0 \\ 0 & \frac{1}{\sqrt{\alpha}} \widehat{R}^T \end{bmatrix} \begin{bmatrix} \sqrt{\alpha} I & 0 \\ 0 & \frac{1}{\sqrt{\alpha}} \widehat{R} \end{bmatrix} = \begin{bmatrix} \alpha I & 0 \\ 0 & \frac{1}{\alpha} \widehat{R}^T \widehat{R} \end{bmatrix}. \tag{4.38}$$

The two–sided application gives

$$M_1^{-1} A_{+,\alpha} M_2^{-1} = \begin{bmatrix} I & A\widehat{R} \\ \widehat{R}^{-T} A^T & 0 \end{bmatrix} \tag{4.39}$$

which only requires two applications of $\widehat{R}$.

## 4.6 Stopping criteria

As for any iterative process, it is of practical interest to be able to stop the algorithm at the right moment, for instance, when the solution accuracy is satisfactory or if we want to abort the process because the convergence is too slow or impossible. In iterative refinement, the convergence is really predictable and fully driven by the convergence rates (4.9) and (4.16). However, these convergence rates can depend on quantities that cannot be measured easily; in particular, the ones of LU-IR3 and LU-GMRES-IR3 depend on $\kappa(A)$ that cannot be computed inexpensively in general. Therefore, (4.9) and (4.16) cannot be used directly to stop the algorithm.

Demmel et al. [60] propose to stop the iterative refinement algorithm if any of the following three conditions applies. These conditions are based on the available computed quantities $\widehat{d}_{i+1}$, $\widehat{d}_i$, and $\widehat{x}_i$ at each iteration of Algorithm 4.1.

- $\|\widehat{d}_{i+1}\|_\infty / \|\widehat{x}_i\|_\infty \leq u$ : The relative computed correction $\widehat{d}_{i+1}$ is lower than the working accuracy at which the computed solution converges, so the computed solution $\widehat{x}_{i+1}$ will not improve $\widehat{x}_i$. It means that the solution has converged, and we can stop the algorithm. This condition is only valid when $u_r \, \mathrm{cond}(A, x) \leq u$, that is, when the limiting accuracy of the forward error does not depend on the condition number of $A$. As $\|\widehat{d}_{i+1}\|_\infty / \|\widehat{x}_i\|_\infty$ is an upper bound of the forward error, we can also choose to stop the algorithm before reaching the working accuracy. We just need to replace $u$ on the right-hand side of the bound with a targeted accuracy for the forward error.

- $\|\widehat{d}_i\|_\infty / \|\widehat{d}_{i+1}\|_\infty \leq \rho_t$ : The convergence rate is smaller than a prescribed quantity $\rho_t$. The convergence rate is measured by comparing the order of magnitude between the corrections from an iteration to the next. As the convergence rate is supposed to be approximately constant (if the solver properties $u_s$, $E_i$, $c_1$, and $c_2$ are relatively constant), when it begins to decrease to finally become smaller than $\rho_t$, it means that either: we have converged and the algorithm can be stopped, or we consider that the convergence rate is too slow and we abort the process.

- $i > i_t$ : The number of iterations exceeded a prescribed maximum number of iterations $i_t$.

Actually, the implemented stopping criteria in Demmel et al. [60] are more sophisticated because they track both the normwise and componentwise convergence. It allows them to stop the algorithm once the solution does not improve in both the normwise and componentwise sense.

The way we set $\rho_t < 1$ and $i_t$ defines how "cautious" or "aggressive" we are. The more we are aggressive (i.e., $\rho_t$ and $i_t$ high), the more we allow the algorithm to use resources to solve a difficult problem. The risk is to put much computational effort before stopping on a problem that will not converge. On the other hand, the more we are cautious (i.e., $\rho_t$ and $i_t$ small), the quicker we will break if the convergence is difficult. While this is the optimal configuration to avoid wasting computational resources, the algorithm might break on a problem where convergence is actually possible.

## 4.7 Scaling

We have seen that using low precision for the factorization with LU-IR3, LU-GMRES-IR3, AQR-IR3 and AQR-GMRES-IR3 restricts their respective conditions for convergence; in particular, the conditions of LU-IR3 and AQR-IR3 become very restrictive. However, we face additional difficulties when dealing with low precision factorization, namely the underflows and overflows. These phenomena, described in section 2.1.1, occur when an entry is outside the range of representation of the low precision arithmetic. In the case of overflow, the entry is rounded to $\pm$inf. In the case of underflow, the entry is rounded to 0. If we suppose that the factorization is done in a low precision $u_f$, we can underflow or overflow when the original matrix $A$ is cast to precision $u_f$ and when we compute the entries of the factors. For instance, in the case of fp16, any number approximately outside of the interval $[10^{-5}, 10^5]$ (subnormal numbers not considered) will overflow or underflow (see Table 2.1). Unfortunately, many matrices from real-life or industrial applications in this manuscript have entries outside this range. This problem is getting even worse with lower precision, such as fp8.

Underflows can cause a severe loss of information, and the cast of nonzero entries to zeros can even transform a nonsingular matrix into a structurally singular one. Overflows are even worse because a factorization cannot produce useful results for a matrix with infinities among the entries. In this context, Higham et al. [121] proposed a scaling algorithm to avoid the overflows entirely and limit as much as possible the underflows when casting the matrix $A$ to precision $u_f$. The strategy is as follows. We first apply a two-sided diagonal scaling which guarantees that the largest entry in absolute value is no more than 1. Second, we multiply the matrix by a scalar $\lambda$ which shifts the entries of the matrix closer to the maximum representable value of $u_f$ that we note $y_{\max}^{(f)}$. This strategy is described in Algorithm 4.7.

Hence, in Algorithm 4.7, $RAS$ is $A$ but with its entries balanced such that $\max_{i,j} |(RAS)_{i,j}| \leq 1$. We define $\lambda = \theta\, y_{\max}^{(f)}/\beta$ which scales the entries of $RAS$ nearer to $y_{\max}^{(f)}$; $\lambda$ depends on the parameter $\theta \in (0, 1]$ which defines how close we map the highest entry of $RAS$ in absolute value to $y_{\max}^{(f)}$, so, if $\theta = 1$, the highest entry in absolute value is $y_{\max}^{(f)}$. The two-sided scaling avoids the overflows while the scaling by $\lambda$ reduces the underflows by increasing the size

**Algorithm 4.7** Squeezing a matrix to precision $u_f$

**Input:** an $n \times n$ matrix $A$ and a parameter $\theta \in (0,1]$.
**Output:** a scaled and rounded matrix $A^{(f)}$ of $A$ in precision $u_f$.

TWO-SIDED DIAGONAL SCALING:
  **for** $i = 1, \ldots, n$ **do**
    $r_i = \|A_{i,1:n}\|_\infty^{-1}$
  **end for**
  $R = \text{diag}(r)$
  $\widetilde{A} = RA$
  **for** $j = 1, \ldots, n$ **do**
    $s_j = \|\widetilde{A}_{1:n,j}\|_\infty^{-1}$
  **end for**
  $S = \text{diag}(s)$
SCALE AND ROUND:
  Let $\beta$ be the maximum magnitude of any entry of $RAS$
  $\lambda = \theta\, y_{\max}^{(f)}/\beta$
  $A^{(f)} = \text{fl}_f(\lambda(RAS))$

of the entries in absolute value.

While choosing the highest $\lambda$ would prevent as much as possible the underflows during the cast of $A$ in precision $u_f$, it can also be dangerous to choose it too high when we need to proceed to the factorization after. Indeed, if we consider the LU factorization, the entries in the factors can get larger in magnitude than $A$. This increase is quantified by the growth factor, which can be quite large even with a stable pivoting strategy such as partial pivoting (see section 2.2.1.2). Therefore, with a too high $\lambda$, there is a high risk of overflow during the factorization and, so, we should choose it small enough to guarantee that the entries of the factors do not exceed $y_{\max}^{(f)}$.

## 4.8 Summary

The previous iterative refinement variants presented (i.e., LU-IR3, LU-GMRES-IR3, AQR-IR3, and AQR-GMRES-IR3) are all built around a direct solver (either an LU or QR one). They are the foundations of the work developed in chapters 5 and 6. We summarize in Table 4.2 their respective convergence conditions for the forward and backward errors. If these conditions are met, the solution of (1.1) is guaranteed to reach the corresponding limiting accuracies listed in Table 4.3.

We listed in section 3.6 four different ways to use iterative refinement. We now clarify for which setting of precisions $u$, $u_s$, and $u_r$ in Algorithm 4.1 we target these different uses:

- 1: $u_r \leq u = u_s$. Get a better forward error without dependence on the condition number.
- 2: $u = u_s = u_r$. Recover backward stability when the solver at step 4 is unstable. This property is also valid for the previous setting.

Table 4.2: Summary of the different convergence conditions on the forward and normwise backward errors for the previously reviewed iterative refinement algorithms.

| Algorithm | Forward error | Backward error |
|---|---|---|
| Generalized | $2u_s \min(\text{cond}(A), \kappa(A)\mu_i) + u_s\|E_i\| \ll 1$ | $u_s(c_1\kappa(A)+c_2) \ll 1$ |
| LU-IR3 | $u_f\kappa(A) \ll 1$ | $u_f\kappa(A) \ll 1$ |
| LU-GMRES-IR3 | $u_f u^{1/2}\kappa(A) \ll 1$ | $u_f^{1/2}u^{1/2}\kappa(A) \ll 1$ |
| AQR-IR3 | $u_f\kappa(A_+) \ll 1$ | $u_f\kappa(A_+) \ll 1$ |
| AQR-GMRES-IR3 | $u_f u^{1/2}\kappa(A) \ll 1^{\S}$ | $u_f^{1/2}u^{1/2}\kappa(A) \ll 1^{\S}$ |

§ scaling (4.34) is applied

Table 4.3: Summary of the limiting accuracies for the forward and normwise backward errors. The $\text{cond}(A,x)$ term becomes $\text{cond}(A_+,z)$ in the least squares case.

| Forward error | Backward error |
|---|---|
| $p\,u_r\,\text{cond}(A,x)+u$ | $p\,u_r+u$ |

- 3: $u = u_r \leq u_s$. Accelerate the solution of linear systems with low accuracy solution on the correction equation (4.2).
- 4: $u_r \leq u \leq u_s$. All previous strategies combined. Stability, performance, and high accuracy can all be achieved at the same time with this setting.

# 5 LU-GMRES-IR in five precisions

LU preconditioned GMRES-based iterative refinement in three precisions (LU-GMRES-IR3), defined by Algorithm 4.3 and discussed broadly in section 4.4, is an algorithm aiming at accelerating the solution of a square linear system (1.1) without compromising numerical stability or robustness. However, the three precisions formulation of the algorithm has major practical limitations for the solution of dense and sparse systems.

In this chapter, we discuss a five precisions version of this algorithm that we call LU preconditioned GMRES-based iterative refinement in five precisions (LU-GMRES-IR5), which overcomes these limitations. We present these limitations and motivate the need for relaxing the requirements on the precisions in LU-GMRES-IR3 in section 5.1. We propose a rounding error analysis for LU-GMRES-IR5 that uses a new result on the backward stability of MGS-GMRES in two precisions in section 5.2. From this rounding error analysis, we can identify a relatively small subset of relevant combinations of precisions that achieve different levels of trade-off between cost and robustness in section 5.3. We discuss the use of a stopping criterion inside GMRES and how it affects the previous error analysis in section 5.4. We conduct numerical experiments on both random dense matrices and real-life sparse matrices from a wide range of applications that assess our theoretical findings in section 5.5. We provide practical advice for choosing variants of iterative refinement based on LU solver in section 5.6. Finally, we extend LU-GMRES-IR5, for the solution of square linear systems, to the least squares problems in section 5.7.

## 5.1 From LU-GMRES-IR3 to LU-GMRES-IR5

Modern hardware increasingly supports low precision floating-point arithmetics that provide unprecedented speed, communication, and energy benefits as explained in section 2.1.3. However, using these low precisions might badly affect the robustness of the iterative refinement algorithms by introducing unrecoverable rounding errors.

In particular, LU-IR3, introduced in section 4.3, is very sensitive to the condition number of the linear system. Indeed, while it can be very attractive for well-conditioned matrices, the forward error is only guaranteed to converge when $\kappa(A)u_f \ll 1$ (4.24). This condition can be quite restrictive, especially when low precision arithmetics are used for the factorization. For example, the condition becomes $\kappa(A) \ll 2 \times 10^3$ with IEEE fp16 (half) precision, and $\kappa(A) \ll 3 \times 10^2$ with bfloat16.

The purpose of LU-GMRES-IR3, introduced in section 4.4, is to recover this robustness when low precision factorizations are targeted. By applying the GMRES solver entirely in working precision $u$ except the preconditioned matrix–vector product which is applied in the higher precision $u^2$, LU-GMRES-IR3 has a less restrictive convergence condition for the forward error $\kappa(A)^2 u_f^2 u \ll 1$ (4.27) and can handle larger condition numbers. For example, if $u$ is set to IEEE fp64 (double) precision, the condition becomes $\kappa(A) \ll 2 \times 10^{11}$ with $u_f$ set to IEEE fp16 (half) precision, and $\kappa(A) \ll 2 \times 10^{10}$ with bfloat16.

However, the requirement that the preconditioner must be applied in precision $u^2$ is a practical limitation because it can be expensive. It is particularly inconvenient if the target accuracy $u$ is fp64 as it requires applying the preconditioner in fp128, which is a very slow arithmetic as stated in section 2.1.2. In fact, practical implementations of LU-GMRES-IR3, as developed by Haidar et al. [101; 103; 102] and implemented in the MAGMA library [155] and the NVIDIA cuSOLVER library [168], have relaxed this requirement by applying the preconditioner in double rather than quadruple precision, even though the error analysis of Carson and Higham [45] (Theorem 4.4) does not cover this case. GMRES-IR variants for symmetric positive definite systems and least squares problems have also used only two precisions (see Carson et al. [47], Higham and Pranesh [120]).

This is why in this chapter, we are particularly interested in addressing the question of whether we can use a lower precision to apply the preconditioner within GMRES and still obtain a LU preconditioned GMRES-based iterative refinement (LU-GMRES-IR) solver able to handle more ill-conditioned matrices than LU-based iterative refinement (LU-IR).

These practical constraints and theoretical questions lead us to propose new variants of LU preconditioned GMRES-based iterative refinement with relaxed requirements on the precisions used within the GMRES solver. We allow the preconditioner (the LU factors) to be applied in an arbitrary precision $u_p$, with $u_p \geq u^2$. We also allow the rest of the GMRES computations to be performed in an arbitrary precision $u_g$, with $u_g \geq u$. We obtain Algorithm 5.1, which has up to *five* independent precisions in play and which we thus call *LU preconditioned GMRES-based iterative refinement in five precisions (LU-GMRES-IR5)*.

Note that even though LU-GMRES-IR5 has five precision parameters, it does not mean that this algorithm uses five different arithmetics simultaneously since some of the precisions can be equal. In fact, we will see that most of the relevant combinations of precisions use only two or three different precisions. Nevertheless, there exist some meaningful variants (as defined in section 2.1.3.3) that employ four or even five different precisions in the same refinement step. We specify "in the same refinement step" because dynamically changing the precisions from one iteration to another could already lead to such a number of different precisions across all steps, and even more in the case of arbitrary precision refinement, which has been studied by Kiełbasiński [134], Lee et al. [139].

**Algorithm 5.1** LU-GMRES-IR5

**Input:** an $n \times n$ matrix $A$ and a right-hand side $b$.
**Output:** an approximate solution to $Ax = b$.

1: Factorize $A = \widehat{L}\widehat{U}$. $(u_f)$
2: Solve $\widehat{L}\widehat{U}x_0 = b$ for $x_0$. $(u_f)$
3: **while not** converged **do**
4: Compute $r_i = b - Ax_i$. $(u_r)$
5: Solve $\widehat{U}^{-1}\widehat{L}^{-1}Ad_i = \widehat{U}^{-1}\widehat{L}^{-1}r_i$ by GMRES at precision $u_g$ with matrix–vector products with $\widetilde{A} = \widehat{U}^{-1}\widehat{L}^{-1}A$ computed at precision $u_p$.
6: Compute $x_{i+1} = x_i + d_i$. $(u)$
7: **end while**

## 5.2 Rounding error analysis

To guarantee the backward stability of LU-GMRES-IR5 and to capture the role of each precision $u, u_f, u_r, u_g$ and $u_p$ on the convergence and the quality of the computed solution, we need to carry out a new rounding error analysis. Specifically, we want the equivalent of Theorem 4.4 on LU-GMRES-IR3 but for our new LU-GMRES-IR5. Therefore, we wish to show that the forward and the normwise backward errors are guaranteed to decrease until they reach a certain size, namely, their limiting accuracies, if some convergence conditions are met. We will informally refer to the attainment of this level as "convergence" while recognizing that the error does not necessarily converge in the formal sense.

Determining the limiting accuracies of LU-GMRES-IR5 is straightforward. Because regardless of the solver used to solve the correction equation at step 4 of Algorithm 4.1, generalized iterative refinement guarantees that the limiting accuracies will be (4.10) and (4.22), respectively, for the forward and backward errors. Thus, our main concern is about determining the convergence conditions of LU-GMRES-IR5 that cannot be directly obtained from the analysis of LU-GMRES-IR3. It is mainly because, under the assumption that $\kappa(A)u \leq 1$, the application of the preconditioner in precision $u^2$ allows for important simplifications in Carson and Higham [44; 45] of the form $\kappa(A)u^2 \leq u$ that we cannot rely on anymore.

We present in Theorem 5.1 the result of our error analysis. The form of this new theorem is similar to Theorem 4.4 and, in particular, it expresses the convergence conditions (5.2) and (5.3) of LU-GMRES-IR5 with the precision $u_f$ and the new precisions $u_g$ and $u_p$. Therefore, the rest of this section will be about proving Theorem 5.1.

**Theorem 5.1** (Convergence of LU-GMRES-IR5)**.** *Let* (1.1) *be solved by LU-GMRES-IR5 (Algorithm 5.1) using MGS-GMRES at step 5 and using GEPP at step 1. If* $u_g \geq u$ *and* $\kappa(A)u_p < 1$, *the forward and backward errors will reach their respective limiting accuracies*

$$p\,u_r\,\mathrm{cond}(A,x) + u \text{ (forward)} \qquad \text{and} \qquad p\,u_r + u \text{ (backward)}, \tag{5.1}$$

*provided that*

$$(u_g + u_p\kappa(A))(1 + \kappa(A)^2 u_f^2) \ll 1 \text{ (forward)} \tag{5.2}$$

*and*

$$(u_g + u_p\kappa(A))(1+\kappa(A)u_f)\kappa(A) \ll 1 \text{ (backward).} \tag{5.3}$$

To generalize the error analysis of LU-GMRES-IR3 to LU-GMRES-IR5 we will proceed as follows. First, we extend the analysis of Paige et al. [175] on the backward stability of MGS-GMRES to arbitrary matrix–vector products satisfying a generic error bound in section 5.2.1. Second, we use this generalized analysis to bound the forward and backward errors of the two precisions MGS-GMRES solver used at step 5 for the solution of $\widetilde{A}d_i = s_i$ in section 5.2.2, where $\widetilde{A} = \widehat{U}^{-1}\widehat{L}^{-1}A$ and $s_i = \widehat{U}^{-1}\widehat{L}^{-1}r_i$. We then use these bounds to rewrite (4.9) and (4.16) and to apply Theorems 4.1 and 4.2 to obtain the specialized conditions on $\kappa(A)$ for LU-GMRES-IR5 to converge in section 5.2.3. Finally, in section 5.2.4, we comment on the results of Theorem 5.1.

Our analysis makes use of the following two assumptions on the precisions:

- $u_g \geq u$: since the solution computed by GMRES is stored in the working precision $u$, we do not expect that running GMRES in precision $u_g < u$ would give a significant benefit. In particular, this assumption guarantees that condition (4.7) is met.

- $\kappa(A)u < 1$: this assumption is already present in the three-precision analysis (see section 4.4); here, we use it for different discussions, but it is not required to derive Theorem 5.1.

- $\kappa(A)u_p < 1$: this assumption is needed to drop certain second order terms in our analysis. It guarantees that $\kappa(A)^2 u_p^2$ is negligible relative to $\kappa(A)u_p$.

### 5.2.1 Error analysis of MGS-GMRES with arbitrary matrix–vector products

We assume that the MGS-GMRES variant of GMRES, represented by Algorithm 2.6, is used in LU-GMRES-IR5. Our aim in this section is to bound the backward error of the two-precision MGS-GMRES used for the solution of the preconditioned system $\widetilde{A}d_i = s_i$. In this algorithm, the precision $u_g$ is used for all the operations except the products with $\widetilde{A}$, which are computed in precision $u_p$. The analysis of Paige et al. [175] is for fixed precision, unpreconditioned MGS-GMRES, and therefore is not directly applicable. Note that, in the analysis of Carson and Higham [44], the preconditioner is applied in precision $u^2$ and, by using the assumption $\kappa(A)u < 1$, it is shown that the products with $\widetilde{A}$ in precision $u^2$ are at least as accurate as the products with $A$ in precision $u$, so that the backward stability result of Paige et al. [175] still holds. In our case, the same argument does not apply, so we must generalize the backward stability result [175, Eq. (8.15)] to the case of arbitrary matrix–vector products satisfying a generic error bound. We will then be able to use this analysis to derive results for the two-precision MGS-GMRES used in LU-GMRES-IR5. We state the conclusion of our analysis in the next theorem.

**Theorem 5.2.** *Consider the solution of a linear system*

$$Px = q, \quad P \in \mathbb{R}^{n\times n}, \quad 0 \neq q \in \mathbb{R}^n \tag{5.4}$$

*with an MGS-GMRES solver carrying out its operations in precision $u_g$, except for the products with $P$, which satisfy instead*

$$\mathrm{fl}(Pv) = Pv + f, \quad \|f\|_2 \lesssim \epsilon_p \|P\|_F \|v\|_2, \tag{5.5}$$

*where $\epsilon_p > 0$ is a parameter quantifying the stability of the matrix–vector products. Provided that*

$$\sigma_{\min}(P) \gtrsim (k^{1/2}\epsilon_p + \tilde{\gamma}_{kn}^g)\|P\|_F, \tag{5.6}$$

*there is a step $k \leq n$ such that the algorithm produces a computed $\widehat{x}_k$ satisfying*

$$(P + \Delta P)\widehat{x}_k = q + \Delta q, \tag{5.7a}$$
$$\|\Delta P\|_F \lesssim (k^{1/2}\epsilon_p + \tilde{\gamma}_{kn}^g)\|P\|_F, \tag{5.7b}$$
$$\|\Delta q\|_2 \lesssim \tilde{\gamma}_{kn}^g \|q\|_2. \tag{5.7c}$$

*Proof.* The proof of Theorem 5.2 relies on the analysis of Paige et al. [175] and, more precisely, on [175, sec. 8] and [175, Eq. (4.3)] therein. For the sake of readability, we will not attempt to make this proof self-contained in this chapter but, rather, we will highlight the differences with the analysis of Paige et al. [175] and refer the reader to the Appendix 8.2 for the full details. The original notation has been slightly adapted to be consistent with the notation of this article inherited from Carson and Higham [44; 45].

In our version of MGS-GMRES we consider a product with $P$ satisfying (5.5). We now show that considering (5.5) with $\epsilon_p \neq \gamma_n^g$ mainly changes [175, Eq. (4.3)]. Let us consider $\widehat{V}_k = [\widehat{v}_1, \ldots, \widehat{v}_k] \in \mathbb{R}^{n\times k}$, the matrix of computed basis vectors, and $\dot{V}_k = [\dot{v}_1, \ldots, \dot{v}_k]$ the same matrix but with its columns correctly normalized; that is, for $j \leq k$,

$$\widehat{v}_j = \dot{v}_j + \Delta v_j^{(1)}, \quad \|\Delta v_j^{(1)}\|_2 \leq \tilde{\gamma}_n^g, \tag{5.8a}$$
$$\widehat{V}_k = \dot{V}_k + \Delta V_k^{(1)}, \quad \Delta V_k^{(1)} = [\Delta v_1^{(1)}, \ldots, \Delta v_k^{(1)}], \tag{5.8b}$$

where $\Delta v_j^{(1)}$ is the error for the normalization of $\widehat{v}_j$ and $\Delta V_k^{(1)}$ is the accumulated error for the normalization of the basis at step $k$. By (5.5) and (5.8), we obtain

$$\mathrm{fl}(P\widehat{v}_j) = P(\dot{v}_j + \Delta v_j^{(1)}) + f_j \tag{5.9a}$$
$$= P\dot{v}_j + \Delta v_j^{(2)}, \tag{5.9b}$$

where $\Delta v_j^{(2)} = P\Delta v_j^{(1)} + f_j$ satisfies $\|\Delta v_j^{(2)}\|_2 \lesssim (\epsilon_p + \tilde{\gamma}_n^g)\|P\|_F$ since $\|\dot{v}_j\|_2 = 1$ and $\|f_j\|_2 \lesssim \epsilon_p \|P\|_F \|\dot{v}_j + \Delta v_j^{(1)}\|_2$. We therefore obtain

$$\mathrm{fl}(P\widehat{V}_k) = P\dot{V}_k + \Delta V_k^{(2)}, \quad \|\Delta V_k^{(2)}\|_F \lesssim k^{1/2}(\epsilon_p + \tilde{\gamma}_n^g)\|P\|_F, \tag{5.10}$$

where $\Delta V_k^{(2)}$ contains the error for both the product and the normalization at the $k$th iteration. Equation (5.10) is our new version of [175, Eq. (4.3)]; adapting the remainder of [175, sect. 8] to take this change into account is straightforward. Consequently, we show that at the $(\bar{m}-1)$st iteration, MGS-GMRES has computed a backward stable solution of the

system, where $\bar{m}$ satisfies [175, Eq. (6.1)]. From now on, we set $k$ such that $k \equiv \bar{m} - 1 \leq n$, and rewrite [175, Eq. (8.2)] as

$$r_k(\widehat{y}_k) \equiv q_k - P_k \widehat{y}_k, \quad q_k \equiv q + \Delta q_k(\widehat{y}_k), \quad P_k \equiv P\dot{V}_k + \Delta V_k^{(3)}(\widehat{y}_k), \tag{5.11a}$$

$$\|\Delta q_k(\widehat{y}_k)\|_2 \leq \widetilde{\gamma}_{kn}^g \|q\|_2, \quad \Delta V_k^{(3)}(y) \equiv \Delta V_k^{(2)} + \Delta C_k(y), \tag{5.11b}$$

$$\|\Delta V_k^{(3)}\|_F \lesssim (k^{1/2}\epsilon_p + \widetilde{\gamma}_{kn}^g)\|P\|_F, \tag{5.11c}$$

where $\Delta C_k(y)$ and $\Delta q_k(y)$ are the errors in the MGS least squares solution [175, sect. 7], and where $\widehat{y}_k$ is the computed least squares solution at the $k$th iteration

$$\widehat{y}_k = \arg\min_y \|q - P\dot{V}_k y\|_2, \quad k < \bar{m}. \tag{5.12}$$

Using the scaling invariance of MGS to scale the right-hand side $q_k$ by some scalar $\phi'$ and making use of [175, Thm. 2.4] gives a bound on the residual [175, Eq. (8.9)]

$$\|r_k(\hat{y}_k)\|_2^2 \leq (\widetilde{\gamma}_{kn}^g)^2(\|q_k\phi'\|_2^2 + \|P_k\|_F^2)2(\phi')^{-2}. \tag{5.13}$$

In addition to bounding $\|P_k\|_F^2$ and $\|q_k\phi'\|_2^2$, we use the nonsingularity condition (5.6) (which is an upper-bound of the condition number $\|P\|_F\|P^{-1}\|_2$ from Demmel [57]) in the same fashion as [175, Eq. (8.11)] to compute a bound for $(\phi')^{-2}$, which allows us to rewrite [175, Eq. (8.12)] as

$$\|r_k(\hat{y}_k)\|_2 \lesssim \widetilde{\gamma}_{kn}^g(\|P\|_F\|\hat{y}_k\|_2 + \|q\|_2). \tag{5.14}$$

Since $\epsilon_p$ appears in second order terms and higher, they have been dropped, making (5.14) equivalent to [175, Eq. (8.12)]. Considering now $\widehat{x}_k = \mathrm{fl}(\widehat{V}_k\widehat{y}_k) = (\widehat{V}_k + \Delta V_k^{(4)})\widehat{y}_k$ and using a standard matrix–vector product in precision $u_g$ satisfying $\|\Delta V_k^{(4)}\|_F \leq \widetilde{\gamma}_k^g\|\widehat{V}_k\|_F$ and $\Delta P_k \equiv [\Delta V_k^{(3)}(\widehat{y}_k) - P(\Delta V_k^{(4)} + \widehat{V}_k - \dot{V}_k)]\widehat{y}_k \frac{\widehat{x}_k^T}{\|\widehat{x}_k\|_2^2}$, we can rewrite [175, Eq. (8.15)] as

$$r_k(\widehat{y}_k) = q + \Delta q_k(\widehat{y}_k) - (P + \Delta P_k)\widehat{x}_k, \tag{5.15a}$$

$$\|r_k(\widehat{y}_k)\|_2 \lesssim \widetilde{\gamma}_{kn}^g(\|P\|_F\|\widehat{x}_k\|_2 + \|q\|_2), \tag{5.15b}$$

$$\|\Delta q_k(\widehat{y}_k)\|_2 \leq \widetilde{\gamma}_{kn}^g\|q\|_2, \tag{5.15c}$$

$$\|\Delta P_k\|_F \lesssim (k^{1/2}\epsilon_p + \widetilde{\gamma}_{kn}^g)\|P\|_F. \tag{5.15d}$$

This leads to (5.7) and completes the proof of the theorem. □

In the original analysis, $Pv$ is a standard matrix–vector product operation and $u_p = u_g$, so $\mathrm{fl}(Pv) = (P + \Delta P)v$ where $\|\Delta P\|_F \leq \gamma_q^g\|P\|_F$ from Theorem 2.1. In this case we can set $f = \Delta P v$ and apply Theorem 5.2 with $\epsilon_p = \gamma_q^g$, recovering the result of Paige et al. [175] for an unpreconditioned MGS-GMRES in uniform precision, which produces a solution $\widehat{x}_k$ of the system $Px = q$ satisfying

$$(P + \Delta P)\widehat{x}_k = q + \Delta q, \quad \|\Delta P\|_F \lesssim \widetilde{\gamma}_{kn}^g\|P\|_F, \quad \|\Delta q\|_2 \lesssim \widetilde{\gamma}_{kn}^g\|q\|_2. \tag{5.16}$$

### 5.2.2 Error analysis of LU-GMRES-IR5 with general $\mathbf{u_g}$ and $\mathbf{u_p}$ precisions

We proceed in three steps. First, we bound the error in $\widehat{s}_i = \mathrm{fl}(\widehat{U}^{-1}\,\mathrm{fl}(\widehat{L}^{-1}\widehat{r}_i))$. Second, we use our analysis of MGS-GMRES of the previous section to prove the backward stability of the solution to the system $\widetilde{A}d_i = \widehat{s}_i$. Third, we combine the previous two results to derive bounds of the type (4.4)–(4.5) for the solution of $\widetilde{A}d_i = s_i$.

We begin by bounding the error introduced in forming the preconditioned right-hand side $s_i = \widehat{U}^{-1}\widehat{L}^{-1}\widehat{r}_i$ in precision $u_p$. The computed $\widehat{s}_i$ is the result of the applications of two triangular solves whose rounding errors are covered by Theorem 2.3. It satisfies

$$(\widehat{L}+\Delta L)(\widehat{U}+\Delta U)\widehat{s}_i = \widehat{r}_i, \quad |\Delta L| \le \gamma_n^p|\widehat{L}|, \quad |\Delta U| \le \gamma_n^p|\widehat{U}|. \tag{5.17}$$

Also, considering Theorem 2.2, the LU factors computed at precision $u_f$ satisfy

$$\widehat{L}\widehat{U} = A + \Delta A^{(1)}, \quad |\Delta A^{(1)}| \le \gamma_n^f|\widehat{L}||\widehat{U}|. \tag{5.18}$$

Note that, technically, $\widehat{L}$ and $\widehat{U}$ are not the computed factors of $A$, but the ones of $A$ cast in precision $u_f$. As the condition number of the cast matrix can be substantially lower due to a regularization effect (see [128]), $\kappa_\infty(A)$ is sometimes an overestimate in the following analysis. We have

$$\begin{aligned} s_i - \widehat{s}_i &= \widehat{U}^{-1}\widehat{L}^{-1}(\Delta L\widehat{U} + \widehat{L}\Delta U + \Delta L\Delta U)\widehat{s}_i && (5.19a)\\ &= (A+\Delta A^{(1)})^{-1}(\Delta L\widehat{U} + \widehat{L}\Delta U + \Delta L\Delta U)\widehat{s}_i && (5.19b)\\ &\approx (A^{-1} - A^{-1}\Delta A^{(1)}A^{-1})(\Delta L\widehat{U} + \widehat{L}\Delta U + \Delta L\Delta U)\widehat{s}_i && (5.19c) \end{aligned}$$

and dropping second order terms we obtain

$$\begin{aligned} \|s_i - \widehat{s}_i\|_\infty &\lesssim \gamma_{2n}^p \||A^{-1}||\widehat{L}||\widehat{U}|\|_\infty \|\widehat{s}_i\|_\infty && (5.20a)\\ &\le n^2\rho_n\widetilde{\gamma}_n^p\kappa_\infty(A)\|\widehat{s}_i\|_\infty \lesssim n^2\rho_n\widetilde{\gamma}_n^p\kappa_\infty(A)\|s_i\|_\infty. && (5.20b) \end{aligned}$$

where the second inequality comes from Theorem 2.5. Actually, in the previous bound, certain terms of order $\kappa(A)^2 u_p u_f$ are dropped (as second order terms). While our assumptions allow $\kappa(A)u_f$ to be arbitrarily large and these terms to not be necessarily negligible, we observe in practice that $\kappa(A)u_f$ is generally at most of order a constant in these terms. This appears to be the result of the regularization effect mentioned earlier.

Next, we show that this new version of GMRES (with general $u_p$ and $u_g$ precisions) provides a backward stable solution to the system $\widetilde{A}d_i = s_i$, where $\widetilde{A} = \widehat{U}^{-1}\widehat{L}^{-1}A$ and $s_i = \widehat{U}^{-1}\widehat{L}^{-1}r_i$. We rely on Theorem 5.2, which provides backward error bounds for MGS-GMRES with general matrix–vector products. Our aim is therefore to prove that (5.5) holds for some $\epsilon_p$ when the matrix–vector products are computed with matrix $\widetilde{A} = \widehat{U}^{-1}\widehat{L}^{-1}A$ and in precision $u_p$. Let $z_j = \widetilde{A}\widehat{v}_j$ be computed in precision $u_p$ by a matrix product followed by two triangular solves. Then

$$(A+\Delta A^{(2)})\widehat{v}_j = \widehat{w}_j, \quad |\Delta A^{(2)}| \le \gamma_q^p|A|, \tag{5.21a}$$

$$(\widehat{L}+\Delta L)\widehat{y}_j = \widehat{w}_j, \quad |\Delta L| \le \gamma_n^p |\widehat{L}|, \tag{5.21b}$$
$$(\widehat{U}+\Delta U)\widehat{z}_j = \widehat{y}_j, \quad |\Delta U| \le \gamma_n^p |\widehat{U}|. \tag{5.21c}$$

The computed vector $\widehat{z}_j$ can therefore be written as

$$\widehat{z}_j = (\widehat{U}+\Delta U)^{-1}(\widehat{L}+\Delta L)^{-1}(A+\Delta A^{(2)})\widehat{v}_j \tag{5.22a}$$
$$\approx (\widehat{U}^{-1}-\widehat{U}^{-1}\Delta U\widehat{U}^{-1})(\widehat{L}^{-1}-\widehat{L}^{-1}\Delta L\widehat{L}^{-1})(A+\Delta A^{(2)})\widehat{v}_j \tag{5.22b}$$
$$= \widetilde{A}\widehat{v}_j + f_j, \tag{5.22c}$$

where

$$f_j \approx (\widehat{U}^{-1}\widehat{L}^{-1}\Delta A^{(2)} - \widehat{U}^{-1}\widehat{L}^{-1}\Delta L\widehat{L}^{-1}A - \widehat{U}^{-1}\Delta U\widehat{U}^{-1}\widehat{L}^{-1}A)\widehat{v}_j \tag{5.23a}$$
$$= (\widetilde{A}A^{-1}\Delta A^{(2)} - \widehat{U}^{-1}\widehat{L}^{-1}\Delta L\widehat{U}\widetilde{A} - \widehat{U}^{-1}\Delta U\widetilde{A})\widehat{v}_j, \tag{5.23b}$$

and so

$$\|f_j\|_2 \lesssim \gamma_n^p(\kappa_F(A)+\kappa_F(\widehat{U})\kappa_F(\widehat{L})+\kappa_F(\widehat{U}))\|\widetilde{A}\|_F\|\widehat{v}_j\|_2. \tag{5.24}$$

Since we can expect $\kappa_F(\widehat{L})$ to be of modest size (because $\widehat{L}$ is a unit triangular matrix with off-diagonal elements bounded by 1 if, for example, partial pivoting is used in the factorization), and since $\kappa_F(\widehat{U}) \lesssim \kappa_F(A)\kappa_F(\widehat{L})$, we obtain

$$\|f_j\|_2 \lesssim \widetilde{\gamma}_n^p \kappa_F(A)\|\widetilde{A}\|_F\|\widehat{v}_j\|_2 \le n\widetilde{\gamma}_n^p \kappa_\infty(A)\|\widetilde{A}\|_F\|\widehat{v}_j\|_2. \tag{5.25}$$

Condition (5.5) is thus satisfied for $\epsilon_p = n\widetilde{\gamma}_n^p\kappa_\infty(A)$, and Theorem 5.2 is therefore applicable. From (5.7) we obtain

$$(\widetilde{A}+\Delta\widetilde{A})\widehat{d}_i = \widehat{s}_i + \Delta\widehat{s}_i, \tag{5.26a}$$
$$\|\Delta\widetilde{A}\|_F \lesssim (\widetilde{\gamma}_{kn}^g + n\widetilde{\gamma}_{k^{1/2}n}^p \kappa_\infty(A))\|\widetilde{A}\|_F, \tag{5.26b}$$
$$\|\Delta\widehat{s}_i\|_2 \lesssim \widetilde{\gamma}_{kn}^g\|\widehat{s}_i\|_2 \lesssim n^{1/2}\widetilde{\gamma}_{kn}^g\|s_i\|_\infty. \tag{5.26c}$$

Rewriting (5.26a) as

$$s_i - \widetilde{A}\widehat{d}_i = \Delta\widetilde{A}\widehat{d}_i - (\widehat{s}_i - s_i) - \Delta\widehat{s}_i \tag{5.27}$$

and using (5.26b), (5.20), and (5.26c) to bound the three terms on the right-hand side, we obtain

$$\|s_i - \widetilde{A}\widehat{d}_i\|_\infty \le \|\Delta\widetilde{A}\|_\infty\|\widehat{d}_i\|_\infty + \|\widehat{s}_i - s_i\|_\infty + \|\Delta\widehat{s}_i\|_\infty \tag{5.28a}$$
$$\lesssim n(\widetilde{\gamma}_{kn}^g + n\widetilde{\gamma}_{k^{1/2}n}^p \kappa_\infty(A))\|\widetilde{A}\|_\infty\|\widehat{d}_i\|_\infty + n^2\rho_n\widetilde{\gamma}_n^p\kappa_\infty(A)\|s_i\|_\infty + n^{1/2}\widetilde{\gamma}_{kn}^g\|s_i\|_\infty \tag{5.28b}$$
$$\le n^3 \max(k^{1/2},\rho_n)(u_g + u_p\kappa_\infty(A))(\|\widetilde{A}\|_\infty\|\widehat{d}_i\|_\infty + \|s_i\|_\infty). \tag{5.28c}$$

In conclusion, the normwise relative backward error of the system $\widetilde{A}\widehat{d}_i = s_i$ is bounded by

$$\frac{\|s_i - \widetilde{A}\widehat{d}_i\|_\infty}{\|\widetilde{A}\|_\infty\|\widehat{d}_i\|_\infty + \|s_i\|_\infty} \lesssim f(n,k,\rho_n)(u_g + u_p\kappa_\infty(A)) \tag{5.29}$$

and the relative error of the computed $\widehat{d}_i$ therefore satisfies

$$\frac{\|d_i - \widehat{d}_i\|_\infty}{\|d_i\|_\infty} \lesssim f(n,k,\rho_n)(u_g + u_p\kappa_\infty(A))\kappa_\infty(\widetilde{A}), \tag{5.30}$$

where $f(n,k,\rho_n) = n^3 \max(k^{1/2},\rho_n)$.

### 5.2.3 Convergence conditions on $\kappa(\mathbf{A})$

We can now use this analysis together with (4.9) and (4.16) of Theorems 4.1 and 4.2 to determine sufficient conditions for the convergence of the forward and backward errors for general $u_p$ and $u_g$ parameters.

Beginning with the forward error, (5.30) shows that (4.4) holds with

$$u_s\|E_i\|_\infty \equiv f(n,k,\rho_n)(u_g + u_p\kappa_\infty(A))\kappa_\infty(\widetilde{A}), \tag{5.31}$$

and so, dropping constants, the convergence condition of Theorem 4.1 becomes

$$(u_g + u_p\kappa(A))\kappa(\widetilde{A}) \ll 1. \tag{5.32}$$

It remains to express $\kappa(\widetilde{A})$ in terms of $\kappa(A)$. By using the formulation of the inverse of a sum of matrices by Henderson and Searle [104] and (5.18), we obtain

$$\widetilde{A} = \widehat{U}^{-1}\widehat{L}^{-1}A = (A + \Delta A^{(1)})^{-1}A \approx I - A^{-1}\Delta A^{(1)}, \tag{5.33a}$$
$$\widetilde{A}^{-1} = A^{-1}\widehat{L}\widehat{U} = A^{-1}(A + \Delta A^{(1)}) = I + A^{-1}\Delta A^{(1)}, \tag{5.33b}$$

which gives the bounds

$$\|\widetilde{A}\|_\infty \lesssim 1 + \gamma_n^f \||A^{-1}||\widehat{L}||\widehat{U}|\|_\infty, \tag{5.34a}$$
$$\|\widetilde{A}^{-1}\|_\infty \leq 1 + \gamma_n^f \||A^{-1}||\widehat{L}||\widehat{U}|\|_\infty, \tag{5.34b}$$
$$\tag{5.34c}$$

and finally by dropping second order terms and using Theorem 2.5 leads to

$$\kappa_\infty(\widetilde{A}) \lesssim (1 + \gamma_n^f \||A^{-1}||\widehat{L}||\widehat{U}|\|_\infty)^2 \lesssim (1 + f(n,\rho_n)u_f\kappa_\infty(A))^2, \tag{5.35}$$

showing that the precondtioned matrix $\widetilde{A}$ is relatively well-conditioned. Combining this bound on $\kappa(\widetilde{A})$ with (5.32), we obtain the condition

$$(u_g + u_p\kappa(A))(1 + \kappa(A)^2 u_f^2) \ll 1 \tag{5.36}$$

for the forward error to converge to its limiting value of order $q\,u_r\,\mathrm{cond}(A,x) + u$.

Next we determine a condition for the backward error to converge. First we need to bound the backward error of the original correction equation $Ad_i = r_i$. By observing that the computed residual $\widehat{r}_i$ satisfies $\widehat{r}_i - A\widehat{d}_i = \widehat{L}\widehat{U}(s_i - \widetilde{A}\widehat{d}_i)$, since $s_i = \widehat{U}^{-1}\widehat{L}^{-1}\widehat{r}_i$, and using

the bound (5.29) and Theorem 2.2 we obtain

$$\|\widehat{r}_i - A\widehat{d}_i\|_\infty \lesssim f(n,k,\rho_n)(u_g + u_p\kappa_\infty(A))\|\widehat{L}\widehat{U}\|_\infty(\|\widetilde{A}\|_\infty\|\widehat{d}_i\|_\infty + \|s_i\|_\infty) \tag{5.37a}$$

$$\lesssim f(n,k,\rho_n)(u_g + u_p\kappa_\infty(A))(\|\widetilde{A}\|_\infty\|A\|_\infty\|\widehat{d}_i\|_\infty + \kappa_\infty(A)\|\widehat{r}_i\|_\infty). \tag{5.37b}$$

We have thus shown that (4.5) holds with $u_s = u_g + u_p\kappa_\infty(A)$, $c_1 = f(n,k,\rho_n)\|\widetilde{A}\|_\infty$, and $c_2 = f(n,k,\rho_n)\kappa_\infty(A)$. Dropping constants and using Theorem 4.2 gives the convergence condition

$$(u_g + u_p\kappa(A))(1 + \|\widetilde{A}\|)\kappa(A) \ll 1. \tag{5.38}$$

From (5.34) we have

$$\|\widetilde{A}\|_\infty \le 1 + \gamma_n^f\||A^{-1}|\widehat{L}\|\widehat{U}|\|\|_\infty \ll 1 + u_f\kappa_\infty(A). \tag{5.39}$$

We finally obtain the condition

$$(u_g + u_p\kappa(A))(1 + \kappa(A)u_f)\kappa(A) \ll 1 \tag{5.40}$$

for the backward error to converge to its limiting accuracy of order $q\,u_r + u$. Note that condition (4.8) is implicitly included in the previous (5.40).

### 5.2.4 Comments on the results of the analysis

Curiously, condition (5.3) is stricter than the condition for the forward error, since (5.2) has an extra $u_f$ term. However, note that since the backward error is bounded by the forward error (see (2.7)), (5.2) is also an obvious condition for the backward error to converge to the limiting value of the forward error, of order $q\,\mathrm{cond}(A,x)u_r + u$, as already explained in section 4.2.3. In particular, if $u_r = u^2$, condition (5.3) is not useful because (5.2) also guarantees a backward error of order $u$ with a less restrictive condition on $\kappa(A)$, since by assumption $\kappa(A)u < 1$.

As a check, we compare our results with Higham [115], which analyzes the case $u_p = u_g = u_r = u$, that is, LU-GMRES-IR in two precisions, and uses a different argument to that here. Considering $1 \ll \kappa(A)u_f$, our condition (5.2) for convergence of the forward error is $\kappa(A)^3 u\,u_f^2 \ll 1$ and our condition (5.3) for convergence of the backward error is $\kappa(A)^3 u\,u_f \ll 1$. These conditions agree with Higham [115, eqs. (3.8), (3.6)].

We also compare with the analysis of Carson and Higham [45] for LU-GMRES-IR3, the conclusions from which we summarized in Theorem 4.4. For $u_g = u$ and $u_p = u^2$, our condition (5.2) for convergence of the forward error is $(u + \kappa(A)u^2)(1 + \kappa(A)^2u_f^2) \ll 1$, and requires in particular $\kappa(A)^2 u\,u_f^2 \ll 1$, since we are assuming $\kappa(A)u < 1$. Our condition (5.3) for convergence of the backward error requires, similarly, $\kappa(A)^2 u\,u_f \ll 1$. These conditions agree with (4.27) of Theorem 4.4. However, note that the condition for the backward error is stricter than what has been originally proposed in Carson and Higham [45], that is, $u\kappa(A) \ll 1$ (see discussion in the proof of Theorem 4.4).

## 5.3 Identifying meaningful combinations of precisions

Table 2.1 shows the most widely available floating-point arithmetics (we will refer to them by their symbols in the text); most modern supercomputers provide hardware (and software) support for at least three of them. Assuming for example that fp8 (E4M3), bfloat16, fp16, fp32, fp64, and fp128 can be used, LU-GMRES-IR5 has more than fifteen thousand different combinations of its five precision parameters. Among these, not all are relevant and, therefore, it is important to identify the subset of meaningful combinations. What we mean by meaningful has been explained in section 2.1.3.3, in the particular case of LU-GMRES-IR5, a combination is meaningful if none of the precisions it employs can be reduced without degrading the convergence conditions (5.2) and (5.3) and the limiting accuracies (5.1). Consequently, every meaningful combination of LU-GMRES-IR5 attains a trade-off between performance, robustness (ability to converge for ill-conditioned matrices), and accuracy (ability to converge to small errors). Note that, as it will be evoked in the next section 5.4, the convergence rate of the inner GMRES solver is somewhat unpredictable and, so, our definition of meaningful cannot take it into account.

The effort of formalizing the identification of the meaningful combinations is becoming essential; indeed, it is no longer straightforward to determine which of these combinations are relevant or not with an algorithm like LU-GMRES-IR5 which has five precision parameters. LU-IR3 and LU-GMRES-IR3 do not face this issue at such extent, and Carson and Higham [45] did not have to discuss much the problem.

As an example, the meaningful combinations for LU-IR3 must satisfy $u^2 \leq u_r \leq u \leq u_f$. Indeed, the limiting backward and forward errors (recalled in Table 4.2) show that we should have $u_r \leq u$ and that setting $u_r < u^2$ is not useful since $u_r = u^2$ is already enough to ensure a forward error of order $u$ (since by assumption $\kappa(A)u < 1$).

Meaningful combinations for LU-GMRES-IR5 also satisfy $u^2 \leq u_r \leq u \leq u_f$, so our aim now is to discuss the choice of the two new precision parameters $u_g \geq u$ and $u_p \geq u^2$. To compute the bounds on $\kappa(A)$ given by the conditions (5.2) and (5.3), we solve the equalities $(u_g + u_p x)(1 + u_f^2 x^2) = 1$ and $(u_g + u_p x)(1 + u_f x)x = 1$, respectively. We now state some observations that can be deduced from our analysis, in particular from condition (5.2).

- $u_p \leq u_g$. This first observation comes from the term $u_g + u_p\kappa(A)$ that appears in the convergence conditions. We would like the components of this term to be balanced, so that $u_p \approx u_g/\kappa(A) \leq u_g$. So $u_p < u_g$ may be required, but $u_p > u_g$ is not meaningful. We also note that there is no advantage to take $u_p < u_g/\kappa(A)$.

- $u_p < u_f$. This second observation comes from the fact that if $u_p = u_f$, condition (5.32) requires $\kappa(A)u_f\kappa(\widetilde{A}) \ll 1$, which is worse than the condition $\kappa(A)u_f \ll 1$ for LU-IR3.

- $u_p < u$, $u_p = u$, and $u_p > u$ are all meaningful. This is one of the main conclusions of our analysis. We know from Carson and Higham [45] (in which $u_p = u^2$ and $u_g = u$) that setting $u_p = u^2$ provides the least restrictive convergence conditions (4.27), but the precise role of $u_p$ in the convergence was not analyzed. With the new conditions (5.2) and (5.3) obtained from our generalized analysis, we can now understand what the conditions become if $u_p$ is taken larger than $u^2$. Crucially, setting $u_p = u$

Table 5.1: Bound on $\kappa(A)$, rounded to one significant figure, given by conditions (5.2) and (5.3) for the forward and backward errors to converge with LU-GMRES-IR5, depending on the precisions $u_f$, $u_g$, and $u_p$, and assuming the working precision $u$ is double. We recall that the forward error convergence condition for LU-IR3 is $\kappa(A) \ll 2 \times 10^1$ for $u_f =$ R, $\kappa(A) \ll 3 \times 10^2$ for $u_f =$ B, $\kappa(A) \ll 2 \times 10^3$ for $u_f =$ H, and $\kappa(A) \ll 2 \times 10^7$ for $u_f =$ S. The red, underlined terms denote combinations of precisions that are not meaningful. Each variant is presented in the form of a triplet $(u_f, u_g, u_p)$, hence HDQ means $u_f =$ H, $u_g =$ D, and $u_p =$ Q. We recall that $u_r$ does not play a role in the convergence bounds.

| | Variants $(u_f, u_g, u_p)$ | Forward | Backward |
|---|---|---|---|
| $u_f =$ R | RRB | $4 \times 10^1$ | $7 \times 10^0$ |
| | RBB RHB RSB RDB | $4 \times 10^1$ | $1 \times 10^1$ |
| | RRH | $5 \times 10^1$ | $9 \times 10^0$ |
| | RRS RRD RRQ | $6 \times 10^1$ | $1 \times 10^1$ |
| | RHH RSH RDH | $8 \times 10^1$ | $3 \times 10^1$ |
| | RBS RBD RBQ | $3 \times 10^2$ | $7 \times 10^1$ |
| | RHS RHD RHQ | $7 \times 10^2$ | $2 \times 10^2$ |
| | RSS RDS | $2 \times 10^3$ | $6 \times 10^2$ |
| | RSD RSQ | $7 \times 10^4$ | $2 \times 10^4$ |
| | RDD | $1 \times 10^6$ | $5 \times 10^5$ |
| | RDQ | $2 \times 10^9$ | $4 \times 10^8$ |
| $u_f =$ B | BRH | $5 \times 10^2$ | $1 \times 10^1$ |
| | BBH BHH BSH BDH | $5 \times 10^2$ | $4 \times 10^1$ |
| | BRS BRD BRQ | $1 \times 10^3$ | $2 \times 10^1$ |
| | BBS BBD BBQ | $4 \times 10^3$ | $2 \times 10^2$ |
| | BHS | $8 \times 10^3$ | $6 \times 10^2$ |
| | BHD BHQ | $1 \times 10^4$ | $6 \times 10^2$ |
| | BSS BDS | $1 \times 10^4$ | $2 \times 10^3$ |
| | BSD BDQ | $1 \times 10^6$ | $7 \times 10^4$ |
| | BDD | $8 \times 10^6$ | $1 \times 10^6$ |
| | BDQ | $2 \times 10^{10}$ | $2 \times 10^9$ |
| $u_f =$ H | HRS HRD HRQ | $8 \times 10^3$ | $2 \times 10^1$ |
| | HBS HBD HBQ | $3 \times 10^4$ | $2 \times 10^2$ |
| | HHS | $4 \times 10^4$ | $1 \times 10^3$ |
| | HSS HDS | $4 \times 10^4$ | $3 \times 10^3$ |
| | HHD HHQ | $9 \times 10^4$ | $1 \times 10^3$ |
| | HSD HSQ | $8 \times 10^6$ | $2 \times 10^5$ |
| | HDD | $3 \times 10^7$ | $3 \times 10^6$ |
| | HDQ | $2 \times 10^{11}$ | $4 \times 10^9$ |
| $u_f =$ S | SRD SRQ | $7 \times 10^7$ | $2 \times 10^1$ |
| | SBD SBQ | $3 \times 10^8$ | $3 \times 10^2$ |
| | SHD SHQ | $7 \times 10^8$ | $2 \times 10^3$ |
| | SSD | $1 \times 10^{10}$ | $1 \times 10^7$ |
| | SDD | $1 \times 10^{10}$ | $5 \times 10^7$ |
| | SSQ | $7 \times 10^{10}$ | $1 \times 10^7$ |
| | SDQ | $2 \times 10^{15}$ | $4 \times 10^{11}$ |

and even $u_p > u$ can potentially yield conditions that remain less restrictive than the LU-IR3 condition $\kappa(A)u_f \ll 1$, and therefore represent meaningful combinations. Let us illustrate this observation with a practical example. Assume $u_f$ is set to fp16 and $u = u_g$ are set to fp64. Then the condition for the forward error to converge with LU-IR3 is $\kappa(A) \ll 2 \times 10^3$. Instead, with LU-GMRES-IR5:

- If $u_p$ is set to fp128 (as in Carson and Higham [45]), condition (5.2) becomes $\kappa(A) \ll 2 \times 10^{11}$. We recover the same condition as in Carson and Higham [45].
- If $u_p$ is set to fp64 (and thus $u_p = u$), condition (5.2) becomes $\kappa(A) \ll 3 \times 10^7$, which is still much better than the LU-IR3 condition. This version of LU-GMRES-IR5 has been successfully used in practice, for example by Haidar et al. [103; 102].
- If $u_p$ is set to fp32 (and thus $u_p > u$), condition (5.2) becomes $\kappa(A) \ll 4 \times 10^4$, which is still over an order of magnitude better than the LU-IR3 condition. Note that this variant uses up to four different precisions if $u_r$ is set to fp128.

These examples illustrate how the $u_p$ precision can be tuned to achieve different levels of trade-off between robustness (ability to handle ill-conditioned matrices) and performance (cost of the application of LU factors within GMRES).

- $u_g = u$ and $u_g > u$ are both meaningful. This is also an important conclusion of our analysis. Whereas in Carson and Higham [45], $u_g$ is set to $u$ to obtain the least restrictive convergence condition, our analysis reveals that setting $u_g > u$ can also be meaningful. In fact, as long as $u_g < 1$, condition (5.2) is better than the LU-IR3 condition $\kappa(A)u_f \ll 1$, so we have much flexibility in choosing $u_g$. Let us again illustrate this with a practical example. Assume $u_f$ is set to fp16 and $u = u_p$ are set to fp64. With $u_g = u$, as previously stated, the condition (5.2) is $\kappa(A) \ll 3 \times 10^7$. However, if $u_g$ is set to fp32, the condition is $\kappa(A) \ll 8 \times 10^6$, which is only slightly worse and remains much better than the LU-IR3 condition. The precision $u_g$ can be chosen even lower, for example if $u_g$ is set to fp16 (i.e., $u_g = u_f$), the condition becomes $\kappa(A) \ll 9 \times 10^4$ and still remains better than the LU-IR3 one. Note that with $u_g$ set to fp32, setting $u_p$ to fp128 instead of fp64 does not improve the condition $\kappa(A) \ll 8 \times 10^6$: this shows that the meaningful values of $u_p$ can be influenced by the choice of $u_g$, and vice versa. In conclusion the $u_g$ precision can also be tuned to achieve different levels of trade-off between robustness and performance (the cost of GMRES and, in particular, the memory footprint of the Krylov basis).

- $u_g < u_f$, $u_g = u_f$, and $u_g > u_f$ are all meaningful. This final observation is that $u_g$ and $u_f$ can be independent. We have already illustrated in previous examples where $u_f$ is fp16 that $u_g < u_f$ and $u_g = u_f$ are both meaningful. This final observation states that even $u_g > u_f$ can be meaningful. To see why, let us take another example with $u_f$ set to fp32 and $u = u_p$ set to fp64. Then setting $u_g$ to fp16 yields the condition $\kappa(A) \ll 7 \times 10^8$, which is better than any combination with the same $u$ and $u_p$ but with $u_f$ set to fp16 (the best possible condition being $\kappa(A) \ll 3 \times 10^7$ for $u_g$ at least in fp64). Usually, one would expect an fp16 LU factorization and fp32 GMRES to be faster than

the converse (fp32 factorization and fp16 GMRES), so in practice the combinations with $u_g > u_f$ are only relevant for a narrow range of $\kappa(A)$ (in this example, for $\kappa(A)$ such that $\kappa(A) \ll 3 \times 10^7$ is not satisfied but $\kappa(A) \ll 7 \times 10^8$ is).

In Table 5.1, we summarize all the possible combinations of $u_p$ and $u_g$ when $u_f$ is set to fp8 (E4M3), bfloat16, fp16, or fp32 (these three precisions being the ones that determine the convergence conditions). The underlined, red terms correspond to combinations of these three precisions that are not meaningful. Interestingly, when the working precision $u$ is fp64 and the residual precision $u_r$ is fp128, Table 5.1 contains twenty six meaningful variants that use at least four different arithmetics, including seven that use all five (RBS, RHS, BRH, BRS, HRS, BHS, HBS).

All the observations made in this part reduce the number of meaningful combinations of the five precision parameters to a subset of 235 combinations, corresponding to 1.6% of all the possibilities. We summarize the numerical properties (convergence condition and limiting accuracy, for both the forward and backward errors) of a selected subset of 66 LU-GMRES-IR5 variants in Table 5.2. Importantly, the table includes variants used in existing implementations, for which our analysis provides new theoretical guarantees. It also includes new combinations of precisions not proposed previously that achieve new, finer trade-offs between the convergence conditions and the precisions used.

In order to select the appropriate combination of precisions, we have shown in this section that the user can choose within the subset of the meaningful combinations those providing the appropriate convergence condition and forward and backward error bounds according to the application requirements. Within this selection the user can take into account hardware and software features to further refine the selection.

## 5.4 GMRES stopping criterion

In this manuscript and in Carson and Higham [44; 45], we stop MGS-GMRES when the normwise backward error of the preconditioned system $\widetilde{A}d_i = \widehat{s}_i$ becomes smaller than a prescribed value $\tau_g$. The use of this stopping criterion helps to reduce the cumulated number of GMRES iterations over the outer iteration of LU-GMRES-IR5 and preserve the backward stability. In section 5.4.1, we explain why it is essential to limit the number of iterations, in section 5.4.2 we discuss the ideal choice of $\tau_g$ to reduce it, and finally, we present the impact of using this stopping criterion on the convergence conditions of Theorem 5.1 in section 5.4.3.

### 5.4.1 On the cost of refinement iterations

Although, in theory, the refinement iterations of LU-GMRES-IR5 are asymptotically negligible compared with the factorization ($\mathcal{O}(n^2)$ compared with $\mathcal{O}(n^3)$ for dense systems), in practice, it is not necessarily the case, and a high number of cumulated inner iterations greatly affects the performance. Unfortunately, what we mean by a high number of cumulated inner iterations is generally just about a few dozen. The inability to recover this asymptotic property originates from the relative inefficiency of the refinement steps using

BLAS 2 kernels compared with the factorization using BLAS 3 kernels. As a consequence, LU-GMRES-IR5 (and to some extent LU-IR3) can easily lose any time improvement from low precision factorization by doing too many iterations. The issue is even more exacerbated by the fact that the solve operations need to be applied in high precision $u_p$ and that, if the precision $u_f$ is chosen low, the preconditioner quality will be degraded and the number of iterations will increase. The cost of the refinement steps has been evaluated, for example, by Haidar et al. [103] for the dense case, and will be evaluated for the sparse case in chapter 6 of this manuscript.

For these reasons, LU-GMRES-IR5 should make a relatively small number of cumulated inner iterations to be useful; sometimes, a well-chosen $\tau_g$ can drastically improve this cumulated number. On the positive side of this constraint, we often work with small Krylov basis, which means that the memory used to store the basis is barely noticeable, and the orthogonalization cost remains negligible compared with the matrix–vector product with $A$ or the LU solve operations.

The fact that LU-GMRES-IR5 requires a small number of iterations to be efficient and competitive is actually a well-known issue of the algorithm. Current efforts are exploring some way of improvement; for example, Oktay and Carson [171] proposed to use a recycling strategy that can limit the cumulated number of inner iteration of GMRES.

### 5.4.2 On the convergence behavior

In LU-GMRES-IR5 we have two nested iterative processes: the outer iterations of iterative refinement and the inner iterations of GMRES. It is important to understand their behavior to understand better how the stopping criterion can be chosen to limit the cumulated iterations.

At each LU-GMRES-IR5 outer iteration, we approximatively decrease the forward and backward errors of the original linear system (1.1) by factors of $(u_g + u_p\kappa(A))(1+\kappa(A)^2 u_f^2)$ and $(u_g + u_p\kappa(A))(1+\kappa(A)u_f)\kappa(A)$, respectively; they are the convergence rates corresponding to the left-hand sides of the conditions of Theorem 5.1. These convergence rates are constant across the iterations and, for the convergence to happen, we require them to be far lower than 1. Consequently, it means that we can efficiently bound the final number of outer iterations needed to reach a certain accuracy on the solution.

However, while the iterative refinement outer iterations have a predictable behavior with a constant convergence rate, this is not the case for the inner iterations of GMRES. Actually, the convergence rate of GMRES is a wide topic still discussed today (see Titley-Peloquin et al. [204], Freitag et al. [77]) and, unfortunately, exploitable bounds allowing a rough prediction of the number of iterations for the general case at low cost do not exist; a classical bound being [204, Eq. (2.1)]

$$\frac{\|r_k\|_2}{\|r_0\|_2} \leq \kappa_2(X) \min_{p\in\Pi_k} \max_{\lambda\in\sigma(A)} |p(\lambda)|, \quad k = 1,2,..., \tag{5.41}$$

where we suppose $A$ diagonalizable such that $A = X\Lambda X^{-1}$ with $\Lambda$ diagonal, where $\sigma(A) = \{\lambda_1,\ldots,\lambda_n\}$ is the spectrum of $A$, where $\Pi_k$ is the set of polynomials of degree at most $k$

Table 5.2: Bounds on $\kappa(A)$, rounded to one significant figure, such that LU-GMRES-IR5 with the precisions in the first five columns is guaranteed to converge to the indicated limiting accuracies.

| | | | | | Forward error | | Backward error | |
|---|---|---|---|---|---|---|---|---|
| $u$ | $u_r$ | $u_f$ | $u_g$ | $u_p$ | $\kappa(A)$ | Limit | $\kappa(A)$ | Limit |
| S | S | R | B | S | $3\times10^2$ | $\mathrm{cond}(A,x)\times$ S | $7\times10^1$ | S |
| S | S | R | H | S | $7\times10^2$ | $\mathrm{cond}(A,x)\times$ S | $2\times10^2$ | S |
| S | S | R | S | S | $2\times10^3$ | $\mathrm{cond}(A,x)\times$ S | $6\times10^2$ | S |
| S | S | B | B | S | $4\times10^3$ | $\mathrm{cond}(A,x)\times$ S | $2\times10^2$ | S |
| S | S | B | H | S | $8\times10^3$ | $\mathrm{cond}(A,x)\times$ S | $6\times10^2$ | S |
| S | S | B | S | S | $1\times10^4$ | $\mathrm{cond}(A,x)\times$ S | $2\times10^3$ | S |
| S | S | H | B | S | $3\times10^4$ | $\mathrm{cond}(A,x)\times$ S | $2\times10^2$ | S |
| S | S | H | H | S | $4\times10^4$ | $\mathrm{cond}(A,x)\times$ S | $1\times10^3$ | S |
| S | S | H | S | S | $4\times10^4$ | $\mathrm{cond}(A,x)\times$ S | $3\times10^3$ | S |
| S | S | R | S | D | $7\times10^4$ | $\mathrm{cond}(A,x)\times$ S | $2\times10^4$ | S |
| S | S | B | S | D | $1\times10^6$ | $\mathrm{cond}(A,x)\times$ S | $7\times10^4$ | S |
| S | S | H | S | D | $8\times10^6$ | $\mathrm{cond}(A,x)\times$ S | $2\times10^5$ | S |
| S | D | R | B | S | $3\times10^2$ | S | $7\times10^1$ | S |
| S | D | R | H | S | $7\times10^2$ | S | $2\times10^2$ | S |
| S | D | R | S | S | $2\times10^3$ | S | $6\times10^2$ | S |
| S | D | B | B | S | $4\times10^3$ | S | $2\times10^2$ | S |
| S | D | B | H | S | $8\times10^3$ | S | $6\times10^2$ | S |
| S | D | B | S | S | $1\times10^4$ | S | $2\times10^3$ | S |
| S | D | H | B | S | $3\times10^4$ | S | $2\times10^2$ | S |
| S | D | H | H | S | $4\times10^4$ | S | $1\times10^3$ | S |
| S | D | H | S | S | $4\times10^4$ | S | $3\times10^3$ | S |
| S | D | R | S | D | $7\times10^4$ | S | $2\times10^4$ | S |
| S | D | B | S | D | $1\times10^6$ | S | $7\times10^4$ | S |
| S | D | H | S | D | $8\times10^6$ | S | $2\times10^5$ | S |
| D | D | R | B | S | $3\times10^2$ | $\mathrm{cond}(A,x)\times$ D | $7\times10^1$ | D |
| D | D | R | H | S | $7\times10^2$ | $\mathrm{cond}(A,x)\times$ D | $2\times10^2$ | D |
| D | D | R | S | S | $2\times10^3$ | $\mathrm{cond}(A,x)\times$ D | $6\times10^2$ | D |
| D | D | B | B | S | $4\times10^3$ | $\mathrm{cond}(A,x)\times$ D | $2\times10^2$ | D |
| D | D | B | H | S | $8\times10^3$ | $\mathrm{cond}(A,x)\times$ D | $6\times10^2$ | D |
| D | D | B | S | S | $1\times10^4$ | $\mathrm{cond}(A,x)\times$ D | $2\times10^3$ | D |
| D | D | H | B | S | $3\times10^4$ | $\mathrm{cond}(A,x)\times$ D | $2\times10^2$ | D |
| D | D | H | H | S | $4\times10^4$ | $\mathrm{cond}(A,x)\times$ D | $1\times10^3$ | D |
| D | D | H | S | S | $4\times10^4$ | $\mathrm{cond}(A,x)\times$ D | $3\times10^3$ | D |
| D | D | R | S | D | $7\times10^4$ | $\mathrm{cond}(A,x)\times$ D | $2\times10^4$ | D |
| D | D | B | S | D | $1\times10^6$ | $\mathrm{cond}(A,x)\times$ D | $7\times10^4$ | D |
| D | D | R | D | D | $1\times10^6$ | $\mathrm{cond}(A,x)\times$ D | $5\times10^5$ | D |

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| D | D | B | D | D | $8\times10^{6}$ | $\mathrm{cond}(A,x)\times$ D | $1\times10^{6}$ | D |
| D | D | H | S | D | $8\times10^{6}$ | $\mathrm{cond}(A,x)\times$ D | $2\times10^{5}$ | D |
| D | D | H | D | D | $3\times10^{7}$ | $\mathrm{cond}(A,x)\times$ D | $3\times10^{6}$ | D |
| D | D | S | H | D | $7\times10^{8}$ | $\mathrm{cond}(A,x)\times$ D | $2\times10^{3}$ | D |
| D | D | R | D | Q | $2\times10^{9}$ | $\mathrm{cond}(A,x)\times$ D | $4\times10^{8}$ | D |
| D | D | S | D | D | $1\times10^{10}$ | $\mathrm{cond}(A,x)\times$ D | $5\times10^{7}$ | D |
| D | D | B | D | Q | $2\times10^{10}$ | $\mathrm{cond}(A,x)\times$ D | $2\times10^{9}$ | D |
| D | D | H | D | Q | $2\times10^{11}$ | $\mathrm{cond}(A,x)\times$ D | $4\times10^{9}$ | D |
| D | D | S | D | Q | $2\times10^{15}$ | $\mathrm{cond}(A,x)\times$ D | $4\times10^{11}$ | D |
| D | Q | R | B | S | $3\times10^{2}$ | D | $7\times10^{1}$ | D |
| D | Q | R | H | S | $7\times10^{2}$ | D | $2\times10^{2}$ | D |
| D | Q | R | S | S | $2\times10^{3}$ | D | $6\times10^{2}$ | D |
| D | Q | B | B | S | $4\times10^{3}$ | D | $2\times10^{2}$ | D |
| D | Q | B | H | S | $8\times10^{3}$ | D | $6\times10^{2}$ | D |
| D | Q | B | S | S | $1\times10^{4}$ | D | $2\times10^{3}$ | D |
| D | Q | H | B | S | $3\times10^{4}$ | D | $2\times10^{2}$ | D |
| D | Q | H | H | S | $4\times10^{4}$ | D | $1\times10^{3}$ | D |
| D | Q | H | S | S | $4\times10^{4}$ | D | $3\times10^{3}$ | D |
| D | Q | R | S | D | $7\times10^{4}$ | D | $2\times10^{4}$ | D |
| D | Q | B | S | D | $1\times10^{6}$ | D | $7\times10^{4}$ | D |
| D | Q | R | D | D | $1\times10^{6}$ | D | $5\times10^{5}$ | D |
| D | Q | B | D | D | $8\times10^{6}$ | D | $1\times10^{6}$ | D |
| D | Q | H | S | D | $8\times10^{6}$ | D | $2\times10^{5}$ | D |
| D | Q | H | D | D | $3\times10^{7}$ | D | $3\times10^{6}$ | D |
| D | Q | S | H | D | $7\times10^{8}$ | D | $2\times10^{3}$ | D |
| D | Q | R | D | Q | $2\times10^{9}$ | D | $4\times10^{8}$ | D |
| D | Q | S | D | D | $1\times10^{10}$ | D | $5\times10^{7}$ | D |
| D | Q | B | D | Q | $2\times10^{10}$ | D | $2\times10^{9}$ | D |
| D | Q | H | D | Q | $2\times10^{11}$ | D | $4\times10^{9}$ | D |
| D | Q | S | D | Q | $2\times10^{15}$ | D | $4\times10^{11}$ | D |

with $p(0)=1$, and where $r_k$ is the residual at step $k$. Note that more descriptive bounds can be achieved, for example, for normal or symmetric positive definite matrices. While we can conclude that the convergence of GMRES is in part linked to the matrix spectrum, the eigenvalues alone certainly do not describe the convergence behavior. Indeed, as shown by Greenbaum et al. [96], every non-increasing convergence curve is possible for GMRES, independently of the spectrum. Hence, it is admitted by the community that this is a hard problem. See Simoncini and Szyld [188, sect. 6] for a good summary of the state of the art on the topic.

Regarding the stopping criterion, it means that the higher $\tau_g$, the fewer the inner iterations done in one call of GMRES. However, the higher $\tau_g$, the more the outer iterations since the accuracy of the solution of the correction equation will be worsened and will readily affect the convergence rate of iterative refinement. Therefore, the ideal $\tau_g$ should

be a trade-off: large enough to guarantee fast GMRES convergence but small enough to guarantee fast iterative refinement convergence. Unfortunately, due to the relative unpredictability of the GMRES convergence, this ideal choice of $\tau_g$ has to be made empirically for a given problem. Actually, the problem of finding a good $\tau_g$ is comparable with finding a suitable restart parameter for restarted GMRES.

### 5.4.3 Rounding error analysis with stopping criterion

Stopping MGS-GMRES when we reach a backward error smaller than $\tau_g$ affects the accuracy of the solution of the preconditioned correction equation $\widetilde{A}d_i = \widehat{s}_i$ and, therefore, the convergence conditions (5.2) and (5.3) of Theorem 5.1. We now explain how this choice of stopping criterion preserves the algorithm's stability and how we can take it into account in our previous results.

Assuming MGS-GMRES is stopped at iteration $j \leq n$ such that

$$\frac{\|\widetilde{A}\widehat{d}_j - \widehat{s}_i\|_2}{\|\widetilde{A}\|_F\|\widehat{d}_j\|_2 + \|\widehat{s}_i\|_2} \leq \tau_g, \tag{5.42}$$

we can state that MGS-GMRES provides a computed solution $\widehat{d}_i \equiv \widehat{d}_j$ corresponding to the true solution of the perturbed system $(\widetilde{A} + \Delta\widetilde{A})\widehat{d}_i = \widehat{s}_i + \Delta\widehat{s}_i$ where the perturbations $\Delta\widetilde{A}$ and $\Delta\widehat{s}_i$ satisfy

$$\|\Delta\widetilde{A}\|_F \leq \tau_g\|\widetilde{A}\|_F, \quad \|\Delta\widehat{s}_i\|_2 \leq \tau_g\|\widehat{s}_i\|_2. \tag{5.43}$$

We can adapt Theorem 5.1 for the convergence of LU-GMRES-IR5 to the case where we use a stopping criterion $\tau_g$ in MGS-GMRES as follows.

**Theorem 5.3** (Convergence of LU-GMRES-IR5 with stopping criterion)**.** *Let* (1.1) *be solved by LU-GMRES-IR5 (Algorithm 5.1) using MGS-GMRES with a stopping criterion $\tau_g$ at step 5 and using GEPP at step 1. If $u_g \geq u$ and $\kappa(A)u_p < 1$, the forward and backward errors will reach their respective limiting accuracies*

$$p\,u_r\,\mathrm{cond}(A,x) + u \text{ (forward)} \qquad \text{and} \qquad p\,u_r + u \text{ (backward)}, \tag{5.44}$$

*provided that*

$$\max(u_g, u_p\kappa(A), \tau_g)(1 + \kappa(A)^2 u_f^2) \ll 1 \text{ (forward)} \tag{5.45}$$

*and*

$$\max(u_g, u_p\kappa(A), \tau_g)(1 + \kappa(A)u_f)\kappa(A) \ll 1 \text{ (backward)}. \tag{5.46}$$

*Proof.* From (5.43), we know that the use of the stopping criterion $\tau_g$ readily affect the accuracy of the computed solution $\widehat{d}_i$, therefore, revisiting the previous analysis of section 5.2.2 with the introduction of the stopping criterion $\tau_g$ will change (5.26), it becomes

$$(\widetilde{A} + \Delta\widetilde{A})\widehat{d}_i = \widehat{s}_i + \Delta\widehat{s}_i, \tag{5.47a}$$

$$\|\Delta\widetilde{A}\|_F \lesssim \max(\widetilde{\gamma}_{kn}^g, n\widetilde{\gamma}_{k^{1/2}n}^p\kappa_\infty(A), \tau_g)\|\widetilde{A}\|_F, \tag{5.47b}$$

$$\|\Delta\widehat{s}_i\|_2 \lesssim \max(\widetilde{\gamma}_{kn}^g, \tau_g)\|\widehat{s}_i\|_\infty. \tag{5.47c}$$

It simply translates the fact that for the computation of $\widehat{d}_i$ with MGS-GMRES, either we meet the stopping criterion and the perturbations $\Delta\widetilde{A}$ and $\Delta\widehat{s}_i$ satisfy (5.43), or that the stopping criterion is not met, but the perturbations are still guaranteed to satisfy (5.26).

Adapting section 5.2.2 by replacing (5.26) with (5.47) is straightforward and concludes the proof. □

A stopping criterion based on the quality of the backward error of the solution rather than a fixed number of iterations (as it is often done in restarted GMRES) guarantees the backward stability of LU-GMRES-IR5. This is because we need a guarantee on the accuracy of the computed solution $\widehat{d}_i$ at each iteration of iterative refinement to make use of Theorems 4.1 and 4.2. However, of course, any kind of stopping criterion can be chosen in practice, and one can choose to stop GMRES after a fixed number of iterations in LU-GMRES-IR5.

## 5.5 Numerical experiments

We now perform numerical experiments to assess the validity of the convergence conditions of LU-GMRES-IR5 derived in section 5.2. Throughout our experiments, we focus on the forward error convergence (condition (5.2)), we fix $u = \mathrm{D}$ and $u_r = \mathrm{Q}$, and we analyze the role of the factorization precision $u_f$ and that of the newly introduced precisions $u_p$ (preconditioner precision) and $u_g$ (GMRES precision). In section 5.5.1, we first use random dense matrices to validate experimentally the forward convergence condition of Theorem 5.1 and, then, in section 5.5.2, we evaluate the numerical behaviors of different meaningful variants of LU-GMRES-IR5 on real-life sparse matrices.

We have written a Julia code that implements LU-GMRES-IR5 and LU-IR3, where half precision arithmetics (fp16 and blfoat16) are simulated. No libraries are available yet to simulate fp8 in Julia. We have made this code publicly available[1].

### 5.5.1 Random dense matrices

We first use random dense matrices with prescribed 2-norm condition number $\kappa(A)$ which are generated in Julia with the command `matrixdepot('randsvd', n, kappa, mode)` where `mode = 2`, that is, matrices with one small singular value. Note that this class of matrices leads to unusually large growth factors $\rho_n$ of order $n$ (see Higham et al. [108]). However, we use only small matrices ($n = 50$) for which $\rho_n$ does not exceed 20.

We take $\kappa(A) = 10^c$, for $c = 0{:}\,17$ and, for each value of $\kappa(A)$, we generate 100 random $50\times 50$ matrices of corresponding condition number. Then, we run LU-IR3 and LU-GMRES-IR5 on each matrix and compute their success rate, that is, the percentage of matrices for which a forward error $\|x-\hat{x}\|_2/\|x\|_2 \leq 4.44\times 10^{-16}$ is achieved, since with $u = \mathrm{D}$ and $u_r = \mathrm{Q}$ the forward error should reach full double precision with no dependency on $\mathrm{cond}(A,x)$ (see the limiting accuracy (5.1)).

Figure 5.1 reports the success rate of LU-IR3 and nine variants of LU-GMRES-IR5, corresponding to all possible combinations of the $u_p$ and $u_g$ parameters over the values

[1] https://github.com/bvieuble/Itref.jl

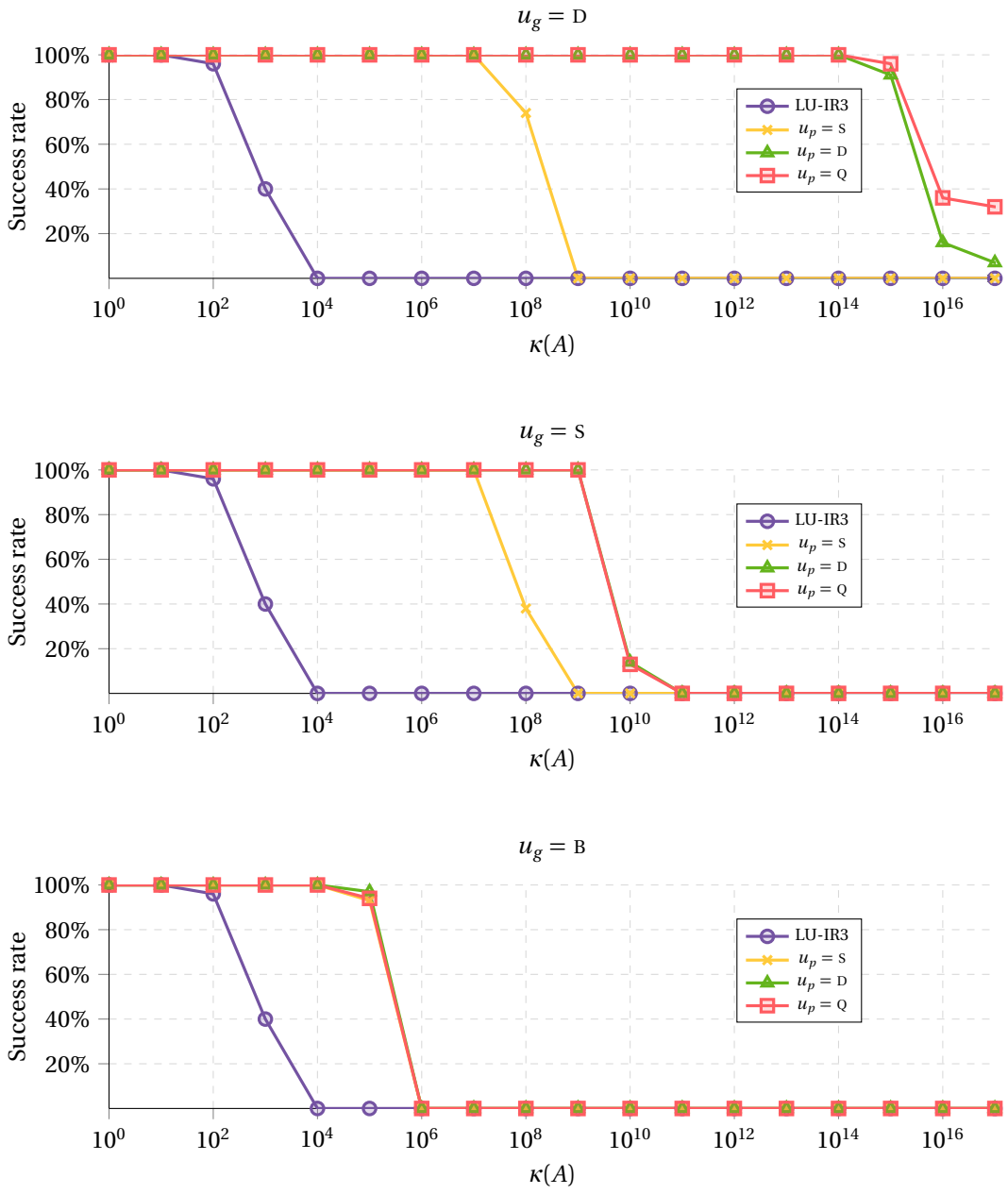



Figure 5.1: Proportion of matrices for which LU-IR3 and LU-GMRES-IR5 with varying $u_p$ and $u_g$ converged to double precision forward error as a function of $\kappa(A)$. For all variants, $u_f = \mathrm{B}$, $u = \mathrm{D}$, and $u_r = \mathrm{Q}$.

$u_p = \mathrm{S},\mathrm{D},\mathrm{Q}$ and $u_g = \mathrm{B},\mathrm{S},\mathrm{D}$, with $u_f = \mathrm{B}$ fixed for all variants. This experiment allows us to obtain an empirical bound on the value of $\kappa(A)$ at which each variant stops being able to converge. For example, the success rate of LU-IR3 is 100% as long as $\kappa(A) \leq 10^2$, but starts decreasing for larger $\kappa(A)$, and quickly becomes 0%. This experimentally confirms the theoretical condition $\kappa(A) \ll 2 \times 10^3$ given by (4.24).

Let us now analyze the LU-GMRES-IR5 variants, starting with the role of the precondi-

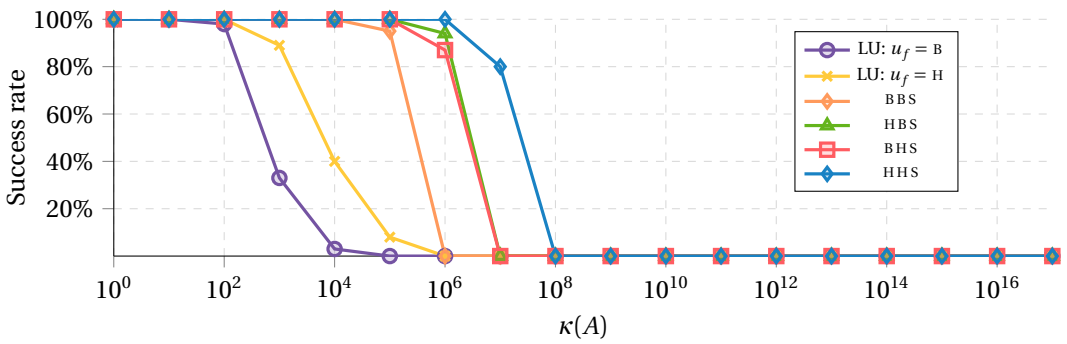



Figure 5.2: Proportion of matrices for which LU-IR3 and LU-GMRES-IR5 variants (including five meaningful precisions combinations, with $u_f$, $u_g$, $u_p$ as specified in the legend) converged to double precision forward error as a function of $\kappa(A)$. For all variants $u = \text{D}$ and $u_r = \text{Q}$.

tioner precision $u_p$ with fixed $u_g = D$ (top graph of Figure 5.1). We can observe that convergence is achieved with 100% success rate as long as $\kappa(A)$ is smaller than $10^7$ for $u_p = \text{S}$ and $10^{15}$ for $u_p = \text{D}$ or Q. The relative robustness of each method is therefore consistent with the theoretical bounds of Table 5.1. However, the variants with $u_p = \text{S}$ and $u_p = \text{D}$ both perform much better than expected. This is not entirely surprising since the analysis can be pessimistic, especially in the bound (5.35) on $\kappa(\widetilde{A})$, which in practice has been observed to often be of order $\kappa(A)u_f$ rather than the worst-case bound $(\kappa(A)u_f)^2$ (see Carson and Higham [44], Ogita [169], Rump [180]).

Next we analyze the role of the GMRES precision $u_g$ by comparing the top graph of Figure 5.1 with the middle and bottom ones. When $u_p = \text{S}$, switching from $u_g = \text{D}$ (top) to $u_g = \text{S}$ (middle) has no impact on the success rate, which equals 100% as long as $\kappa(A) \leq 10^7$; reducing the precision even further by setting $u_g = \text{B}$ (bottom) only has a slight impact: the success rate remains at 100% for $\kappa(A) \leq 10^5$. When $u_p = \text{D}$ or Q, reducing the GMRES precision has a much more visible impact: a 100% success rate is achieved only when $\kappa(A) \leq 10^9$ ($u_g = \text{S}$, middle) or $\kappa(A) \leq 10^5$ ($u_g = \text{B}$, bottom).

These experiments also show that the success rate is independent of $u_p$ when $u_g = \text{B}$, and almost independent of $u_g$ when $u_p = \text{S}$. This illustrates that the meaningful choices of $u_p$ depend on the choice of $u_g$, and vice versa. Overall, the experiments of Figure 5.1 are therefore in good agreement with the theoretical bounds. Importantly, they confirm that even with relaxed requirements on the precisions $u_p$ and $u_g$, LU-GMRES-IR5 can still handle matrices that are much more ill-conditioned than LU-IR3.

Finally, we evaluate in Figure 5.2 two meaningful variants of Table 5.1 that use five different precisions: BHS ($u_f = \text{B}$, $u_g = \text{H}$, $u_p = \text{S}$) and HBS ($u_f = \text{H}$, $u_g = \text{B}$, $u_p = \text{S}$). We compare these two variants with LU-IR3 with $u_f = \text{B}$ or H, and with the BBS and HHS LU-GMRES-IR5 variants ($u_f = \text{B}$, $u_g = \text{B}$, $u_p = \text{S}$ and $u_f = \text{H}$, $u_g = \text{H}$, $u_p = \text{S}$), which are the four-precision variants right below and above in terms of convergence condition. The figure experimentally confirms that both of these five-precision variants are in between BBS and HHS and are able to handle more ill-conditioned matrices than LU-IR3, showing

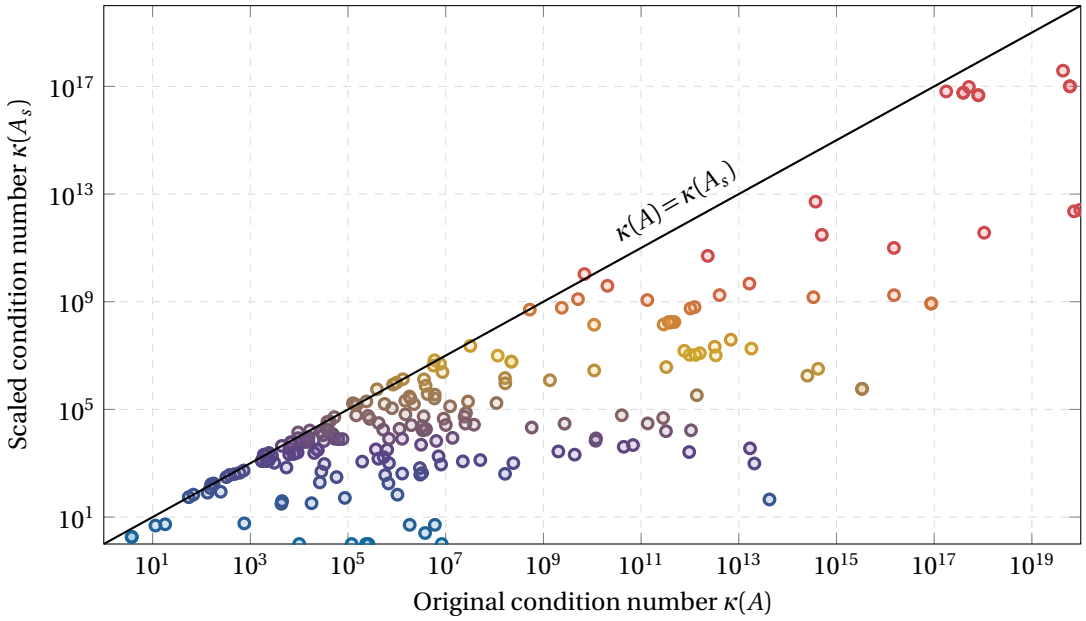



Figure 5.3: Condition number of the scaled matrix $\kappa(A_s)$ according to the original condition number before scaling $\kappa(A)$.

that they are meaningful not only theoretically but also in practice.

### 5.5.2 Real-life matrices from SuiteSparse

With the following set of experiments carried out on a large set of matrices from the SuiteSparse collection (Davis and Hu [55]) we want to verify two main points. First, we want to check whether reducing the preconditioner and/or GMRES precisions impacts the number of iterations required to converge. Second, we want to observe if the conclusions of the previous experiments made on random dense matrices for the forward error convergence extend to real-life sparse matrices. To do so, we use a set of 230 matrices; these matrices are all real, square, and of dimension between 500 and 3000. We compare LU-IR3 with the meaningful LU-GMRES-IR5 variants over the values $u_f = \mathrm{B}, \mathrm{H}, \mathrm{S}$. In these experiments, rather than counting the number of iterations, we count the number of calls to LU triangular solves: this is because in the case of LU-GMRES-IR5, an extra LU solve is required at the start of each iterative refinement step.

For the experiments that use an LU factorization in fp16 arithmetic, we must pay attention to the narrow range of this arithmetic (see Table 2.1 and discussion of section 2.1.3.2). Many matrices in our set have entries outside of this range, and so it is essential to address this issue. We use the diagonal scaling method of Higham et al. [121] described in section 4.7, which first normalizes every row and column by its maximum value (preventing overflow), and then scales the matrix by a quantity $\lambda$ close to the maximum representable value (to minimize underflow).

One difficulty is that the number of iterations is very sensitive to the choice of the GMRES stopping criterion $\tau$ and the scaling factor $\lambda$, and that the optimal choice of $\tau$

Table 5.3: Number of LU solves on a sample of SuiteSparse matrices for LU-IR3 and LU-GMRES-IR5 variants with $u_f = \mathrm{B}$. $\kappa(A)$ is the original condition number, $\kappa(A_s)$ is the condition number after scaling. The matrices are sorted by increasing $\kappa(A_s)$. A "—" indicates failure to converge.

| Name | $n$ | $\kappa(A)$ | $\kappa(A_s)$ | LU-IR3 | BBS | BBD | BSS | BSD | BDD | BDQ |
|---|---|---|---|---|---|---|---|---|---|---|
| dw256B | 512 | 3.7E+00 | 1.8E+00 | 8 | 24 | 24 | 10 | 10 | 10 | 10 |
| gre_512 | 512 | 1.6E+02 | 1.7E+02 | 16 | 28 | 28 | 17 | 17 | 14 | 14 |
| mahindas | 1258 | 2.1E+13 | 9.8E+02 | 12 | 34 | 36 | 14 | 12 | 12 | 12 |
| dw1024 | 2048 | 2.1E+03 | 1.9E+03 | 18 | 36 | 36 | 20 | 20 | 19 | 19 |
| fs_541_4 | 541 | 1.2E+10 | 6.9E+03 | — | 48 | 48 | 33 | 35 | 28 | 28 |
| rajat12 | 1879 | 6.9E+05 | 8.1E+03 | — | 94 | 69 | 29 | 29 | 24 | 24 |
| sherman1 | 1000 | 1.6E+04 | 9.3E+03 | — | 78 | 78 | 30 | 26 | 26 | 26 |
| watt_2 | 1856 | 1.4E+11 | 3.0E+04 | — | 522 | 497 | 32 | 32 | 26 | 26 |
| bp_600 | 822 | 1.5E+06 | 6.6E+04 | — | 56 | 54 | 23 | 25 | 19 | 19 |
| bwm2000 | 2000 | 2.4E+05 | 2.0E+05 | — | — | — | 237 | 197 | 148 | 148 |
| meg1 | 2904 | 1.4E+12 | 3.4E+05 | 13 | 28 | 26 | 17 | 17 | 14 | 14 |
| lnsp_511 | 511 | 3.3E+15 | 5.8E+05 | — | 302 | 268 | 39 | 30 | 23 | 23 |
| hangGlider_2 | 1647 | 1.4E+09 | 1.2E+06 | — | 52 | 56 | 32 | 33 | 31 | 31 |
| tub1000 | 1000 | 1.3E+06 | 1.3E+06 | — | — | — | 354 | 254 | 114 | 114 |
| 1138_bus | 1138 | 8.6E+06 | 2.4E+06 | — | — | — | 238 | 163 | 83 | 83 |
| gre_1107 | 1107 | 3.2E+07 | 2.3E+07 | — | — | — | 244 | 185 | 75 | 75 |
| rajat19 | 1157 | 1.1E+10 | 1.4E+08 | — | — | — | — | — | 70 | 69 |
| spaceStation_7 | 1134 | 3.9E+11 | 1.7E+08 | — | — | — | — | 220 | 127 | 127 |
| bcsstk19 | 817 | 1.3E+11 | 1.2E+09 | — | — | — | — | — | 307 | 209 |
| reorientation_3 | 2513 | 1.5E+21 | 8.9E+11 | — | — | — | — | — | 144 | 188 |
| fs_760_3 | 760 | 9.8E+19 | 2.6E+12 | — | — | — | — | — | — | — |
| nnc1374 | 1374 | 3.7E+14 | 5.2E+12 | — | — | — | — | — | 85 | 92 |
| lung1 | 1650 | 5.1E+19 | 1.1E+13 | — | — | — | — | — | 27 | 27 |

and $\lambda$ is different for each variant and matrix. For the comparison to be as fair as possible, for each variant and each matrix, we have tested eight different values of $\tau$ ($10^{-10}$, $10^{-8}$, $10^{-6}$, $10^{-4}$, $10^{-3}$, $10^{-2}$, $10^{-1}$, and $5 \times 10^{-1}$) and five different values of $\lambda$ ($10^4$, $10^3$, $10^2$, $10^1$, and $10^0$), and taken the values that leads to the lowest number of iterations.

We display in Figure 5.3 the effect of the scaling on the condition numbers of the matrices with the optimal $\lambda$. We can observe that, in addition of preventing overflows and reducing underflows, this scaling also affect positively the condition numbers by reducing it on many matrices by few order of magnitude. Indeed, the scaled condition number $\kappa(A_s)$ is almost always under the original one $\kappa(A)$ and, in particular, this is this new condition number $\kappa(A_s)$ that affects the convergence conditions (5.2) and (5.3).

We present in Figure 5.4 performance profiles for $u_f = \mathrm{B}, \mathrm{H}, \mathrm{S}$, which plot the percentage $\phi$ of matrices for which a given variant converges in less than $\alpha$ times the number of LU solves required by the best variant. We also provide detailed results on a representative sample of these matrices for the case $u_f = \mathrm{B}$ in Table 5.3.

These results are in agreement with our theoretical study and with the previous experiments of section 5.5.1, and provide a confirmation of the relative robustness of each variant on a dataset of real-life applications matrices. For example, with $u_f = \mathrm{B}$, LU-IR3 only con-

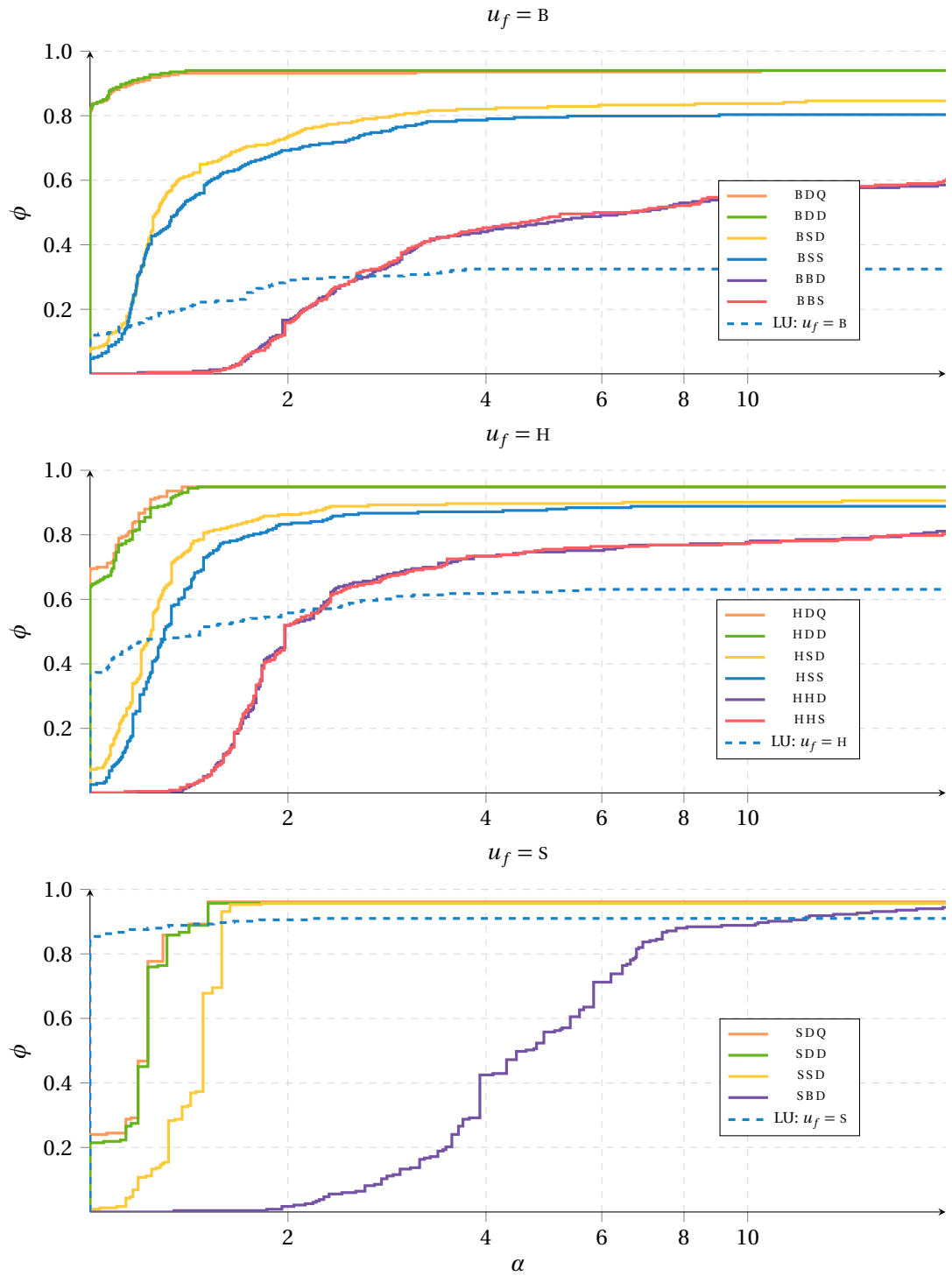



Figure 5.4: Performance profile of LU-IR3 and LU-GMRES-IR5 variants (with $u_f$, $u_g$, $u_p$ as specified in the legend) on 230 SuiteSparse matrices. $\phi$ (y-axis) indicates the percentage of matrices for which a given variant requires less than $\alpha$ (x-axis) times the number of LU solves required by the best variant.

verges for about 35% f the matrices, whereas with LU-GMRES-IR3 (BDQ variant), we are able to process about 95% of the matrices (the 5% remaining being highly ill-conditioned matrices). Crucially, the new LU-GMRES-IR5 variants (with relaxed $u_p$ and/or $u_g$) are all more robust than LU-IR3, converging for a much larger percentage of matrices. In particular, on this set of matrices, the BDD variant is as robust as BDQ and requires the same number of LU solves. Therefore, these experiments show that, in practice, we can switch from $u_p = \mathrm{Q}$ to $u_p = \mathrm{D}$ with no impact on the convergence of LU-GMRES-IR.

The other two performance profiles with $u_f = \mathrm{H}$ and S show similar trends. Unsurprisingly, as we increase the precision of the factorization, LU-IR3 is able to converge on a wider range of matrices and so the range of matrices where LU-GMRES-IR is relevant is narrower. This is especially the case for an fp32 factorization, where LU-IR3 is able to converge on almost 90% of the matrices. Note that this observation heavily depends on the dataset: for these matrices, the distribution of the $\kappa(A)$ is not uniform and is mainly concentrated between $10^3$ and $10^6$, which explains why LU-IR3 performs well with an fp32 factorization.

A performance comparison of the actual runtime of the variants is outside the scope of this chapter, but we can nevertheless extrapolate some performance trends based on Figure 5.4 and on the assumptions that (1) the LU solves dominate the overall runtime of the iterative phase of the solver; and (2) a bfloat16 LU solve is twice faster than an fp32 one, which is itself twice faster than an fp64 one. With this performance model and considering for example $u_f = \mathrm{B}$, we can expect LU-IR3 (which uses bfloat16 LU solves) to be the fastest method as long as it does not require more than twice the number of LU solves of a variant with $u_p = \mathrm{S}$, that is, for about 30% of the matrices. Similarly, the BSS variant should outperform BDD for over 70% of the matrices, making it the best variant for about $70-30 = 40\%$ of the matrices. Finally, on our set of matrices, the BDD variant never requires more LU solves than the BDQ one and therefore it should be the best variant for the remaining 30% of the matrices. A high performance implementation of LU-GMRES-IR5 is necessary to confirm these predictions, and it will be the object of chapter 6 specialized on sparse linear systems.

In Figure 5.5, we report the values of the converged forward errors of the 230 SuiteSparse matrices used in Figure 5.4 for variants with $u_f = \mathrm{H}$. Similarly as for the random dense matrices in section 5.5.1, we observe that the robustness increases as we increase the precision $u_g$ and $u_p$ inside the GMRES. For example, switching from LU-IR3 with $u_f = \mathrm{H}$ to HHS allows us to obtain a double precision accuracy on the forward error for almost all the matrices in the range $\kappa(A_s) \in [10^4, 10^8]$. Overall, our observations on real-life sparse matrices are in good agreement with what has been observed on random dense matrices and with the convergence condition (5.2). Therefore, we expect that the conclusions of section 5.5.1 would extend nicely on real-life problems, particularly on the large sparse problems from real-life and industrial applications that we will consider in chapter 6.

## 5.6 Practical advice

In this section, we propose simple practical tips to pick an algorithm among LU-IR3 and LU-GMRES-IR5 with adequate precisions $u_g$ and $u_p$. Their differences are mainly based on the degree of robustness on the condition number of the matrix $A$. Naturally, we expect

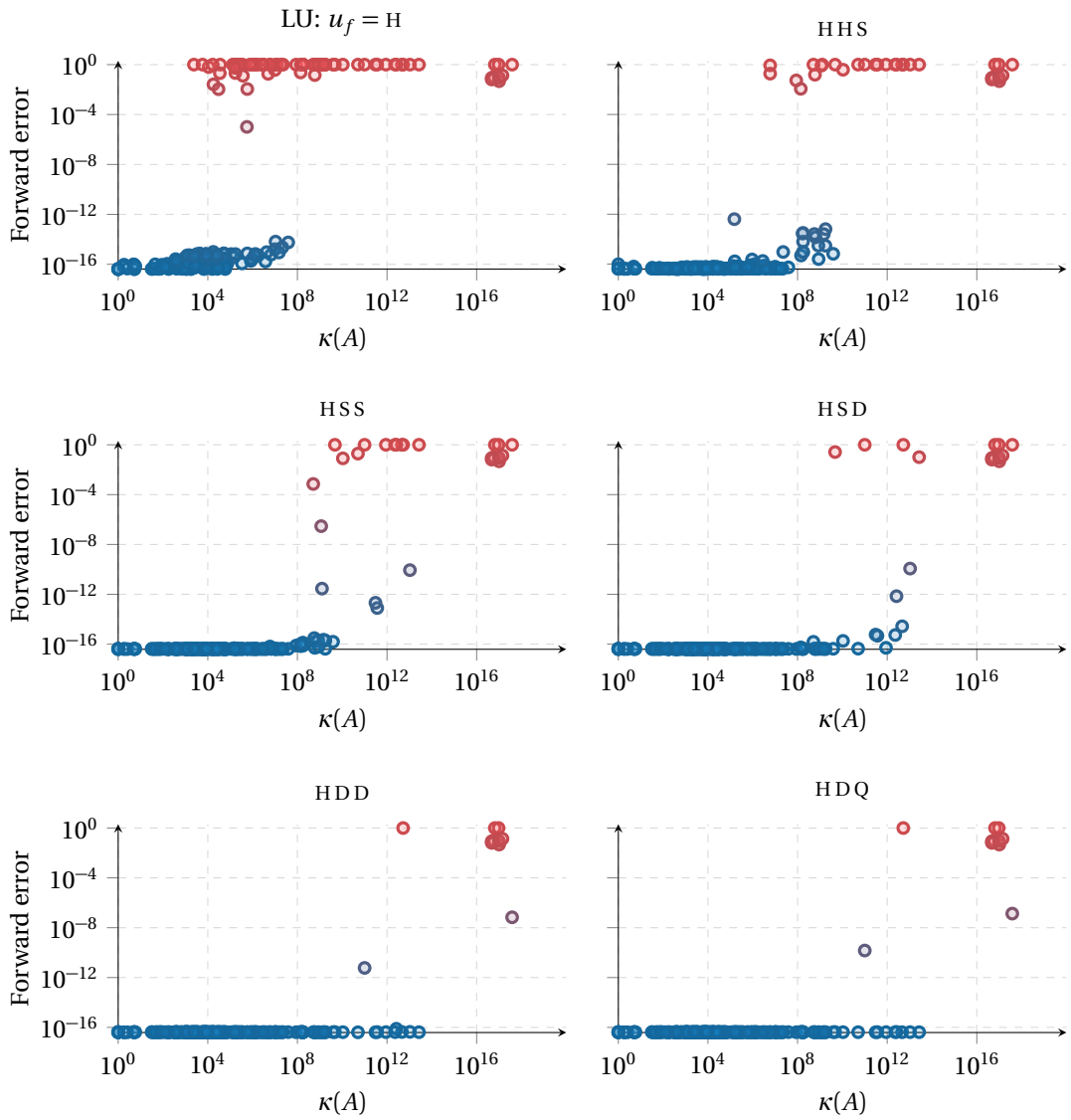



Figure 5.5: Converged forward error according to $\kappa(A)$ for LU-IR3 and LU-GMRES-IR5 variants with fixed $u_f = \mathrm{H}$ (and with $u_g$, $u_p$ as specified in the legend) on 230 SuiteSparse matrices.

generally that the higher the robustness, the lower the time and memory savings; therefore, the user should make a trade-off between the two according to his/her needs. Indeed, the cost for computing the solution of the correction equation is higher with LU-GMRES-IR5 than with LU-IR3 because, first, the inner GMRES of LU-GMRES-IR5 applies its LU solves in a higher precision $u_p \ll u_f$ and might require several iterations to converge, leading to a time increase. Second, the factors need to be cast in precision $u_p \ll u_f$, then, if the factors are copied explicitly in precision $u_p$ it leads to a substantial memory consumption increase. Similar comments can be made when we increase the precisions $u_g$ and $u_p$ in LU-GMRES-IR5; the more we increase these precisions, the more an iteration of GMRES

will cost in time and memory.

Note that if LU-IR3 does far more outer iterations than the cumulated number of GMRES iterations in LU-GMRES-IR5, LU-GMRES-IR5 might be faster than LU-IR3. This observation also applies to the precisions $u_g$ and $u_p$ in LU-GMRES-IR5; when we choose them low, it can increase the number of inner iterations significantly compared with the variant using higher $u_g$ and $u_p$. However, in practice, we observed that if $\kappa(A)$ is reasonably lower than the convergence condition of the algorithm, we can generally expect the algorithm using lower precisions to be faster.

Consequently, in order to clarify how to choose a method between LU-IR3, LU-GMRES-IR5, and standard LU direct solvers in full precision $u_f$ or $u$, here are some general tips:

- For best performance but low accuracy solution: LU solver in $u_f$.
- For best robustness but worst performance: LU solver in $u$.
- For trade-off between time and accuracy on well-conditioned problems: LU-IR3.
- For trade-off between time and accuracy on ill-conditioned problems: LU-GMRES-IR5.
- For memory consumption (in dense, ordered from best to worst): LU solver in $u_f$ < LU solver in $u$ < LU-IR3 < LU-GMRES-IR5.

In dense, iterative refinement algorithms cannot do the factorization in-place as for the LU direct solvers because they require a copy of the matrix $A$ in precision $u_r$ and, thus, they will consume more memory. Also, as proposed by Oktay and Carson [170], LU-IR3 and LU-GMRES-IR5 can be smartly combined so that we will preferably use the variant with the best performance and switch the solver (LU to GMRES) and increase the precisions ($u_g$ and $u_p$) on the go if the robustness on the conditioning is not satisfactory. Therefore, the choice between LU-IR3 and LU-GMRES-IR5 can be conveniently hidden from the user.

## 5.7 LU-GMRES-IR5 for least squares problem

Carson et al. [47] extended LU-GMRES-IR3 for the solution of the square linear system (1.1) to the solution of the least squares problem (2.25) with AQR-GMRES-IR3 represented by Algorithm 4.6 and described in section 4.5.3. This algorithm essentially replaces the LU factors with the QR factors used as preconditioners for the augmented system (4.29) whose solution is equivalent to (2.25). They proved Theorem 4.6 on the convergence of the solution of AQR-GMRES-IR3, which established comparable convergence conditions as in Theorem 4.4 for LU-GMRES-IR3.

For the sake of completeness, our aim in this section will be to adapt, in the same fashion, the result on LU-GMRES-IR5 to the least squares problem. Therefore, we propose to study *QR preconditioned GMRES-based iterative refinement on the augmented system in five precisions (AQR-GMRES-IR5)* described in Algorithm 5.2, which is AQR-GMRES-IR3 but with relaxed precisions $u_g$ and $u_p$ inside GMRES. As the limitations of LU-GMRES-IR3 mentioned in section 5.1 are inherited to AQR-GMRES-IR3 and are potentially stronger

since the factors are applied two times per iteration, the relaxation to AQR-GMRES-IR5 is as important and allows for the same benefits.

**Algorithm 5.2** AQR-GMRES-IR5

**Input:** an $m \times n$ matrix $A$ and a right-hand side $b$.
**Output:** an approximate solution to $\min_x \|b - Ax\|_2$.

1: Factorize $A = \begin{bmatrix} \widehat{Q}_1 & \widehat{Q}_2 \end{bmatrix} \begin{bmatrix} \widehat{R} \\ 0 \end{bmatrix}$. $(u_f)$
2: Solve $\widehat{R}\, x_0 = \widehat{Q}_1^T b$ for $x_0$. $(u_f)$
3: **while not** converged **do**
4: Compute $\begin{bmatrix} h_i \\ g_i \end{bmatrix} = \begin{bmatrix} b - r_i - Ax_i \\ -A^T r_i \end{bmatrix}$. $(u_r)$
5: Solve $M^{-1} A_{+,\alpha} \begin{bmatrix} d_{r_i} \\ d_{x_i} \end{bmatrix} = M^{-1} \begin{bmatrix} h_i \\ g_i \end{bmatrix}$ by GMRES at precision $u_g$ with matrix–vector products with $\widetilde{A}_+ = M^{-1} A_{+,\alpha}$ computed at precision $u_p$.
6: Compute $\begin{bmatrix} r_{i+1} \\ x_{i+1} \end{bmatrix} = \begin{bmatrix} r_i \\ x_i \end{bmatrix} + \begin{bmatrix} d_{r_i} \\ d_{x_i} \end{bmatrix}$. $(u)$
7: **end while**

We present in Theorem 5.4 the result of the error analysis of AQR-GMRES-IR5. The rest of the section will be devoted to proving this theorem.

**Theorem 5.4** (Convergence of AQR-GMRES-IR5)**.** *Let* (1.1) *be solved by AQR-GMRES-IR5 (Algorithm 5.2) using MGS-GMRES at step 5 and using the Householder method at step 1. If $u_g \geq u$, $\kappa(A) u_p < 1$, and $|\alpha| \approx \|A^\dagger\|_2^{-1}$, the forward and backward errors will reach their respective limiting accuracies*

$$p\, u_r \operatorname{cond}(A_+, z) + u \text{ (forward)} \qquad \text{and} \qquad p\, u_r + u \text{ (backward)}, \tag{5.48}$$

*provided that*

$$(u_g + u_p \kappa(A))(1 + \kappa(A)^2 u_f^2) \ll 1 \text{ (forward)} \tag{5.49}$$

*and*

$$(u_g + u_p \kappa(A))(1 + \kappa(A) u_f)\kappa(A) \ll 1 \text{ (backward)}. \tag{5.50}$$

*Proof.* In the same way as in section 5.2 we should proceed in three steps: bounding the error on forming the preconditioned right-hand side $\widehat{s}_i = \mathrm{fl}(M^{-1}[\widehat{h}_i^T \widehat{g}_i^T]^T)$, using our analysis of MGS-GMRES to prove the backward stability of the solution to the system $\widetilde{A}_+ [d_{r_i}^T d_{x_i}^T]^T = \widehat{s}_i$, and combining the two previous results to derive bounds of the type (4.4) and (4.5) for the solution of $\widetilde{A}_+ [d_{r_i}^T d_{x_i}^T]^T = M^{-1}[\widehat{h}_i^T \widehat{g}_i^T]^T$. From this, we will be able to derive the convergence conditions (5.49) and (5.50) that are expressed in terms of $\kappa(A)$.

Fortunately, much of the work has already been done in the error analysis of section 5.2. The major difficulty of the adaptation to AQR-GMRES-IR5 is to bound the error on the preconditioned matrix–vector product $z_j = \widetilde{A}_+ \widehat{v}_j$ to be able to use Theorem 5.2. Because of the complex form of the application of $\widetilde{A}_+$, which implies few different matrix operations, this task becomes relatively cumbersome. That is why, this will be the concern of the most

part of the proof, and the step we begin with.

We recall the scaled augmented matrix and its preconditioner as well as their respective inverses,

$$A_{+,\alpha} \equiv \begin{bmatrix} \alpha I & A \\ A^T & 0 \end{bmatrix}, \qquad A_{+,\alpha}^{-1} \equiv \begin{bmatrix} 0 & A^{-T} \\ A^\dagger & -\alpha A^\dagger A^{-T} \end{bmatrix}, \tag{5.51a}$$

$$M \equiv \begin{bmatrix} \alpha I & \widehat{Q}_1 \widehat{R} \\ \widehat{R}^T \widehat{Q}_1^T & 0 \end{bmatrix}, \qquad M^{-1} \equiv \begin{bmatrix} \frac{1}{\alpha}(I - \widehat{Q}_1 \widehat{Q}_1^T) & \widehat{Q}_1 \widehat{R}^{-T} \\ \widehat{R}^{-1} \widehat{Q}_1^T & -\alpha \widehat{R}^{-1} \widehat{R}^{-T} \end{bmatrix}, \tag{5.51b}$$

where $\widehat{Q}_1$ and $\widehat{R}$ are the computed QR factors of $A$ in precision $u_f$ and applied in precision $u_p$, and $A^\dagger$ and $A^{-T}$ are, respectively, the pseudo inverse of $A$ and the transpose of the pseudo inverse. Let $z_j = M^{-1} A_{+,\alpha} \widehat{v}_j$ be computed in precision $u_p$, the computed $\widehat{z}_j$ can therefore be written as

$$\widehat{z}_j = \mathrm{fl}(M^{-1} A_{+,\alpha} \widehat{v}_j) = (M^{-1} + \Delta M)(A_{+,\alpha} + \Delta A_+)\widehat{v}_j = M^{-1} A_{+,\alpha} \widehat{v}_j + f_j. \tag{5.52}$$

where $f_j$ carries the error on the product. After dropping second order terms, $f_j$ becomes

$$f_j \approx (M^{-1} \Delta A_+ + \Delta M A_{+,\alpha})\widehat{v}_j = (M^{-1} A_{+,\alpha} A_{+,\alpha}^{-1} \Delta A_+ + \Delta M M M^{-1} A_{+,\alpha})\widehat{v}_j, \tag{5.53}$$

with $\Delta M$ and $\Delta A_+$, respectively, the error for the application of the preconditioner and the error for the multiplication with the augmented matrix. Thus, we have

$$\|f_j\|_2 \lesssim (\|A_{+,\alpha}^{-1} \Delta A_+\|_F + \|\Delta M M\|_F)\|\widetilde{A}_+\|_F \|\widehat{v}_j\|_2, \tag{5.54}$$

which gives a bound on the error of the product of the form (5.5) satisfying the condition for the use of Theorem 5.2. However, for this bound to be useful we need to express the errors $\Delta A_+$ and $\Delta M$ and evaluate the quantities $\|A_{+,\alpha} \Delta A_+\|_F$ and $\|\Delta M M\|_F$.

Before, we need to introduce few more error terms. The application of $A_{+,\alpha}$ and $M^{-1}$ is composed of matrix–vector products and triangular solves with $A$, $A^T$, $\widehat{Q}_1$, $\widehat{Q}_1^T$, $\widehat{R}$ and $\widehat{R}^T$ generating, respectively, the backward errors $\Delta A$, $\Delta A'$, $\Delta Q$, $\Delta Q'$, $\Delta R$ and $\Delta R'$. From Theorems 2.1 and 2.3 we can bound these errors as such

$$|\Delta A| \le \gamma_n^p |A|, \quad |\Delta A'| \le \gamma_m^p |A^T|, \quad |\Delta Q| \le \gamma_n^p |\widehat{Q}_1|, \tag{5.55a}$$

$$|\Delta Q'| \le \gamma_m^p |\widehat{Q}_1^T|, \quad |\Delta R| \le \gamma_n^p |\widehat{R}|, \quad |\Delta R'| \le \gamma_n^p |\widehat{R}^T|. \tag{5.55b}$$

We begin by expressing $\Delta A_+$. If we note $\widehat{v}_j^T = [\widehat{v}_{j,1}^T \widehat{v}_{j,2}^T]$ where $\widehat{v}_{j,1} \in \mathbb{R}^m$ and $\widehat{v}_{j,2} \in \mathbb{R}^n$, the product $y_j = A_{+,\alpha} \widehat{v}_j$ produces the following computed quantity

$$\widehat{y}_j = \mathrm{fl}(A_{+,\alpha} \widehat{v}_j) = \begin{bmatrix} \mathrm{fl}(\alpha \widehat{v}_{j,1} + A \widehat{v}_{j,2}) \\ \mathrm{fl}(A^T \widehat{v}_{j,1}) \end{bmatrix} = \begin{bmatrix} \alpha \widehat{v}_{j,1} + A \widehat{v}_{j,2} \\ A^T \widehat{v}_{j,1} \end{bmatrix} + \begin{bmatrix} \Delta A_{+,1,1} & \Delta A_{+,1,2} \\ \Delta A_{+,2,1} & 0 \end{bmatrix} \begin{bmatrix} \widehat{v}_{j,1} \\ \widehat{v}_{j,2} \end{bmatrix}, \tag{5.56}$$

where we can identify the error $\Delta A_+ = [\Delta A_{+,i,j}]$ for $i = 1,2$ and $j = 1,2$ made of the errors on the applications of $A$ and $A^T$ and the errors on the vector addition and scalar product;

we obtain the following bounds

$$\|\Delta A_{+,1,1}\|_F \lesssim |\alpha|\widetilde{\gamma}^p_{m^{1/2}}, \quad \|\Delta A_{+,1,2}\|_F \lesssim \widetilde{\gamma}^p_n\|A\|_F, \quad \|\Delta A_{+,2,1}\|_F \leq \gamma^p_m\|A\|_F. \tag{5.57}$$

We now express $\Delta M$. As it would unnecessarily burden the reading, we will not consider the errors coming from vector additions and scalar products which do not affect the final conclusion. If we note $\widehat{y}_j^T = [\widehat{y}_{j,1}^T \widehat{y}_{j,2}^T]$ where $\widehat{y}_{j,1} \in \mathbb{R}^m$ and $\widehat{y}_{j,2} \in \mathbb{R}^n$, the product $z_j = M^{-1}\widehat{y}_j$ produces the following computed quantity

$$\widehat{z}_j = \mathrm{fl}(M^{-1}\widehat{y}_j) = \begin{bmatrix} \mathrm{fl}\left(\frac{1}{\alpha}(I - \widehat{Q}_1\widehat{Q}_1^T)\widehat{y}_{j,1} + \widehat{Q}_1\widehat{R}^{-T}\widehat{y}_{j,2}\right) \\ \mathrm{fl}\left(\widehat{R}^{-1}\widehat{Q}_1^T\widehat{y}_{j,1} - \alpha\widehat{R}^{-1}\widehat{R}^{-T}\widehat{y}_{j,2}\right) \end{bmatrix} \tag{5.58a}$$

$$= \begin{bmatrix} \frac{1}{\alpha}(I - \widehat{Q}_1\widehat{Q}_1^T)\widehat{y}_{j,1} + \widehat{Q}_1\widehat{R}^{-T}\widehat{y}_{j,2} \\ \widehat{R}^{-1}\widehat{Q}_1^T\widehat{y}_{j,1} - \alpha\widehat{R}^{-1}\widehat{R}^{-T}\widehat{y}_{j,2} \end{bmatrix} + \begin{bmatrix} \Delta M_{1,1} & \Delta M_{1,2} \\ \Delta M_{2,1} & \Delta M_{2,2} \end{bmatrix} \begin{bmatrix} \widehat{y}_{j,1} \\ \widehat{y}_{j,2} \end{bmatrix}, \tag{5.58b}$$

where we need to analyse each matrix operation kernels to determine $\Delta M = [\Delta M_{i,j}]$ for $i = 1,2$ and $j = 1,2$. We have

$$\mathrm{fl}(\widehat{Q}_1\widehat{Q}_1^T\widehat{y}_{j,1}) = (\widehat{Q}_1 + \Delta Q)(\widehat{Q}_1^T + \Delta Q')\widehat{y}_{j,1} \approx \widehat{Q}_1\widehat{Q}_1^T\widehat{y}_{j,1} + (\Delta Q\widehat{Q}_1^T + \widehat{Q}_1\Delta Q')\widehat{y}_{j,1}, \tag{5.59a}$$

$$\mathrm{fl}(\widehat{Q}_1\widehat{R}^{-T}\widehat{y}_{j,2}) = (\widehat{Q}_1 + \Delta Q)(\widehat{R}^T + \Delta R')^{-1}\widehat{y}_{j,2} \approx (\widehat{Q}_1 + \Delta Q)(\widehat{R}^{-T} - \widehat{R}^{-T}\Delta R'\widehat{R}^{-T})\widehat{y}_{j,2} \tag{5.59b}$$

$$\approx \widehat{Q}_1\widehat{R}^{-T}\widehat{y}_{j,2} + (\Delta Q\widehat{R}^{-T} - \widehat{Q}_1\widehat{R}^{-T}\Delta R'\widehat{R}^{-T})\widehat{y}_{j,2}, \tag{5.59c}$$

$$\mathrm{fl}(\widehat{R}^{-1}\widehat{Q}_1^T\widehat{y}_{j,1}) = (\widehat{R} + \Delta R)^{-1}(\widehat{Q}_1^T + \Delta Q')\widehat{y}_{j,1} \approx (\widehat{R}^{-1} - \widehat{R}^{-1}\Delta R\widehat{R}^{-1})(\widehat{Q}_1^T + \Delta Q')\widehat{y}_{j,1} \tag{5.59d}$$

$$\approx \widehat{R}^{-1}\widehat{Q}_1^T\widehat{y}_{j,1} + (\widehat{R}^{-1}\Delta Q' - \widehat{R}^{-1}\Delta R\widehat{R}^{-1}\widehat{Q}_1^T)\widehat{y}_{j,1}, \tag{5.59e}$$

$$\mathrm{fl}(\widehat{R}^{-1}\widehat{R}^{-T}\widehat{y}_{j,2}) = (\widehat{R} + \Delta R)^{-1}(\widehat{R}^T + \Delta R')^{-1}\widehat{y}_{j,2} \tag{5.59f}$$

$$\approx (\widehat{R}^{-1} - \widehat{R}^{-1}\Delta R\widehat{R}^{-1})(\widehat{R}^{-T} - \widehat{R}^{-T}\Delta R'\widehat{R}^{-T})\widehat{y}_{j,2} \tag{5.59g}$$

$$\approx \widehat{R}^{-1}\widehat{R}^{-T}\widehat{y}_{j,2} - (\widehat{R}^{-1}\Delta R\widehat{R}^{-1}\widehat{R}^{-T} + \widehat{R}^{-1}\widehat{R}^{-T}\Delta R'\widehat{R}^{-T})\widehat{y}_{j,2}, \tag{5.59h}$$

where (5.59a) corresponds to the computation of two matrix–vector products, (5.59c) of one triangular solve and one matrix–vector product, (5.59e) of one matrix–vector product and one triangular solve, and (5.59h) of two triangular solves, and where we can identify

$$\Delta M_{1,1} \equiv \frac{1}{\alpha}(\Delta Q\widehat{Q}_1^T + \widehat{Q}_1\Delta Q'), \quad \Delta M_{1,2} \equiv \Delta Q\widehat{R}^{-T} - \widehat{Q}_1\widehat{R}^{-T}\Delta R'\widehat{R}^{-T}, \tag{5.60}$$
$$\Delta M_{2,1} \equiv \widehat{R}^{-1}\Delta Q' - \widehat{R}^{-1}\Delta R\widehat{R}^{-1}\widehat{Q}_1^T, \quad \Delta M_{2,2} \equiv \alpha(\widehat{R}^{-1}\Delta R\widehat{R}^{-1}\widehat{R}^{-T} + \widehat{R}^{-1}\widehat{R}^{-T}\Delta R'\widehat{R}^{-T}).$$

We now bound $\|A_{+,\alpha}^{-1}\Delta A_+\|_F$ with the scaling parameter set to $\alpha \approx \|A^\dagger\|_2^{-1}$. We have from (5.51) and (5.57)

$$\|A_{+,\alpha}^{-1}\Delta A_+\|_F = \left\| \begin{bmatrix} A^{-T}\Delta A_{+,2,1} & 0 \\ A^\dagger\Delta A_{+,1,1} - \alpha A^\dagger A^{-T}\Delta A_{+,2,1} & A^\dagger\Delta A_{+,1,2} \end{bmatrix} \right\|_F \tag{5.61a}$$

$$\leq \|A^{-T}\Delta A_{+,2,1}\|_F + \|A^\dagger\Delta A_{+,1,1} - \alpha A^\dagger A^{-T}\Delta A_{+,2,1}\|_F + \|A^\dagger\Delta A_{+,1,2}\|_F \tag{5.61b}$$

$$\lesssim \sqrt{n}\widetilde{\gamma}^p_m\|A^\dagger\|_F\|A\|_F. \tag{5.61c}$$

Next, we bound $\|\Delta M M\|_F$ with (5.51), (5.55) and (5.60), in particular we can use Theorems 2.7 and 2.8 and drop second order terms to state $\widehat{Q}_1^T\widehat{Q}_1 \approx I$, $\|\widehat{Q}_1\|_F \approx \sqrt{n}$ and $\widehat{Q}\widehat{R} \approx A$, also we need to use the following observation

$$\||\widehat{Q}_1||\widehat{R}|\|_F \leq \sqrt{n}\|\widehat{Q}_1^T\widehat{Q}_1\widehat{R}\|_F \lesssim n\|A\|_F; \tag{5.62}$$

we have

$$\begin{aligned}
\|\Delta M M\|_F &\approx \left\| \begin{bmatrix} 2\Delta Q\widehat{Q}_1^T + \widehat{Q}_1\Delta Q' - \widehat{Q}_1\widehat{R}^{-T}\Delta R'\widehat{Q}_1^T & \frac{1}{\alpha}(\Delta Q\widehat{R} + \widehat{Q}_1\Delta Q'\widehat{Q}_1\widehat{R}) \\ \alpha(\widehat{R}^{-1}\Delta Q' + \widehat{R}^{-1}\widehat{R}^{-T}\Delta R'\widehat{Q}_1^T) & \widehat{R}^{-1}\Delta Q'\widehat{Q}_1\widehat{R} - \widehat{R}^{-1}\Delta R \end{bmatrix} \right\|_F && \text{(5.63a)} \\
&\leq \|2\Delta Q\widehat{Q}_1^T + \widehat{Q}_1\Delta Q' - \widehat{Q}_1\widehat{R}^{-T}\Delta R'\widehat{Q}_1^T\|_F + \|\frac{1}{\alpha}(\Delta Q\widehat{R} + \widehat{Q}_1\Delta Q'\widehat{Q}_1\widehat{R})\|_F && \text{(5.63b)} \\
&\quad + \|\alpha(\widehat{R}^{-1}\Delta Q' + \widehat{R}^{-1}\widehat{R}^{-T}\Delta R'\widehat{Q}_1^T\|_F + \|\widehat{R}^{-1}\Delta Q'\widehat{Q}_1\widehat{R} - \widehat{R}^{-1}\Delta R\|_F \\
&\lesssim n\widetilde{\gamma}_m^p\|A^\dagger\|_F\|A\|_F. && \text{(5.63c)}
\end{aligned}$$

Finally, we can bound the error on the preconditioned matrix–vector product, from (5.54), we can say

$$\|f_j\|_2 \lesssim n\widetilde{\gamma}_m^p\kappa_F(A)\|\widetilde{A}_+\|_F\|\widehat{v}_j\|_2 \leq \sqrt{mn}\,n\widetilde{\gamma}_m^p\kappa_\infty(A)\|\widetilde{A}_+\|_F\|\widehat{v}_j\|_2. \tag{5.64}$$

This bound is nearly equivalent to the bound on the matrix–vector product (5.25) of the analysis of LU-GMRES-IR5 in section 5.2. Thus, the rest of the proof is almost identical as what has been done in sections 5.2.2 and 5.2.3 and comes straightforwardly.

Although, two steps might not be direct. First, the bound on the error of the computed preconditioned right-hand side (5.20) in section 5.2.2 can be adapted as such

$$s_i = M^{-1}(M^{-1} + \Delta M)^{-1}\widehat{s}_i = \widehat{s}_i - \Delta M M\widehat{s}_i, \tag{5.65}$$

where $s_i = M^{-1}[\widehat{h}_i^T\,\widehat{g}_i^T]^T$ and $\widehat{s}_i = \mathrm{fl}(M^{-1}[\widehat{h}_i^T\,\widehat{g}_i^T]^T) = (M^{-1} + \Delta M)[\widehat{h}_i^T\,\widehat{g}_i^T]^T$, giving the equivalent bound

$$\|s_i - \widehat{s}_i\|_\infty \leq \sqrt{m+n}\|\Delta M M\|_F\|\widehat{s}_i\|_\infty \lesssim \sqrt{(m+n)mn}\,n\widetilde{\gamma}_m^p\kappa_\infty(A)\|\widehat{s}_i\|_\infty \tag{5.66}$$

that we obtain by using (5.63). Second, to express our convergence conditions on the forward and backward errors in terms of $\kappa(A)$ as for (5.36) and (5.40), we need bounds on $\kappa(\widetilde{A}_+)$ and $\|\widetilde{A}_+\|$, they can be retrieved from [47, eq. (3.1) and (3.7)] which gives

$$\begin{aligned}
\|\widetilde{A}_+\|_\infty &\lesssim 1 + 2m\sqrt{n}\widetilde{\gamma}_{mn}^f\kappa_\infty(A) && \text{(5.67a)} \\
\kappa_\infty(\widetilde{A}_+) &\lesssim (1 + 2m\sqrt{n}\widetilde{\gamma}_{mn}^f\kappa_\infty(A))^2. && \text{(5.67b)}
\end{aligned}$$

□

## 5.8 Conclusion

In this chapter, we have addressed the solution of linear systems of equations by means of LU preconditioned GMRES-based iterative refinement in mixed precision. Our baseline is the work by Carson and Higham [45], who proposed a method, LU-GMRES-IR3, that employs up to three precisions and in which a mixed precision preconditioned MGS-GMRES method is used for solving the correction equation in order to converge on ill-conditioned problems. In its original form, this method requires twice the working precision when applying the preconditioned matrix to a vector, which can be expensive, especially when the working precision is double precision. By relaxing the assumptions on the precision of the operations within the GMRES solver, we have proposed a method with up to five different precisions in play, which we call LU-GMRES-IR5. We extended the rounding error analysis of Carson and Higham to cover LU-GMRES-IR5. As a key component of this analysis, we extended the work of Paige et al. [175] to prove the backward stability of a mixed precision MGS-GMRES method. Based on this result, we derived conditions on $\kappa(A)$ that guarantee the convergence of LU-GMRES-IR5.

Our results show that LU-GMRES-IR5 can accurately and reliably solve relatively ill-conditioned problems in potentially lower time and memory than LU-GMRES-IR3 thanks to the use of lower precision arithmetic in the GMRES iterations. Although the combined use of multiple precisions results in hundreds or thousands of different variants (depending on how many precisions are available on the system), we provided a list of rules that we used to identify a much smaller set of practical interests. It includes variants for which only an experimental evaluation was available prior to this work. We also discussed the use of a stopping criterion inside MGS-GMRES that can help to limit the cumulated number of GMRES iterations, but which readily affect the convergence conditions of LU-GMRES-IR5. Then, we presented an extensive experimental analysis of the LU-GMRES-IR5 method on randomly generated matrices and matrices from real-life applications. The experimental results are in good agreement with our theoretical analysis and show that LU-GMRES-IR5 is a robust and versatile method for solving linear systems of equations. In addition, we provided practical advice that aimed at clarifying how to pick among LU-IR3 and LU-GMRES-IR5 and their different combinations of precisions for a given application. Finally, we showed that in the same way as LU-GMRES-IR3, LU-GMRES-IR5 can be extended for the solution of the least squares problem by solving the augmented system through a QR factorization in a potentially low precision.

Most of this chapter is the object of the preprint *"Five-Precision GMRES-based iterative refinement"* (Amestoy et al. [18]).

# 6 Iterative refinement for sparse approximate factorizations

Most recent work on new mixed precision iterative refinement variants to enhance in speed and accuracy the solution of linear systems has focused on dense systems. In this chapter, we investigate the potential of mixed precision iterative refinement to enhance methods for sparse systems based on common approximate sparse factorizations.

In doing so, we first present in section 6.1 the use of numerical approximations and low precision for reducing the computational cost of LU sparse solvers. In section 6.2, we talk about the differences between iterative refinement for sparse and dense systems. We then develop a new error analysis for LU-IR3 and LU-GMRES-IR5 under a general model of LU factorization that accounts for the approximation methods typically used by modern sparse solvers in section 6.3. We finally provide a detailed performance analysis of both the execution time and memory consumption of different algorithms based on a selected set of iterative refinement variants and approximate sparse factorizations in section 6.4. Our performance study uses the multifrontal solver MUMPS, which can exploit block low-rank (BLR) factorization and static pivoting. We evaluate the performance of the algorithms on large sparse problems coming from a variety of real-life and industrial applications, showing that the proposed approach can lead to considerable reductions in both time and memory consumption.

## 6.1 Reducing the computational cost of sparse direct solvers

Direct methods for the solution of sparse linear systems (1.1) are widely used and generally appreciated for their robustness and accuracy. These desirable properties, however,

come at the cost of high operational complexity, high memory consumption, and limited scalability on large scale parallel supercomputers compared with iterative solvers. In order to mitigate some of these limitations, we can use various approaches to trade off some accuracy and robustness for lower complexity and memory consumption or better computational efficiency and scalability. These include the use of numerical approximations (see section 2.2.4), such as low-rank approximations or relaxed numerical pivoting. Furthermore, the recent appearance and increasingly widespread adoption of low precision arithmetics (see section 2.1.3) offer additional opportunities for reducing the computational cost of sparse direct solvers. However, these approaches often lead to a poor-quality solution that can be unsatisfactory for many applications. In that context, using iterative refinement algorithms applying lightweight refinement steps to recover the lost accuracy is natural.

In the 80s–90s, the fixed precision form of iterative refinement, that is, using the same precision for every operation, has been studied extensively with sparse factorization for correcting instability arising from approximation techniques (see section 3.3). For example, it has been combined with some less stable pivoting strategies (Li and Demmel [142], Dongarra et al. [66], Gill et al. [81]) or with drop strategies (Arioli et al. [26], Zlatev [217]). In the late 2000s, a two-precision LU-based iterative refinement has also been used by Buttari et al. [42], Baboulin et al. [29], Hogg and Scott [123] to accelerate sparse direct solvers through single precision factorization for double precision accuracy. Most of the efforts to improve sparse direct solvers with iterative refinement ended with these last studies. Several variants of the novel mixed precision iterative refinement algorithms LU-IR3, LU-GMRES-IR3, and our new LU-GMRES-IR5 presented in chapter 5 have been implemented on modern hardware, notably supporting half precision such as GPUs, and have been shown to be highly successful at accelerating the solution of dense linear systems (Haidar et al. [101; 103; 102], Lopez and Mary [153]). However, they have not been considered for improving sparse direct solvers.

In this chapter, we aim to fill this gap by investigating the potential of mixed precision arithmetic to accelerate the solution of large sparse linear systems by combining state-of-the-art iterative refinement variants with state-of-the-art sparse factorizations considering the use of numerical approximations. There are two main components in our study. First, we tackle this subject from a theoretical point of view and extend the error analyses of LU-IR3 and LU-GMRES-IR5 to the case of approximate factorizations which better fit a common use of sparse direct solvers. Second, we address the issues related to a high performance parallel implementation of mixed precision iterative refinement for sparse linear systems and provide an in-depth analysis of experimental results obtained on real-life and industrial problems. For the parallel implementation of the sparse direct solver, we use the multifrontal approach described in section 2.2.3.4. While our conclusions on the execution time of LU-IR3 and LU-GMRES-IR5 should be valid to any sparse solvers, some conclusions on the memory consumption might be specific to the multifrontal approach. It is because they might be related to the use of active memory that, for example, supernodal approaches do not use.

To reduce the computational cost of the sparse direct solvers, we will use in our exper-

iments either single precision factorization, BLR, and static pivoting numerical approximations described in section 2.2.4, or most importantly, a mix of all of them to achieve significant time and memory reductions while still achieving high accuracy. Our new theoretical analysis of LU-IR3 and LU-GMRES-IR5 covers all the variants we will use.

## 6.2 Specific features of iterative refinement with sparse direct solvers

The most important difference between iterative refinement for dense and sparse linear systems lies in its practical cost. As explained in section 2.2.3.2, a key property of sparse direct solvers is that they generate *fill-in*, that is, the LU factors of $A$ are typically much denser than $A$ itself. Therefore, as the size of the matrix grows, the storage for $A$ becomes negligible compared with that for its LU factors. Note that this still holds for data sparse solvers despite the reduced asymptotic complexity. For example, as explained in section 2.2.4.1, BLR sparse direct solvers reduce the size of the LU factors to at best $O(n \log n)$ entries, but with the constants hidden in the big $O$, the size of the LU factors typically remains several orders of magnitude larger than that of the original matrix.

In Algorithm 6.1 we display the different complexities in dense and in sparse at each step of LU-IR3 and LU-GMRES-IR5 to better compare how the structural sparsity and the fill-in readily affect the practical cost of these iterative refinement variants. Hence, in orange we show the complexities for the solution of dense linear systems, and in blue we show the sparse counterparts for 3D problems (see Table 2.2).

**Algorithm 6.1** LU-IR and LU-GMRES-IR: complexities **Dense** VS **Sparse**

**Input:** an $n \times n$ matrix $A$ and a right-hand side $b$.
**Output:** an approximate solution to $Ax = b$.

| | | | |
|---|---|---|---|
| 1: Factorize $A = \widehat{L}\widehat{U}$. | $\mathcal{O}(n^3)$ | $\mathcal{O}(n^2)$ | $(u_f)$ |
| 2: Solve $\widehat{L}\widehat{U}x_0 = b$ for $x_0$. | $\mathcal{O}(n^2)$ | $\mathcal{O}(n^{4/3})$ | $(u_f)$ |
| 3: **while not** converged **do** | | | |
| 4: Compute $r_i = b - Ax_i$ | $\mathcal{O}(n^2)$ | $\mathcal{O}(n)$ | $(u_r)$ |
| 5: Solve $Ad_i = r_i$ by LU solver or by GMRES preconditioned with the LU factors. | $\mathcal{O}(n^2)$ | $\mathcal{O}(n^{4/3})$ | $(u_s)$ |
| 6: Compute $x_{i+1} = x_i + d_i$ | $\mathcal{O}(n)$ | $\mathcal{O}(n)$ | $(u)$ |
| 7: **end while** | | | |

A crucial consequence of the existence of fill-in is that, with a lower precision factorization ($u_f > u$), LU-IR3 can achieve not only higher speed but also lower memory consumption than a standard sparse direct solver run entirely in precision $u$. This is because the LU factors, which account for most of the memory footprint, need be stored only in precision $u_f$. We emphasize that LU-IR3 does not reduce the memory footprint in the case of dense linear systems, since in this case the matrix $A$ and the LU factors require the same number of entries, and $A$ must be stored at least in precision $u$. In fact, since a copy of $A$ must be kept in addition to its LU factors, iterative refinement for dense linear systems

actually consumes more memory than a standard in-place LU factorization in precision $u$; this has already been remarked in section 5.6.

Similar comments apply to the cost of the matrix–vector products $Ax_i$ in the computation of the residual (step 4 of Algorithm 6.1). Whereas for a dense matrix this cost is comparable with that of the LU triangular solves, when the matrix is sparse it becomes, most of the time, negligible. In particular, this means that we have more flexibility to choose the precision $u_r$, especially when the target precision $u$ is double precision: performing the matrix–vector products in high precision ($u_r = u^2$) does not necessarily have a significant impact on the performance, even for arithmetics usually not supported in hardware, such as quadruple precision (fp128). This is illustrated and further discussed in section 6.4.

To summarize, LU-IR3 is attractive for sparse linear systems because it can lead to memory gains and because the most costly step of the iterative phase, that is, the triangular solves with the LU factors, is carried out in the low precision $u_f$.

Unfortunately these last points are not guaranteed to be met when using LU-GMRES-IR5 because the triangular solves have to be applied in precision $u_p < u_f$. As a consequence the cost of the iterations is higher and the factors need to be cast in precision $u_p$ (more is said in section 6.4.3 about different options for casting the factors). As an extreme case, using LU-GMRES-IR3 by setting $u_g = u$ and $u_p = u^2$ as originally proposed by Carson and Higham [45] would make the iterative phase significantly more costly compared with the factorization. The issue is even worse since if the working precision is fp64, we should apply the LU solve in fp128, which is not supported by many popular parallel sparse solvers. Therefore, our relaxed version LU-GMRES-IR5 proposed in chapter 5, which allows for setting $u_p > u^2$, is even more relevant in the sparse case. This is why, in this chapter, we focus on variants where $u_p \geq u$.

Finally, another specific feature of sparse direct solvers is related to pivoting. While partial pivoting is the most common approach for dense linear systems, sparse direct solvers commonly use other approaches that better preserve the sparsity of the LU factors and limit the communications in parallel contexts as explained in section 2.2.3.7. While partial pivoting guarantees the practical stability of the solver (see section 2.2.1.2), these methods do not. However, combined with iterative refinement, a sparse direct solver can achieve a satisfactory stability under suitable conditions as explained in sections 3.3 and 4.2, and combining the two is thus a natural match.

## 6.3 Error analysis of iterative refinement with a general approximate factorization

Carson and Higham [44; 45] and the content of chapter 5 provide error analyses for LU-IR3 and LU-GMRES-IR5, under the assumption that the LU factors are computed with standard GEPP. However, as explained above, modern sparse direct solvers often depart from this assumption, because they typically do not implement partial pivoting, and because they take advantage of data sparsity resulting in numerical approximations. This affects the error analyses of LU-IR3 and LU-GMRES-IR5 and the conditions under which they are guaranteed to converge. For this reason, in this section, we propose a new error analysis

under a general approximate factorization model. Our model can be applied to at least BLR, static pivoting, and their combined use, and we expect it to cover several other approximate approaches used in direct solvers. Moreover, although in this chapter we are particularly motivated by sparse applications, the results of this section carry over to the dense case as well.

The section is therefore articulated as follows. We first propose our approximate factorization model in section 6.3.1 with which we carry out the error analyses of LU-IR3 and LU-GMRES-IR5 in sections 6.3.2 and 6.3.3. In section 6.3.4 we summarize and interpret the results of these two analyses. Finally, in section 6.3.5 we apply our results for BLR, static pivoting, and BLR combined with static pivoting factorizations.

### 6.3.1 Approximate factorization model

Essentially, the purpose of this section is to revisit with numerical approximations the results of Theorem 4.3 and Theorem 5.1 which are providing the respective convergence conditions and limiting accuracies for the forward and backward errors of LU-IR3 and LU-GMRES-IR5. As the use of approximations in the LU solver only affect steps 1 and 5 of Algorithm 6.1, only the convergence conditions shall be re-determined. To do so we need to express the accuracy of the computed solution $\widehat{d}_i$ by the solver at step 5 to be able to rewrite (4.9) and (4.16), then, we can apply Theorems 4.1 and 4.2 to obtain the specialized conditions on $\kappa(A)$ for LU-IR3 and LU-GMRES-IR5 to converge.

To carry out this new error analysis we propose a general model of an approximate LU factorization which covers different well-known numerical approximations. Specifically, our model makes the following two assumptions:

- The approximate factorization performed at precision $u_x$ provides computed LU factors of a square nonsingular matrix $A$ satisfying

$$A = \widehat{L}\widehat{U} + \Delta A^{(1)}, \quad |\Delta A^{(1)}| \lesssim c_1 \epsilon \|A\|_\infty e\, e^T + c_2 u_x |\widehat{L}||\widehat{U}|, \tag{6.1}$$

  where $c_1$ and $c_2$ are constants related to the dimension of the problem, and where $e$ is the vector of ones and $\epsilon$ is a parameter quantifying the quality of the approximate factorization.

- The triangular solve $\widehat{T}y = v$, where $\widehat{T}$ is one of the approximately computed LU factors, performed at precision $u_x$ provides a computed solution $\widehat{y}$ satisfying

$$(\widehat{T} + \Delta\widehat{T})\widehat{y} = v + \Delta v, \quad |\Delta\widehat{T}| \lesssim c_3 u_x |\widehat{T}|, \quad |\Delta v| \lesssim c_4 u_x |v|, \tag{6.2}$$

  where $c_3$ and $c_4$ are constants related to the dimension of the problem.

From (6.1) and (6.2), it follows that the solution of the linear system $Ay = v$ provides a computed solution $\widehat{y}$ satisfying

$$\begin{aligned} &(A + \Delta A^{(2)})\widehat{y} = v + \Delta v, \\ &|\Delta A^{(2)}| \lesssim c_1 \epsilon \|A\|_\infty e\, e^T + (c_2 + 2c_3) u_x |\widehat{L}||\widehat{U}|, \end{aligned} \tag{6.3}$$

$$|\Delta v| \lesssim c_4 u_x (|v| + |\widehat{L}||\widehat{U}||\widehat{y}|).$$

Note that, in order for our approximate factorization model to be more general, we have not enforced a particular sparsity structure on the term $c_1\epsilon\|A\|_\infty e e^T$, which is in fact dense. The extension of the analysis to backward error with a sparse structure, such as in Arioli et al. [26], is outside our scope.

### 6.3.2 Error analysis for LU-IR3

We want to determine the convergence conditions for LU-IR3 (Algorithm 4.2). We can apply Theorems 4.1 and 4.2 respectively for the convergence of the forward and normwise backward errors of the computed solution of (1.1), and for both we need respectively a bound on the forward and backward errors of the computed solution $\widehat{d}_i$ of the correction equation $Ad_i = \widehat{r}_i$. Note that for LU-IR3, the factorization (6.1) and the LU solves (6.2) are performed in precision $u_f$.

Considering the solution of the linear system $Ad_i = \widehat{r}_i$, (6.3) yields

$$d_i - \widehat{d}_i = A^{-1}\Delta A^{(2)}\widehat{d}_i - A^{-1}\Delta\widehat{r}_i. \tag{6.4}$$

Taking norms, we obtain

$$\frac{\|d_i - \widehat{d}_i\|_\infty}{\|\widehat{d}_i\|_\infty} \lesssim (c_1\epsilon + c_4 u_f)\|A^{-1}\|_\infty\|A\|_\infty + (c_2 + 2c_3 + c_4)u_f\|A^{-1}\|_\infty\||\widehat{L}||\widehat{U}|\|_\infty. \tag{6.5}$$

Using Theorem 2.5

$$\||\widehat{L}||\widehat{U}|\|_\infty \le (1 + 2(n^2 - n)\rho_n)(\|A\|_\infty + \|\Delta A^{(1)}\|_\infty), \tag{6.6}$$

where $\rho_n$ is the growth factor for $A + \Delta A^{(1)}$. Dropping second order terms finally gives

$$\frac{\|d_i - \widehat{d}_i\|_\infty}{\|\widehat{d}_i\|_\infty} \lesssim f(n)(\epsilon + \rho_n u_f)\kappa_\infty(A). \tag{6.7}$$

In the same fashion we can show that

$$\|\widehat{r}_i - A\widehat{d}_i\|_\infty \lesssim f(n)(\epsilon + \rho_n u_f)(\|A\|_\infty\|\widehat{d}_i\|_\infty + \|\widehat{r}_i\|_\infty). \tag{6.8}$$

Dropping constants and applying Theorems 4.1 and 4.2 using (6.7) and (6.8) guarantees that as long as

$$(\rho_n u_f + \epsilon)\kappa(A) \ll 1, \tag{6.9}$$

the forward and the normwise backward errors of the system $Ax = b$ will converge to their limiting accuracies (4.23).

As a check, if we set $\epsilon = 0$ (no approximation) and drop $\rho_n$ (negligible element growth), we recover (4.24).

Before commenting in section 6.3.4 on the significance of these new LU-IR3 convergence conditions, we first similarly derive the LU-GMRES-IR5 conditions.

### 6.3.3 Error analysis for LU-GMRES-IR5

We now determine the convergence conditions of LU-GMRES-IR5 (Algorithm 5.1). We proceed similarly as for LU-IR3, seeking bounds on the forward and backward errors of the computed solution $\widehat{d}_i$ of the correction equation $Ad_i = \widehat{r}_i$. One difference lies in the fact that the GMRES solver is applied to the preconditioned system $\widetilde{A}d_i = \widehat{s}_i$ where $\widetilde{A} = \widehat{U}^{-1}\widehat{L}^{-1}A$ and $s_i = \widehat{U}^{-1}\widehat{L}^{-1}\widehat{r}_i$. This analysis follows closely the analysis of section 5.2 made in the previous chapter, so we mainly focus on the changes coming from the use of a more general approximate factorization model and refer the reader to this section for the full details. For the sake of readability, we do not keep track of the constants $c_k$ for $k = 1, 2, \ldots$ and we gather them in $f(n)$ whose precise value is not of interest.

We first need to bound the error introduced in forming the preconditioned right-hand side $s_i$ in precision $u_p$. Computing $s_i$ implies two triangular solves (6.2) which differ from the original GMRES-IR analysis by having an error term on the right-hand side. Adapting (5.17) and (5.20) with (6.1) and (6.2), supposing $\kappa(A)\rho_n u_p < 1$ to drop second order terms, and using (6.6) provides the bound

$$\|s_i - \widehat{s}_i\|_\infty \lesssim f(n)u_p\rho_n\kappa_\infty(A)\|s_i\|_\infty. \tag{6.10}$$

As in section 5.2 we compute the error of the computation of the preconditioned matrix–vector product $z_j = \widetilde{A}\widehat{v}_j$ in order to use Theorem 5.2. We obtain $z_j$ through a standard matrix–vector product with $A$ followed by two triangular solves (6.2) with $\widehat{L}$ and $\widehat{U}$. The computed $\widehat{z}_j$ satisfies $\widehat{z}_j = z_j + f_j$, where $f_j$ carries the error of the computation. With a very similar reasoning as for deriving (5.25) and considering our new assumptions, we obtain the bound

$$\|f_j\|_2 \lesssim f(n)u_p\rho_n\kappa_\infty(A)\|\widetilde{A}\|_F\|\widehat{v}_j\|_2. \tag{6.11}$$

Apart from the constants and the presence of the growth factor $\rho_n$ which can be arbitrarily large without assumptions on the pivoting, (6.11) and (6.10) are similar to (5.25) and (5.20) and meet the assumptions of Theorem 5.2 which can be used to compute a bound of $\|s_i - \widetilde{A}\widehat{d}_i\|_\infty$.

We can finally bound the normwise relative backward error of the system $\widetilde{A}\widehat{d}_i = s_i$ (5.29) by

$$\frac{\|s_i - \widetilde{A}\widehat{d}_i\|_\infty}{\|\widetilde{A}\|_\infty\|\widehat{d}_i\|_\infty + \|s_i\|_\infty} \lesssim f(n)(u_g + u_p\rho_n\kappa_\infty(A)) \tag{6.12}$$

and the relative error of the computed $\widehat{d}_i$ (5.30) by

$$\frac{\|d_i - \widehat{d}_i\|_\infty}{\|d_i\|_\infty} \lesssim f(n)(u_g + u_p\rho_n\kappa_\infty(A))\kappa_\infty(\widetilde{A}). \tag{6.13}$$

In addition, the backward error of the original correction equation $Ad_i = \widehat{r}_i$ can be bounded using $\widehat{r}_i - A\widehat{d}_i = \widehat{L}\widehat{U}(s_i - \widetilde{A}\widehat{d}_i)$ and (6.12), yielding

$$\|\widehat{r}_i - A\widehat{d}_i\|_\infty \lesssim f(n)(u_g + u_p\rho_n\kappa_\infty(A))(\|\widetilde{A}\|_\infty\|A\|_\infty\|\widehat{d}_i\|_\infty + \kappa_\infty(A)\|\widehat{r}_i\|_\infty). \tag{6.14}$$

It is essential to study the conditioning of the preconditioned matrix $\widetilde{A}$ in order to express the convergence conditions according to the conditioning of the original matrix $\kappa(A)$. Using the same reasoning as for (5.35) we obtain

$$\|\widetilde{A}\|_\infty \lesssim 1 + f(n)(u_f \rho_n \kappa_\infty(A) + \epsilon \kappa_\infty(A)),$$
$$\kappa_\infty(\widetilde{A}) \lesssim (1 + f(n)(u_f \rho_n \kappa_\infty(A) + \epsilon \kappa_\infty(A)))^2.$$

Dropping constants and applying Theorems 4.1 and 4.2 using (6.13) and (6.14) guarantees that as long as

$$(u_g + u_p \rho_n \kappa(A))(1 + (u_f \rho_n + \epsilon)^2 \kappa(A)^2) \ll 1 \qquad \text{(forward error)}, \tag{6.15}$$
$$(u_g + u_p \rho_n \kappa(A))(1 + (u_f \rho_n + \epsilon)\kappa(A))\kappa(A) \ll 1 \qquad \text{(backward error)}, \tag{6.16}$$

the forward and the normwise backward errors of the system $Ax = b$ will converge to their limiting accuracies (5.1).

As a check, with $\epsilon = 0$ and dropping $\rho_n$, we recover (5.2) and (5.3).

### 6.3.4 Summary of the error analysis and interpretation

We summarize the analysis in the following two theorems which extend Theorems 4.3 and 5.1 to the use of approximate factorization.

**Theorem 6.1** (Convergence of approximate LU-IR3)**.** *Let* (1.1) *be solved by Algorithm 4.2 (Algorithm 4.2) using an approximate LU factorization satisfying* (6.1)–(6.2)*. If* $\kappa(A)(u_f \rho_n + \epsilon) \geq u$*, then the forward and backward errors will reach their respective limiting accuracies* (4.23) *provided that*

$$(u_f \rho_n + \epsilon)\kappa(A) \ll 1 \text{ (forward and backward).} \tag{6.17}$$

**Theorem 6.2** (Convergence of approximate LU-GMRES-IR5)**.** *Let* (1.1) *be solved by Algorithm 5.1 (Algorithm 5.1) using an approximate LU factorization satisfying* (6.1)–(6.2) *and using MGS-GMRES at step 5. If* $u_g \geq u$ *and* $\kappa(A)\rho_n u_p < 1$*, the forward and backward errors will reach their respective limiting accuracies* (4.23) *provided that*

$$(u_g + u_p \rho_n \kappa(A))(1 + (u_f \rho_n + \epsilon)^2 \kappa(A)^2) \ll 1 \text{ (forward)} \tag{6.18}$$

*and*

$$(u_g + u_p \rho_n \kappa(A))(1 + (u_f \rho_n + \epsilon)\kappa(A))\kappa(A) \ll 1 \text{ (backward).} \tag{6.19}$$

We now comment on the significance of these results. Compared with the original convergence conditions (4.24), (5.2), and (5.3), the new conditions of Theorems 6.1 and 6.2 include two new terms. The first is the growth factor $\rho_n$ that, without any assumption on the pivoting strategy, cannot be assumed to be small. This shows that a large growth factor can prevent iterative refinement from converging. The second is $\epsilon$ which reflects the degree of approximation used by the factorization. The terms $\rho_n u_f + \epsilon$ show that we can expect the approximation to impact the convergence of iterative refinement when $\epsilon \gtrsim \rho_n u_f$ (ignoring

the difference between the constants in front of each term). It is important to note that the instabilities introduced by element growth and numerical approximations combine additively, rather than multiplicatively (there is no $\epsilon\rho_n$ term). In particular, this means that the usual wisdom that it is not useful to use a very high precision for an approximate factorization ($u_f \ll \epsilon$) is no longer true in presence of large element growth. This is a key property that we confirm experimentally in section 6.4.

### 6.3.5 Convergence conditions for BLR and static pivoting

We now apply the above analysis to the use of BLR approximations (see section 2.2.4.1) and static pivoting (see section 2.2.4.2).

The BLR format approximates the blocks of the matrix by replacing them by low-rank matrices. The ranks are determined by a threshold, $\tau_b$ in this manuscript, that controls the accuracy of the approximations. Higham and Mary [118] have carried out error analysis of the BLR LU factorization and obtained a backward error bound of order $\tau_b\|A\|+u_f\|\widehat{L}\|\|\widehat{U}\|$, see Theorem 2.10. One issue is that their analysis derives normwise bounds, whereas our model (6.1) and (6.2) requires componentwise bounds. However, we have checked that, at the price of slightly larger constants by about a factor $O(n^{3/4})$ and a more complicated analysis, analogous componentwise bounds can be obtained. Therefore, using the componentwise version of Theorem 2.10 for (6.1) and of Theorem 2.11 for (6.2), we conclude that Theorems 6.1 and 6.2 apply with $\epsilon = \tau_b$.

We now turn to static pivoting, assuming a strategy that replaces pivots smaller in absolute value than $\tau_s\|A\|_\infty$ by $\tau_s\|A\|_\infty$, where $\tau_s$ is a threshold that controls the accuracy of the factorization. With such a strategy we are actually solving a perturbed system

$$Mx = b, \quad M = A + E, \tag{6.20}$$

where $E$ is a diagonal matrix having nonzero entries equal to $\tau_s\|A\|_\infty$ in the positions corresponding to pivots that were replaced. By applying Theorem 2.2 to (6.20) we meet the condition (6.1) with $\epsilon = \tau_s$, while condition (6.2) is met since the triangular solves are standard. Therefore Theorems 6.1 and 6.2 apply with $\epsilon = \tau_s$.

Finally, we can also derive convergence conditions for the case where BLR and static pivoting are combined. This amounts to using BLR approximations on the perturbed system (6.20), and so Theorems 6.1 and 6.2 apply with $\epsilon = \tau_b + \tau_s$.

Note that Theorems 6.1 and 6.2 can cover other numerical approximations, such as dropping (Arioli et al. [26], Zlatev [217]) or ILU (Saad [184]).

## 6.4 Performance analysis

We have implemented a selected set of iterative refinement variants and we analyze in this section their practical performance for the solution of large scale, real-life and industrial sparse problems on parallel computers.

We first describe our implementation details and our experimental setting in sections 6.4.1 and 6.4.2. In particular, we further discuss the issue of casting the factors for LU-GMRES-

IR5 in section 6.4.3, where we explain that the implementation choices for the cast have significant impacts on the memory consumption and the execution time. Next, we compare five variants with the case of a plain fp64 factorization plus solve in section 6.4.4, where we carry out detailed analyses of the time and memory performance. In section 6.4.5, we investigate the use of four iterative refinement variants combined with BLR, static pivoting, and BLR with static pivoting. Finally, we study the scalability of these algorithms in section 6.4.7.

### 6.4.1 Implementation details

To perform our experiments we implemented both LU-IR3 and LU-GMRES-IR5 for their execution on parallel architectures. In the following we describe our implementation choices.

For the sparse LU factorization and LU triangular solves, we rely on the MUMPS solver (Amestoy et al. [17; 12]), which implements the multifrontal method described in section 2.2.3. It must be noted that most of our analysis readily applies to other sparse factorization approaches, such as the right- or left-looking supernodal method used, for example, in SuperLU (Demmel et al. [59]), PaStiX (Hénon et al. [125]), or PARDISO (Schenk et al. [185]). The only exception is the memory consumption analysis in section 6.4.4.2, where we rely on features of the multifrontal method, namely, the use of active memory (see section 2.2.3.4). The default pivoting strategy used in the MUMPS solver is threshold partial pivoting (Duff et al. [72]) which provides great stability; alternatively, static pivoting (as described in section 2.2.4.2) can be used, where possible, to improve performance. MUMPS also implements the BLR factorization method described in section 2.2.4.1; for a detailed description of the BLR feature of MUMPS, we refer to the papers of Amestoy et al. [17; 14].

For the GMRES solver, we have used an in-house implementation of the unrestarted MGS-GMRES method described in section 2.3.1. This code does not use MPI parallelism, but is multithreaded; as a result, all computations are performed on a single node, using multiple cores, except for the solves in the preconditioning which are operated through a call to the corresponding MUMPS routine which benefits from MPI parallelism. This also implies that the original system matrix and all the necessary vectors (including the Krylov basis) are centralized on MPI rank zero. We use the GMRES stopping criterion described in section 5.4, that is, the GMRES method is stopped when the scaled residual falls below a prescribed threshold $\tau_g$.

In the LU-GMRES-IR5 case, the solves require the LU factors to be in a different precision than what was computed by the factorization, that is, $u_f \neq u_p$. Two options are possible to handle this requirement. The first is to make an explicit copy of the factors by casting the data into precision $u_p$, which is our choice; the second is to make the solve operations blockwise, as is commonly done to take advantage of BLAS-3 operations, and cast the blocks on the fly using temporary storage as in the work of Anzt et al. [24]. We further discuss our choice for the cast of the factors in section 6.4.3.

For the symmetric matrices, we use the $LDL^T$ factorization. It must be noted that the matrix–vector product is not easily parallelizable when a compact storage format is used for symmetric matrices (such as one that stores only the upper or lower triangular part); for

this reason, we choose to store symmetric matrices with a non-compact format in order to make the residual computation more efficiently parallelizable.

The code implementing the methods has been written in Fortran 2003, supports real and complex arithmetics, and supports both multithreading (through OpenMP) and MPI parallelism (through MUMPS). The results presented below were obtained with MUMPS version 5.4.0; the default settings were used except we used the advanced multithreading option of L'Excellent and Sid-Lakhdar [141]. We used the Metis (Karypis [132]) tool version 5.1.0 for computing the fill-reducing permutation. BLAS and LAPACK routines are from the Intel Math Kernel Library version 18.2 and the Intel C and Fortran compilers version 18.2 were used to compile our code as well as the necessary third party packages. The code was compiled with the "flush to zero" option to avoid inefficient computations on sub-normal numbers; this issue is discussed by Zounon et al. [218]. Since commonly available BLAS libraries do not support quadruple precision arithmetic, we had to implement some operations (copy, norms) by taking care of multithreading them.

### 6.4.2 Experimental setting

Throughout our experiments we analyze several variants of iterative refinement that use different combinations of precisions and different kinds of factorization, with and without approximations such as BLR or static pivoting.

In all experiments, the working precision is set to double ($u = \mathrm{D}$) and GMRES is used in fixed precision ($u_g = u_p$) for a reason explained below. The factorization precision $u_f$, the residual precision $u_r$, and the precisions inside GMRES $u_g$ and $u_p$ may vary according to the experiments. Alongside the text, we define an iterative refinement variant with the solver employed (LU or GMRES) and the set of precisions $u_f$, $u$, and $u_r$ (and $u_g$, $u_p$ if GMRES is the solver used). If the solver employed is LU we refer to it as an LU-IR3 variant and if it is GMRES we call it a LU-GMRES-IR5 variant. We use the symbols from Table 2.1: S, D, and Q to refer to single, double, and quadruple precision arithmetic. We compare the iterative refinement variants to a standard double precision direct solver, namely, MUMPS, which we refer to as DMUMPS (Double precision MUMPS).

The values of the BLR threshold $\tau_b$ and the static pivoting threshold $\tau_s$ are specified alongside the text. For simplicity we set $\tau_g$, the threshold used to stop GMRES, to $10^{-6}$ in all the experiments, even though it could be tuned on a case by case basis for optimized performance.

We do not cover all combinations of precisions of LU-IR3 and LU-GMRES-IR5; rather, we focus our study on a restricted number of combinations of $u_f$, $u_g$, and $u_p$, all meaningful by the rules of section 5.3, and where their convergence conditions and limiting accuracies (without approximations) can be found in Tables 5.1 and 5.2. This is motivated by several reasons.

- Hardware support for half precision is still limited and the MUMPS solver on which we rely for this study does not currently support its use for the factorization; this prevents us from experimenting with $u_f = \mathrm{H}$.

- Setting $u_p = \mathrm{Q}$ might lead to excessively high execution time and memory consumption. In addition, it has been noticed in the numerical experiments on LU-GMRES-IR5 in section 5.5 that in practice this brings only a marginal improvement in the convergence compared with the case $u_p = \mathrm{D}$ on a wide set of real life problems.

- In our experiments we rarely observed the Krylov basis to exceed more than a few dozen vectors except in section 6.4.5 for very high thresholds $\tau_b$ and $\tau_s$. Hence setting $u_g > u_p$ to reduce the memory footprint associated with the Krylov basis is not a priority for this study and we focused on the case $u_g = u_p$.

In sparse direct solvers, the factorization is commonly preceded by a so called analysis step to prepare the factorization (see section 2.2.3.3). We do not report results on this step since:

- Its behavior is independent of the variants and precisions chosen.

- It can be performed once and reused for all problems that share the same structure.

- The fill-reducing ordering can be more efficiently computed when the problem geometry is known (which is the case in numerous applications).

- The efficiency of this step is very much implementation-dependent.

All the experiments were run on the Olympe supercomputer of the CALMIP supercomputing center of Toulouse, France. It is composed of 360 bi-processors nodes equipped with 192GB of RAM and 2 Intel Skylake 6140 processors (2.3Ghz, 18 cores) each. All experiments were done using 18 threads per MPI process because this was found to be the most efficient combination. Depending on the matrix, we use 2 or 4 MPI processes (that is, 1 or 2 nodes) for the problem to fit in memory; the number of MPI processes for each matrix is specified in Table 6.1 and is the same for all the experiments, except for the experiments of sections 6.4.3 and 6.4.7.

Table 6.1 shows the matrices coming from the SuiteSparse Matrix Collection (Davis and Hu [55]) (not bold) and industrial applications provided by industrial partners (bold) that were used for our experiments. These matrices are chosen such that a large panel of applications and a large range of condition numbers are covered. The data reported in the last three columns of the table are computed by the MUMPS solver with the settings described above. As MUMPS applies a scaling for numerical stability on the input matrix, the displayed condition number is therefore the one of the scaled matrix.

In all tests the right-hand side vector was set to $b = Ax$ with a generated $x$ vector having all its components set to 1, which also served as the reference solution to compute the forward error. Note that, in a real context, the true solution is not known; without access to the forward error to stop the algorithm, the stopping criteria reviewed in section 4.6 can be used.

We give a short description of the matrices provided by our industrial partners:

- **ElectroPhys10M**: Cardiac electrophysiology model (Niederer et al. [164]).

Table 6.1: Set of matrices from SuiteSparse and industrial applications used in our experiments. $n$ is the dimension; NNZ the number of nonzeros in the matrix; Arith. the arithmetic of the matrix (R: real, C: complex); Sym. the symmetry of the matrix (1: symmetric, 0: general); MPI the number of MPI processes used for the experiments with this matrix; $\kappa(A)$ the condition number of the matrix; Fact. flops the number of flops required for the factorization; Slv. flops the number of flops required for one LU solve.

| ID | Name | $n$ | NNZ | Arith. | Sym. | MPI | $\kappa(A)$ | Fact. (flops) | Slv. (flops) |
|---|---|---|---|---|---|---|---|---|---|
| 1 | **ElectroPhys10M** | 1.0E+07 | 1.4E+08 | R | 1 | 4 | 1E+01 | 3.9E+14 | 8.6E+10 |
| 2 | ss | 1.7E+06 | 3.5E+07 | R | 0 | 2 | 1E+04 | 4.2E+13 | 1.2E+10 |
| 3 | nlpkkt80 | 1.1E+06 | 2.9E+07 | R | 1 | 2 | 2E+04 | 1.8E+13 | 7.4E+09 |
| 4 | Serena | 1.4E+06 | 6.4E+07 | R | 1 | 2 | 2E+04 | 2.9E+13 | 1.1E+10 |
| 5 | Geo_1438 | 1.4E+06 | 6.3E+07 | R | 1 | 2 | 6E+04 | 1.8E+13 | 1.0E+10 |
| 6 | Chevron4 | 7.1E+05 | 6.4E+06 | C | 0 | 2 | 2E+05 | 2.2E+10 | 1.6E+08 |
| 7 | ML_Geer | 1.5E+06 | 1.1E+08 | R | 0 | 2 | 2E+05 | 4.3E+12 | 4.1E+09 |
| 8 | Transport | 1.6E+06 | 2.4E+07 | R | 0 | 2 | 3E+05 | 1.1E+13 | 5.2E+09 |
| 9 | Bump_2911 | 2.9E+06 | 1.3E+08 | R | 1 | 2 | 7E+05 | 2.0E+14 | 3.9E+10 |
| 10 | **DrivAer6M** | 6.1E+06 | 5.0E+07 | R | 1 | 2 | 9E+05 | 6.5E+13 | 2.6E+10 |
| 11 | vas_stokes_1M | 1.1E+06 | 3.5E+07 | R | 0 | 2 | 1E+06 | 1.5E+13 | 6.3E+09 |
| 12 | Hook_1489 | 1.5E+06 | 6.1E+07 | R | 1 | 2 | 2E+06 | 8.3E+12 | 6.2E+09 |
| 13 | Queen_4147 | 4.1E+06 | 3.3E+08 | R | 1 | 2 | 4E+06 | 2.7E+14 | 5.7E+10 |
| 14 | dielFilterV2real | 1.2E+06 | 4.8E+07 | R | 1 | 2 | 6E+06 | 1.1E+12 | 2.3E+09 |
| 15 | Flan_1565 | 1.6E+06 | 1.2E+08 | R | 1 | 2 | 1E+07 | 3.9E+12 | 6.2E+09 |
| 16 | **tminlet3M** | 2.8E+06 | 1.6E+08 | C | 0 | 4 | 3E+07 | 1.1E+14 | 2.1E+10 |
| 17 | **perf009ar** | 5.4E+06 | 2.1E+08 | R | 1 | 2 | 4E+08 | 1.9E+13 | 1.9E+10 |
| 18 | Pflow_742 | 7.4E+05 | 3.7E+07 | R | 1 | 2 | 3E+09 | 1.4E+12 | 2.1E+09 |
| 19 | Cube_Coup_dt0 | 2.2E+06 | 1.3E+08 | R | 1 | 2 | 3E+09 | 9.9E+13 | 2.7E+10 |
| 20 | **elasticity-3d** | 5.2E+06 | 1.2E+08 | R | 1 | 2 | 4E+09 | 1.5E+14 | 5.2E+10 |
| 21 | fem_hifreq_circuit | 4.9E+05 | 2.0E+07 | C | 0 | 2 | 4E+09 | 4.3E+11 | 7.6E+08 |
| 22 | **lfm_aug5M** | 5.5E+06 | 3.7E+07 | C | 1 | 4 | 6E+11 | 2.2E+14 | 4.7E+10 |
| 23 | Long_Coup_dt0 | 1.5E+06 | 8.7E+07 | R | 1 | 2 | 6E+12 | 5.2E+13 | 1.7E+10 |
| 24 | **CarBody25M** | 2.4E+07 | 7.1E+08 | R | 1 | 2 | 9E+12 | 9.6E+12 | 2.6E+10 |
| 25 | **thmgas** | 5.5E+06 | 3.7E+07 | R | 0 | 4 | 8E+13 | 1.1E+14 | 3.5E+10 |

- **DrivAer6M**: Incompressible CFD, pressure problem, airflow around an automobile (Theissen et al. [202]).
- **tminlet3M**: Noise propagation in an airplane turbine (Antwerpen et al. [20]).
- **perf009ar**: Elastic design of a pump subjected to a constant interior pressure. It was provided by Électricité de France (EDF), who carries out numerical simulations for structural mechanics applications using Code_Aster[1].
- **elasticity-3d**: Linear elasticity problem applied on a beam combosed of hereogenous materials (Al Daas et al. [8]).
- **lfm_aug5M**: Electromagnetic modelling, stabilized formulation for the low frequency solution of Maxwell's equation (Streich et al. [196]).

[1]http://www.code-aster.org

- **CarBody25M**: structural mechanics, car body model.
- **thmgas**: coupled thermal, hydrological, and mechanical problem.

### 6.4.3 Cast of the factors

With the LU-GMRES-IR5 algorithm, the LU solves must be applied at a higher precision $u_p$ than the precision $u_f$ at which the factors are computed. Two options are possible to handle this requirement: an explicit copy of the factors used to apply the subsequent LU solves or casting the factors on the fly at each LU solve application. In the following, we cover both and explain why we choose to keep the explicit copy solution for the experiments of this chapter.

*6.4.3.1 MUMPS data structure.* We begin by giving a few general details about the data structure of the MUMPS software. To every problem is associated a MUMPS data structure which contains the original information on the problem, the different variables, pointers, and arrays. With the default MUMPS options, the factors and the active memory are stored in a single array workspace statically allocated. Some other vectors live outside of this static workspace, for example, the scaling vectors, the fill permutations, the mapping, or the structure of the assembly tree. However, they are, for the most part, symbolic and can be considered negligible.

To implement LU-GMRES-IR5 with, for example, $u_f = \text{S}$ and $u_p = \text{D}$, we need two MUMPS data structures that will share the same workspace; one is in single precision and serves for the factorization, the other is in double precision and serves for the preconditioning. When the single precision instance of MUMPS finishes the factorization, the factors are cast from single to double precision to apply the forward substitution and the backward substitution. Other parts of the structure that does not need to be cast, such as the integers, are shared through pointers.

Note that the active memory part, which is composed of the contribution blocks created and consumed as we browse the assembly tree (see section 2.2.3.4), is only used during the factorization and is not useful for the solve operation. Hence, it does not need to be cast. In addition, note that for some advanced options of MUMPS, the factors (and the active memory) are not, or not entirely, contained in the single static workspace, but are spread over smaller dynamically allocated arrays. For example, it is the case when BLR is activated.

*6.4.3.2 Explicit copy.* The explicit copy approach fully casts the factors once and for all from precision $u_f$ to precision $u_p$. Hence, at the end of the cast, the full LU factors in precision $u_p$ are accessible. In order to avoid having at the same time in memory the factors in $u_f$ and $u_p$, this cast can be done in-place at the cost of a moderate reduction of its parallelization. To achieve this, an array of the size of the factors in precision $u_p$ is allocated. This array is used to compute the factors in precision $u_f$, and then to cast them in-place through a recursive process from precision $u_f$ to precision $u_p$. We describe such a recursive cast in Figure 6.1 for $u_p = u_f^2$; for example, single precision factorization and double precision preconditioner.

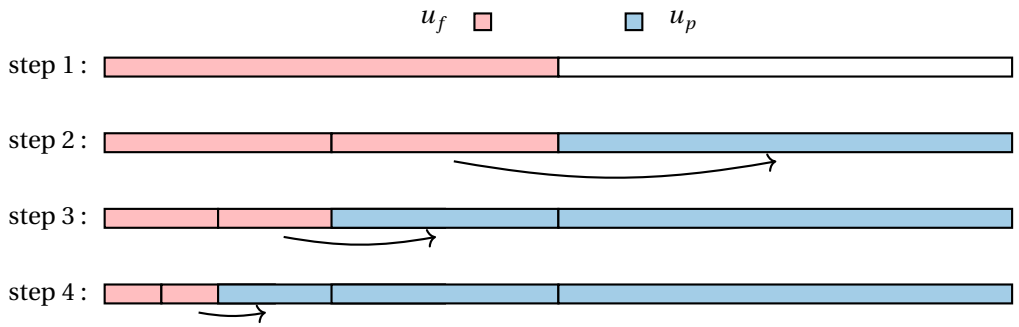



Figure 6.1: Recursive approach of cast in-place of the factors contained in a contiguous array from precision $u_f$ to precision $u_p$ where $u_p = u_f^2$.

At step 1 of Figure 6.1, the factors have been computed and are in memory in precision $u_f$; half of the array remains empty. At step 2, half of the factors are cast in precision $u_f$ to $u_p$, which fills the remaining empty space. At step 3, the memory of the previous half of the factors that have been cast can be reused to cast a quarter of the factors. This recursive process is repeated until the whole factors have been cast. While these steps cannot be executed concurrently, parallelism can be exploited within each step.

We now describe how we implemented the cast in-place in the MUMPS solver. When the single precision factorization finishes, the factors are cast in-place from single to double, erasing then the active memory overhead and the single precision factors in the shared workspace. To properly allocate the shared workspace, we need to consider the two possible scenarios illustrated in Figure 6.2: 1. the memory peak consumption happens during the single precision factorization because the active memory overhead consumes more memory than the single precision factors. 2. the peak consumption happens during the refinement steps because the double precision factors require more memory than the single precision factorization peak (i.e., factors plus active memory overhead in single). In both cases, the MUMPS estimates of the size of the factors and the active memory overhead provided after the analysis phase (see section 2.2.3.3) can be used to allocate the shared workspace. These estimates are accurate if numerical pivoting is not too heavy; however, if it is not the case, it may produce too much additional fill-in that would increase memory consumption. This is why safety relaxation memory is allocated to avoid memory shortage.

With some advanced MUMPS options, such as the activation of the BLR, the factors might not be (fully) contained in the workspace but are stored in smaller arrays that have been dynamically allocated. As the portion of the factors contained in each of these arrays is relatively negligible compared with the whole factors in memory, we can simply cast and deallocate each array one by one to prevent an increase in memory consumption.

*6.4.3.3 Cast on the fly.* The cast on the fly approach temporarily casts the factors' entries when they are used during the solve operation. It means that the factors in precision $u_p$ never entirely exist in memory; the only persistent version of the factors is in precision $u_f$. When used with LU-GMRES-IR5, it guarantees that the memory peak will happen during the factorization in precision $u_f$ if the Krylov subspace is not too large. In particular, it

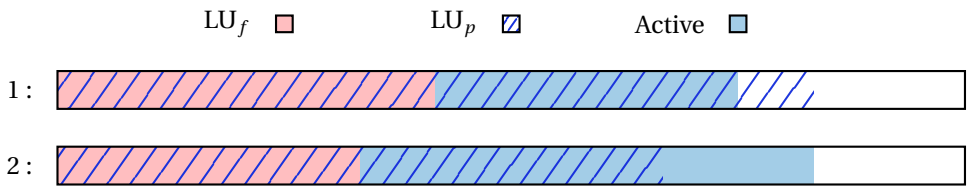



Figure 6.2: Two possibilities for the peak consumption in the memory workspace of one MPI process for LU-GMRES-IR5 with $u_p = u_f^2$. 1 : peak is reached during the factorization in precision $u_f$. 2 : peak is reached during the refinement steps. We represent the portion of the memory occupied by the factors in precision $u_f$ and $u_p$ and by the active memory overhead. The remaining white part corresponds to unused allocated memory due to safety relaxation.

means that the algorithm would have approximately the same memory consumption as LU-IR3. Consequently, cast on the fly can be very attractive from a memory standpoint; however, it requires at every LU solve a cast of the whole factors.

We implemented an early prototype of cast on the fly inside MUMPS working with only 1 MPI. Contrary to the cast in-place, with the cast on the fly, the MUMPS workspace is not fully cast from precision $u_f$ to precision $u_p$. Instead, in our implementation, at each solve operation, the factors are cast front-by-front during forward and backward substitutions such that no more than one front is in precision $u_p$ simultaneously in memory.

*6.4.3.4 Comparison.* We want to compare the execution time of the two implementations of the MUMPS LU solve based, respectively, on the explicit copy and the cast on the fly approaches presented in sections 6.4.3.2 and 6.4.3.3. In the case of the explicit copy, LU-GMRES-IR5 applies a single cast in-place after the factorization and multiple standard MUMPS LU solves during the refinement steps. On the other hand, in the case of the cast on the fly, LU-GMRES-IR5 does not fully copy the factors and applies multiple MUMPS LU solves with cast on the fly during the refinement steps. Hence, in the first case, the cost of the copy is paid once, and in the second, it is paid at every iteration of GMRES.

In Table 6.2, we report execution times for modified MUMPS solves using cast on the fly and standard MUMPS solves applied after one cast in-place of the factors. We can clearly see that the overhead of casting on the fly during the solve is huge compared with standard solve. For example, the cast on the fly overhead (i.e., solve with cast on the fly minus standard solve) on Serena is 6.9s and is even higher than the cast in-place execution time, which is 6.5s. It should be noted, however, that all the matrices in Table 6.2 are symmetric, which is a disadvantage for the cast on the fly approach. Indeed, for a LDL$^T$ factorization, the cast on the fly will need to cast two times the L factor at each solve, one for the forward substitution and one for the backward substitution, while the cast in-place casts it only once. We expect things to be slightly better for cast on the fly with unsymmetric LU factorization, where both methods have to cast each factor L and U.

From this comparison, we conclude that with our actual implementation of cast on the fly, the time overhead is too costly and not affordable in the context of our study. This is why we choose to use the explicit copy approach.

Table 6.2: Comparison of the execution time between cast on the fly solve and standard solve applicable after the full cast in-place of the factors. The solves are applied in precision $u_p = \text{D}$ and the factors are computed in precision $u_f = \text{S}$. The number of MPI is fixed to 1, and the number of threads is fixed to 18.

| Matrices | Solve (Cast on the fly) | Solve (Standard) | Cast (In-place) |
|---|---|---|---|
| Serena | 14.88 | 7.98 | 6.5 |
| Geo_1438 | 4.47 | 2.55 | 1.7 |
| Bump_2911 | 4.14 | 2.39 | 1.9 |
| Hook_1498 | 3.41 | 2.15 | 1.15 |
| Queen_4147 | 20.8 | 12.37 | 11.2 |
| Flan_1565 | 3.43 | 1.85 | 1.4 |
| PFlow_742 | 1.43 | 0.92 | 0.47 |
| Cube_Coup_dt0 | 9.91 | 5.24 | 4.32 |
| Long_Coup_dt0 | 6.21 | 3.4 | 2.84 |

However, note that our cast on the fly implementation is quite naive, mainly because a whole front is cast at once, which probably does not fit in cache, so we cannot benefit from data locality. A blockwise approach for casting the fronts would probably be better because it allows us to design block sizes that can be entirely stored in cache. However, the existing implementation of the MUMPS sparse solve operation is not blockwise and would require heavy developments inside the software structure. Consequently, this is a solution that we keep for future work. In certain contexts and with the proper implementation, cast on the fly can actually be made very efficient; it has been explored widely by Anzt et al. [24], Flegar et al. [73].

### 6.4.4 Performance of LU-IR and LU-GMRES-IR using standard LU factorization

In this first set of experiments we analyze the time and memory savings that different iterative refinement variants without approximate factorization are able to achieve and we show how the specific features discussed in section 6.2, the choice of a multifrontal solver, and the matrix properties can affect the performance of the method.

In Table 6.3 we present the execution time and memory consumption of five iterative refinement variants and DMUMPS for the set of the test matrices of Table 6.1. We classify the variants into two categories; in the first, we have variants that achieve a forward error equivalent to that obtained with the double precision direct solver DMUMPS (the ones using $u_r = \text{D}$) and, in the second, those whose forward error is of order $10^{-16}$, the double precision unit roundoff (the ones using $u_r = \text{Q}$). Actually, for the first category, LU-IR3 and LU-GMRES-IR5 can provide a better accuracy on the solution than DMUMPS, which is why we stop their iterations when they reach a forward error of the same order as the solution obtained with DMUMPS. We denote by a "—" the failure of a method to converge. For each matrix, we highlight in bold the execution time and memory consumption that do not exceed by more than 10% the best execution time or memory consumption.

Table 6.3: Execution time (in seconds) and memory consumption (in GBytes) of IR variants and DMUMPS for the set of matrices listed in Table 6.1. The solver and the precisions $u_f$, $u_r$, $u_g$, and $u_p$ are specified in the table for each IR variant, $u$ is fixed to $u = \mathrm{D}$.

| Solver | DMUMPS | LU | GMRES | LU | LU | GMRES | DMUMPS | LU | GMRES | LU | LU | GMRES |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $u_f$ | | S | S | D | S | S | | S | S | D | S | S |
| $u_r$ | | D | D | Q | Q | Q | | D | D | Q | Q | Q |
| $u_p$=$u_g$ | | — | D | — | — | D | | — | D | — | — | D |
| ID | Time eq. DMUMPS (s) | | | Time eq. $10^{-16}$ (s) | | | Mem eq. DMUMPS (GB) | | | Mem eq. $10^{-16}$ (GB) | | |
| 1 | 265.2 | **154.0** | **166.5** | 269.4 | **155.9** | **168.2** | 272.0 | **138.0** | 171.3 | 272.0 | **138.0** | 173.5 |
| 2 | 52.7 | **31.7** | **33.4** | 53.7 | **33.3** | **36.3** | 64.8 | **33.1** | 46.1 | 64.8 | **33.1** | 46.7 |
| 3 | 31.0 | **23.1** | 25.9 | 31.5 | **24.8** | 28.0 | 28.2 | **14.2** | **14.9** | 28.2 | **14.2** | **15.4** |
| 4 | 44.3 | **31.2** | **32.8** | 45.2 | **32.7** | **35.4** | 40.9 | **20.7** | **21.9** | 40.9 | **20.7** | 23.0 |
| 5 | 28.2 | **22.3** | 27.0 | 29.0 | **23.7** | 27.5 | 33.4 | **16.9** | 19.9 | 33.4 | **16.9** | 21.0 |
| 6 | 2.1 | **1.7** | 3.4 | 2.4 | **2.1** | 3.5 | 1.8 | **1.0** | 1.3 | 1.8 | **1.0** | 1.5 |
| 7 | 13.1 | **9.6** | 11.0 | 13.7 | **11.1** | **11.7** | 21.9 | **11.3** | 16.4 | 21.9 | **11.3** | 18.2 |
| 8 | 17.2 | **10.9** | 12.6 | 17.6 | **12.1** | **12.7** | 28.1 | **14.3** | 21.0 | 28.1 | **14.3** | 21.4 |
| 9 | 205.4 | **129.3** | 144.5 | 208.5 | **136.3** | 155.8 | 135.7 | **68.4** | 77.8 | 135.7 | **68.4** | 79.9 |
| 10 | 91.8 | **67.6** | 77.9 | 94.6 | **75.0** | **79.2** | 81.6 | **41.7** | 52.9 | 81.6 | **41.7** | 53.7 |
| 11 | 25.3 | **15.2** | **16.0** | 26.0 | **16.5** | **17.7** | 34.1 | **17.3** | 25.2 | 34.1 | **17.3** | 25.8 |
| 12 | 15.2 | **10.7** | 12.7 | 15.9 | **12.2** | 14.9 | 19.8 | **10.2** | 12.5 | 19.8 | **10.2** | 13.5 |
| 13 | 284.2 | **165.2** | 184.7 | 288.6 | **177.9** | 201.4 | 178.0 | **89.8** | 114.5 | 178.0 | **89.8** | 119.7 |
| 14 | **4.2** | **4.4** | 5.7 | **4.7** | 8.4 | 7.9 | 7.1 | **3.7** | 4.6 | 7.1 | **3.7** | 5.4 |
| 15 | 10.4 | **8.4** | 10.1 | **11.2** | 13.6 | 12.7 | 18.1 | **9.3** | 12.4 | 18.1 | **9.3** | 14.3 |
| 16 | 294.5 | **136.2** | 157.9 | 299.3 | **180.3** | **180.2** | 241.0 | **121.0** | 169.9 | 241.0 | **121.0** | 175.1 |
| 17 | **46.1** | 57.5 | 52.0 | **50.6** | 235.1 | 73.1 | 55.6 | **28.9** | 38.1 | 55.6 | **28.9** | 41.4 |
| 18 | **5.6** | 74.8 | 16.6 | **6.3** | 164.3 | 24.3 | 6.6 | **3.5** | 4.4 | 6.6 | **3.5** | 4.9 |
| 19 | 114.5 | **68.7** | **73.8** | 116.4 | **74.0** | **79.2** | 89.9 | **45.3** | 54.0 | 89.9 | **45.3** | 56.1 |
| 20 | 156.7 | — | **118.6** | **160.3** | — | 179.4 | 153.0 | — | **103.6** | 153.0 | — | **105.5** |
| 21 | **7.5** | — | 22.9 | **8.0** | — | 33.5 | 8.4 | — | **6.7** | 8.4 | — | **7.3** |
| 22 | 536.2 | **254.5** | **269.3** | 546.9 | **271.7** | 307.2 | 312.0 | **157.0** | 187.5 | 312.0 | **157.0** | 188.7 |
| 23 | 67.2 | **46.6** | **49.0** | 70.0 | **55.1** | **59.5** | 52.9 | **26.7** | 33.1 | 52.9 | **26.7** | 34.5 |
| 24 | **62.9** | — | 109.8 | **71.6** | — | 170.4 | 77.6 | — | **54.3** | 77.6 | — | **65.6** |
| 25 | 97.6 | **65.4** | 79.8 | 103.1 | **90.2** | **92.2** | 192.0 | **97.7** | 141.7 | 192.0 | **97.7** | 142.3 |

Some general conclusions can be drawn from the results in this table. The LU-IR3 variants with single precision factorization generally achieve the lowest execution times, except for a few cases where iterative refinement underperforms for reasons we will discuss in section 6.4.4.1 or where convergence is not achieved. They also always achieve the lowest memory consumption when they converge, which comes at no surprise because most of the memory is consumed in the factorization step.

Since the LU-GMRES-IR5 variants with single precision factorization typically require more LU solves to achieve convergence than the LU-IR3 variants with single precision factorization, they usually have a higher execution time. Their memory consumption is also higher because in our implementation the factors are cast to double precision. These variants, however, generally provide a more robust and reliable solution with respect to the LU-IR3 ($u_f = \mathrm{S}$) ones. As a result, LU-GMRES-IR5 variants can solve problems where LU-IR3 do not achieve convergence. In such cases, for our matrix set, their execution time can be higher than that of variants that employ double precision factorization (DMUMPS or LU-IR3 with $u_f = \mathrm{D}$ and $u_r = \mathrm{Q}$); however their memory footprint usually remains smaller.

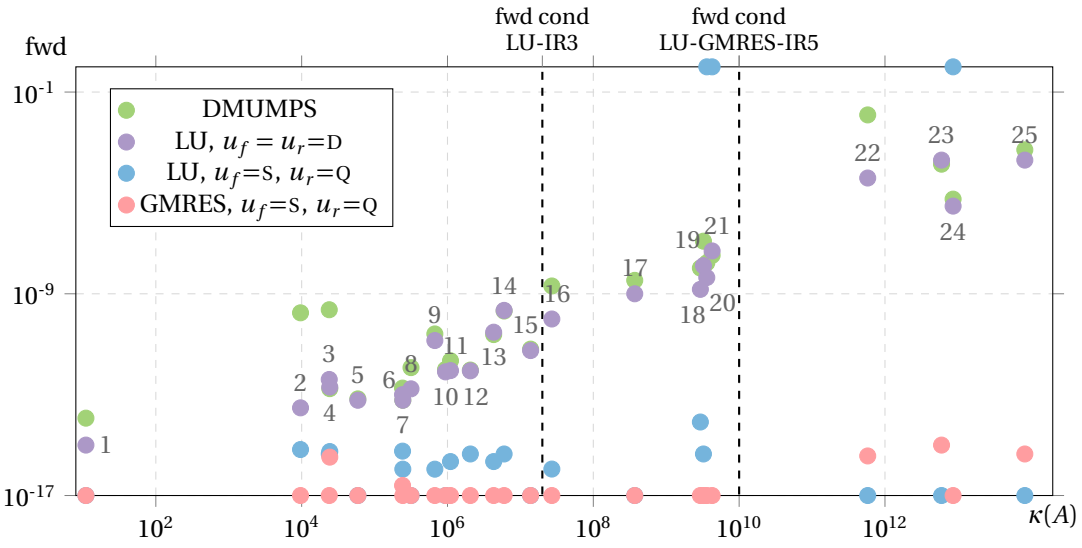



Figure 6.3: Forward error achieved by three IR variants for the matrices used in Table 6.3 (denoted by their ID) as a function of their condition number $\kappa(A)$. We fix $u = u_g = u_p =$ D. The vertical dashed lines show the forward convergence condition for LU-IR3 ($u_f =$ S, $u =$ D) and for LU-GMRES-IR5 ($u_f =$ S, $u = u_g = u_p =$ D).

Overall, Table 6.3 shows that the LU-GMRES-IR5 variants provide a good compromise between performance and robustness: unlike LU-IR3 ($u_f =$ S), they converge for all matrices in our set, while still achieving a significantly better performance than double precision based factorization variants.

It is also worth noting that, with respect to variants with $u_r =$ D, variants with $u_r =$ Q can achieve a forward error of order $10^{-16}$ with only a small additional overhead in both time (because the residual is computed in quadruple rather than double precision and a few more iterations are required) and memory consumption (because the matrix is stored in quadruple precision). As a result, these variants can produce a more accurate solution than a standard double precision direct solver (DMUMPS) with a smaller memory consumption and, in most cases, faster. We illustrate the accuracy improvement in Figure 6.3, which reports the forward error achieved by variants DMUMPS, LU-IR3 with $u_f = u = u_r =$ D (stopped when the forward error stagnates), and LU-IR3 and LU-GMRES-IR5 with $u_f =$ S and $u_r =$ Q.

In order to provide more insight into the behavior of each variant, we next carry out a detailed analysis of time and memory consumption in sections 6.4.4.1 and 6.4.4.2, respectively.

*6.4.4.1 Detailed execution time analysis.* The potential gain in execution time of mixed precision iterative refinement comes from the fact that the most time consuming operation, the LU factorization, is carried in low precision arithmetic and high precision is only used in refinement steps which involve low complexity operations as explained in section 4.2.4. For this gain to be effective, the cost of the refinement iterations must not exceed the time re-

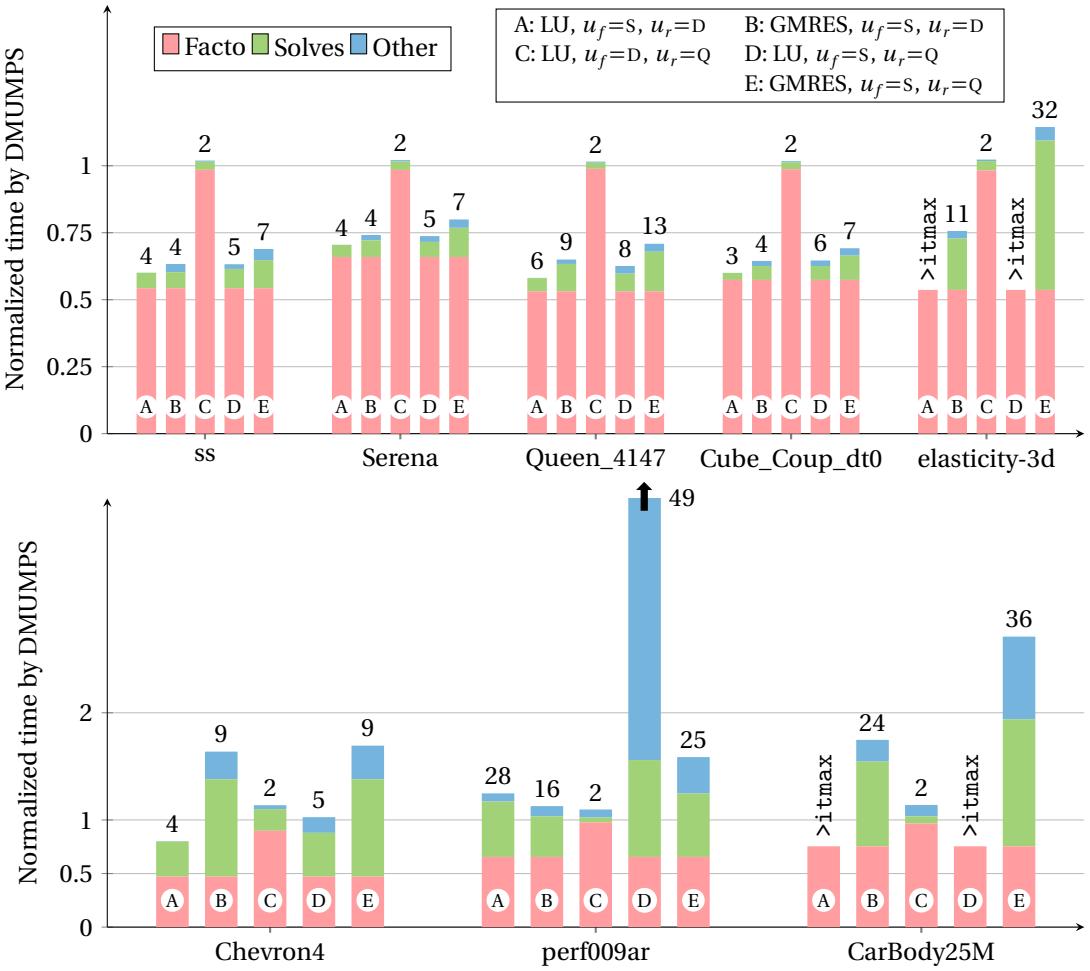



Figure 6.4: Execution time for different LU-IR3 and LU-GMRES-IR5 variants normalized by that of DMUMPS, for a subset of our test matrices (denoted by their ID on the x-axis). Each bar shows the time breakdown into LU factorization, LU solves, and other computations. We print on top of each bar the total number of calls to LU solves. We fix $u = u_g = u_p =$ D. Variants with $u_r =$ D provide a forward error equivalent to the one obtained with DMUMPS (A and B), while variants with $u_r =$ Q provide a forward error of order $10^{-16}$ (C, D, and E).

duction resulting from running the factorization in low precision. This is very often the case. First of all, on current processors (such as the model used for our experiments) computations in single precision can be up to twice as fast as those in double precision. Additionally, operations performed in the refinement steps have a lower asymptotic complexity compared with the factorization. Nonetheless, in practice, the overall time reduction can vary significantly depending on a number of parameters. First of all, the ratio between the complexity of the factorization and that of the solution phase is less favorable on 2D problems than on 3D problems (see Table 2.2). Second, the single precision factorization may be less than twice as fast as the double precision one; this may happen, for example, on small problems where the overhead of symbolic operations in the factorization (data indexing, handling of data structures, etc.) is relatively high or, in a parallel setting, because the single

precision factorization is less scalable than the double precision one due to the relatively lower weight of floating-point operations with respect to that of symbolic ones. It must also be noted that although the factorization essentially relies on efficient BLAS-3 operations, the operations done in the iterative refinement, in particular the LU solves, rely on memory-bound BLAS-2 operations and are thus less efficient. Finally, in the case of badly conditioned problems, iterative refinement may require numerous iterations to achieve convergence.

Figure 6.4 shows the execution time of variants encountered in Table 6.3 normalized with respect to that of DMUMPS for a selected subset of matrices from our test set; each bar also shows the time breakdown into LU factorization, LU solves and all the rest which includes computing the residual and, for the GMRES-based variants, casting the factors, computing the Krylov basis, orthonormalizing it, etc. The values on top of each bar are the number of LU solve operations; note that for GMRES-based variants, multiple LU solve operations are done in each refinement iteration.

In the first row of this figure we report problems that behave well, in the sense that all the parameters mentioned above align in the direction that leads to a good overall time reduction. For these problems the single precision factorization is roughly twice as fast as the double precision one, the complexity of the solve is much lower than that of the factorization (three orders of magnitude in all cases, as reported in Table 6.1), and relatively few iterations are needed to achieve convergence. For all these problems, the complexity of the matrix–vector product is more than two orders of magnitude lower than that of the solve (see columns "NNZ" and "Slv." of Table 6.1). As a result, the computation of the residual only accounts for a small portion of the total execution time—even for variants with $u_r = \mathrm{Q}$, for which it is carried out in slow quadruple precision arithmetic (which is not supported by our hardware). This is a very desirable property since these variants greatly improve the forward error with only a modest overhead. The figure clearly shows, however, that despite their relatively low complexity, the operations in iterative refinement are relatively slow and, therefore, the gain is considerably reduced when many solves are necessary. This issue is exacerbated in the case of LU-GMRES-IR5 variants, because the solves are carried out in double instead of single precision as for LU-IR3 variants ($u_f = \mathrm{S}$).

In the second row of Figure 6.4 we report some cases where mixed precision iterative refinement does not reduce execution time. Chevron4 is a relatively small 2D problem where the cost of the solve and the matrix–vector product relative to that of the factorization is high; as a result, even for a moderate number of refinement iterations, variant DMUMPS achieves the best execution time and all other variants are much slower. perf009ar is one where the single precision factorization is only 1.6 times faster than the double precision one and, additionally, it produces little fill-in (as shown by the small ratio Slv./NNZ in Table 6.1) and so the relative cost of computing the residual in quadruple precision is high. Finally, CarBody25M is badly conditioned and variants based on single precision factorization either do not converge or require so many iterations that the execution time is higher than that of DMUMPS. It is however worth noting that on these particular matrices variants based on single precision factorization may be slower than DMUMPS but at a significantly reduced memory cost (as shown in Table 6.3).

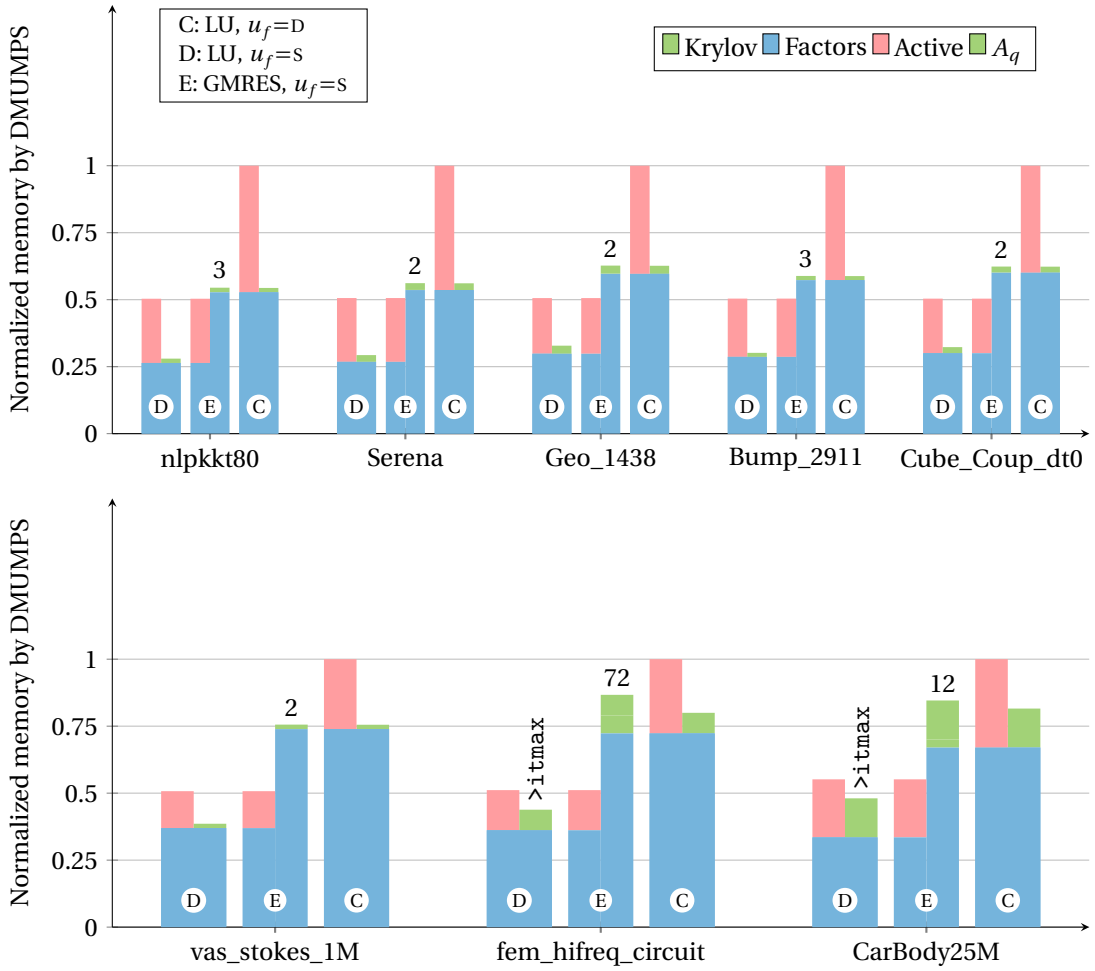



Figure 6.5: Memory consumption for different LU-IR3 and LU-GMRES-IR5 variants normalized by that of DMUMPS, for a subset of our test matrices (denoted by their ID on the x-axis). The bars show the memory breakdown in factors memory, active memory overhead, the storage for the Krylov basis (LU-GMRES-IR5 variant only), and the storage for the matrix in quadruple precision. Each variant bar is split into two subbars corresponding to the peak consumption during the factorization and solve phases, respectively (and thus the overall required memory by the variant is the maximum of the two peaks). We print on top of the LU-GMRES-IR5 variant the maximum size reached by the Krylov basis over the refinement iterations. We fix $u = u_g = u_p =$ D and $u_r =$ Q. All the variants (C, D, and E) provide a forward error of order $10^{-16}$.

*6.4.4.2 Detailed memory consumption analysis.* One distinctive feature of the multifrontal method in comparison with left or right-looking ones is in the way it uses memory as explained in section 2.2.3.4. In addition to the memory needed to store the factors which grows monotonically throughout the factorization, the multifrontal method also needs a temporary workspace which we refer to as active memory to store the contribution blocks. As a result, the peak memory consumption achieved in the course of the factorization is generally higher than the memory needed to store the factors. It must also be noted that

parallelism does not have any effect on the memory needed to store the factors but generally increases the size of the active memory: this is because more node of the tree are processed at the same time (see Figure 2.6) and, so, more contribution blocks have to be stored at the same time. For a thorough discussion of the memory consumption in the multifrontal method we refer the reader to the paper by Agullo et al. [5].

We assume that at the end of the factorization all the active memory is freed and only the factors are left in memory. It is only at this moment that the original problem matrix is cast to quadruple precision for computing the residual at each refinement iteration. Therefore, the active memory overhead and the memory required to store a quadruple precision version of the matrix do not accumulate. In our implementation, the LU-GMRES-IR5 variants with $u_p = u^2 = \text{D}$ also require the factors to be cast to double precision which we do upon completion of the factorization, when the active memory is freed as described in section 6.4.3.2. We also report the size of the Krylov basis in the GMRES solver: although in most of our experiments this is completely negligible, there might be cases (we will show one) where the number of GMRES iterations is sufficiently high to make the memory consumed by the Krylov basis relevant. Finally, we do not count the memory consumption of the solution, residual and correction vectors.

All these assumptions lead us to Figure 6.5 where we present the normalized memory consumption of certain LU-IR3 and LU-GMRES-IR5 variants relative to that of variant DMUMPS for a selected subset of problems. We do not include variants using $u_r = \text{D}$ because they behave very similarly to variants with $u_r = \text{Q}$. For each problem and variant the bar is split in two parts showing that the memory consumption peak can happen during and after the factorization, respectively.

In the first row we report problems that behave well, which corresponds to the most common case as shown in Table 6.3. It shows, as expected, that LU-IR3 with single precision factorization consumes half as much memory as DMUMPS because the memory needed to store the problem matrix in quadruple precision does not exceed the active memory overhead. Thus, the highest memory consumption corresponds to the single precision factorization peak. GMRES-based variant ($u_p = \text{D}$) casts the factors to double precision which exceeds the peak of the single precision factorization. Nonetheless, even if on top of this we have to add the memory needed to store the quadruple precision matrix, the overall consumption is lower than the double precision factorization peak by a factor which can be almost up to 50% on this set, making the memory consumption of the LU-GMRES-IR5 variant almost identical to that of the LU-IR3 one in a few cases (such as matrices nlpkkt80 and Serena). As for the LU-IR3 variant with $u_f = \text{D}$, it clearly does not bring any improvement with respect to DMUMPS but no loss either because the memory for storing the matrix in quadruple precision is much lower than the active memory overhead.

In the second row of Figure 6.5 we report problems where the memory reduction is not as good, for four different reasons, the last two of which are exclusive to LU-GMRES-IR5.

1. In the case of CarBody25M, the single precision factorization consumes more than half the memory of the double precision one (about 55%). This is because the relative weight of the symbolic data structures, which is counted as part of the active memory overhead and does not depend on the factorization precision, is high for this matrix.

2. In the case of fem_hifreq_circuit and CarBody25M the factorization generates little fill-in which makes the relative weight of the quadruple precision matrix copy significant compared with the size of the factors. Here this storage remains less than the active memory overhead and so the overall memory consumption of LU-IR3 is not impacted; however, it does impact LU-GMRES-IR5, leading to less memory savings.

3. In the case of vas_stokes_1M and fem_hifreq_circuit (and to a lesser extent 24) the active memory overhead represents a smaller fraction of the total memory, further reducing the memory savings for LU-GMRES-IR5.

4. Finally, fem_hifreq_circuit (and to a lesser extent CarBody25M) is one of the few matrices having a non-negligible Krylov basis memory footprint, showing that an increase in the number iterations for the GMRES to converge diminishes here the potential memory gain.

### 6.4.5 Performance of LU-IR and LU-GMRES-IR using approximate factorizations

In this set of experiments we are interested in studying the performance of LU-IR3 and LU-GMRES-IR5 combined with BLR, static pivoting, and BLR with static pivoting. For each experiment, we use a selected set of matrices from Table 6.1 which are representative of different types of behavior that can be encountered in practice.

These approximation techniques have two conflicting effects on the performance: if, on the one hand, they reduce the time and memory of the factorization, on the other hand, they increase the number of refinement iterations.

*6.4.5.1 BLR factorization.* In Table 6.4 we present the execution time and memory consumption of four iterative refinement variants using low-rank BLR factorization for different values of the compression threshold $\tau_b$, and in Table 6.5 we provide the associated factorizations, LU solves, and matrix–vector products execution times. All variants provide a forward error on the solution equivalent to the one of DMUMPS. If $\tau_b$ = "full-rank", the factorization is run without BLR, this is a standard factorization as in section 6.4.4. It should be noted that in this case the double precision factorization LU-IR3 variant is equivalent to DMUMPS and we will refer to it as DMUMPS in the text. We denote by "—" the cases where convergence is not reached and, for each matrix, we highlight in bold the execution time and memory consumption that do not exceed by more than 10% the best execution time or memory consumption. We choose to work with the default BLR settings of MUMPS, in which case the data in the active memory is not compressed with low-rank approximations. We consider the compression of the active memory in the next section 6.4.5.2.

The experimental results of Table 6.4 are in good agreement with the theoretical convergence conditions of Theorems 6.1 and 6.2 developed in section 6.3. We can clearly see how the robustness of the presented variants is related to both the condition number $\kappa(A)$ of the matrix (specified for each matrix in Table 6.1) and the BLR threshold $\tau_b$. Convergence is not achieved for excessively large values of the BLR threshold; the largest $\tau_b$ value for

which convergence is achieved depends on the matrix condition number and, in general, it is smaller for badly conditioned problems. In the case of LU-GMRES-IR5 variants, which are more robust, the BLR threshold can be pushed to larger values without breaking convergence. Note that there is no situation where a variant does not converge when in theory it should (that is, when the convergence condition is met). However, as the theoretical convergence conditions in Theorems 6.1 and 6.2 can be pessimistic, there are several cases where convergence is achieved even though the theoretical convergence condition is not met.

The use of BLR with a good choice of compression threshold $\tau_b$ generally results in substantial reductions of the LU-IR3 and LU-GMRES-IR5 execution times. As the BLR threshold increases, the operational complexity and, consequently, the execution time of the factorization and solve operations decreases; conversely, the number of iterations increases up to the point where convergence may not be achieved anymore. The optimal BLR threshold value which delivers the lowest execution time obviously depends on the problem. It must be noted that even though the LU-GMRES-IR5 variants achieve convergence for larger $\tau_b$ values, this leads to an excessive number of iterations whose cost exceeds the improvement provided by BLR; as a result, these variants are slower than LU-IR3 ones ($u_f = \text{S}$ but also $u_f = \text{D}$) in all cases. Consequently, the single precision factorization LU-IR3 variant generally achieves the best execution time on this set of problems, similarly to what was observed in section 6.4.4, with a few exceptions. On perf009ar the double precision factorization LU-IR3 variant is the best due to the fact that similarly to the full-rank case (see section 6.4.4.1) the BLR factorization is less than twice as fast when single precision is used instead of double for the same $\tau_b$ value; additionally, a substantial number of iterations is needed to achieve convergence. It is worth mentioning that on this matrix the LU-GMRES-IR5 variant with $u_g = u_p = \text{D}$ is faster than the single precision factorization LU-IR3 variant (36.9s versus 40.3s) and consumes less memory than the double precision factorization LU-IR3 variant (20.0GB versus 37.1GB). On CarBody25M, DMUMPS is the fastest variant as in the full-rank case; this is due to the fact that, on this problem, BLR does not achieve a good reduction of the operational complexity and, therefore, of the execution time.

As for the storage, the use of BLR leads to a different outcome with respect to the case where a full-rank factorization is used (see section 6.4.4) where the single precision factorization LU-IR3 variant is the best. This is due to the combination of two factors. First, when BLR is used, the relative weight of the active memory is higher because it corresponds to data which is not compressed due to the choice of parameters we have made; consequently, the memory consumption peak is often reached during the factorization rather than during the refinement steps. Second, the memory consumption of the factorization decreases monotonically when the BLR threshold is increased. As a result of these two effects, the LU-GMRES-IR5 variants achieve the lowest memory consumption on this set of problems, because they can preserve convergence for larger values of $\tau_b$ than the LU-IR3 variants can. For example, on tminlet3M the LU-GMRES-IR5 variant with $u_g = u_p = \text{D}$ consumes almost 15% less memory than the LU-IR3 one with $u_f = \text{S}$ (70.9GB versus 82.4GB), on thmgas the LU-GMRES-IR5 variant with $u_g = u_p = \text{D}$ consumes almost 30% less memory than variant LU-IR3 with $u_f = \text{S}$ (43.7GB versus 61.4GB), and on matrix 24 the LU-GMRES-IR5

Table 6.4: Execution time, memory consumption and number of LU solve calls of IR variants for the industrial matrices listed in bold in Table 6.1 and depending on the compression threshold $\tau_b$. We fix $u_r = u = \text{D}$.

| | Solver | LU | LU | GMRES | GMRES | LU | LU | GMRES | GMRES | LU | LU | GMRES | GMRES |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | $u_f$ | D | S | S | S | D | S | S | S | D | S | S | S |
| | $u_p$=$u_g$ | — | — | D | S | — | — | D | S | — | — | D | S |
| ID | $\tau_b$ | Time (s) | | | | Memory (GB) | | | | Nb LU solves | | | |
| ElectroPhys10M | full-rank | 265.2 | 154.0 | 166.5 | 163.3 | 272.0 | 138.0 | 171.3 | 138.0 | 1 | 3 | 5 | 7 |
| | 1E-10 | 101.0 | **70.5** | 73.7 | 72.7 | 158.0 | 84.5 | 84.6 | 84.6 | 2 | 3 | 5 | 7 |
| | 1E-08 | 93.1 | **70.0** | 72.8 | 72.1 | 157.0 | 80.6 | 82.2 | 82.2 | 2 | 3 | 5 | 7 |
| | 1E-06 | 91.1 | **64.9** | **68.2** | **68.5** | 149.0 | **77.8** | 79.6 | 79.6 | 3 | 3 | 6 | 10 |
| | 1E-04 | 88.3 | **66.3** | **69.3** | **70.8** | 143.0 | **73.6** | **77.0** | **77.0** | 4 | 4 | 7 | 13 |
| | 1E-02 | 89.8 | 71.4 | 75.0 | 128.1 | 147.0 | **73.6** | **73.6** | **73.6** | 9 | 9 | 11 | 125 |
| | 1E-01 | 97.8 | 73.6 | 81.0 | 119.5 | 147.0 | **71.8** | **73.4** | **73.4** | 19 | 18 | 24 | 115 |
| DrivAer6M | full-rank | 91.8 | 67.6 | 77.9 | 77.4 | 81.6 | 41.7 | 52.9 | 41.7 | 1 | 3 | 5 | 7 |
| | 1E-10 | 55.6 | **43.7** | 46.5 | 46.9 | 54.5 | 28.2 | 28.2 | 28.2 | 2 | 6 | 7 | 10 |
| | 1E-08 | 54.3 | **42.2** | **44.8** | **46.1** | 51.9 | 26.9 | 26.9 | 26.9 | 4 | 6 | 7 | 11 |
| | 1E-06 | 62.6 | 47.8 | **44.7** | **45.8** | 49.3 | 25.5 | 25.5 | 25.5 | 15 | 15 | 9 | 13 |
| | 1E-04 | — | — | 73.2 | 106.2 | — | — | **24.0** | **24.0** | — | — | 45 | 95 |
| | 1E-02 | — | — | 248.4 | 404.3 | — | — | **22.7** | **22.7** | — | — | 256 | 502 |
| | 1E-01 | — | — | 471.3 | 1490.4 | — | — | **22.4** | 41.0 | — | — | 528 | 1997 |
| tminlet3M | full-rank | 294.5 | 136.2 | 157.9 | 176.3 | 241.0 | 121.0 | 169.9 | 121.0 | 1 | 7 | 15 | 56 |
| | 1E-10 | 232.9 | 158.8 | 174.0 | 181.2 | 188.0 | 118.0 | 169.9 | 118.0 | 2 | 7 | 16 | 55 |
| | 1E-08 | 204.9 | 149.7 | 165.3 | 182.7 | 171.0 | 114.0 | 161.9 | 114.0 | 3 | 7 | 17 | 79 |
| | 1E-06 | 179.0 | **88.3** | 98.8 | 105.5 | 154.0 | 82.4 | 93.8 | 82.8 | 5 | 7 | 16 | 54 |
| | 1E-04 | — | — | 105.6 | 116.3 | — | — | **70.9** | **70.9** | — | — | 69 | 181 |
| perf009ar | full-rank | 46.1 | 57.5 | 52.0 | 110.0 | 55.6 | 28.9 | 38.1 | 28.9 | 1 | 28 | 16 | 92 |
| | 1E-10 | **32.9** | 40.3 | 36.9 | 83.1 | 38.7 | 20.5 | 25.6 | 20.5 | 2 | 22 | 15 | 94 |
| | 1E-08 | **33.6** | 41.5 | 37.3 | 88.0 | 37.1 | **19.7** | 22.8 | **19.7** | 4 | 26 | 16 | 107 |
| | 1E-06 | — | — | 40.9 | 187.8 | — | — | **20.0** | **18.6** | — | — | 25 | 280 |
| | 1E-04 | — | — | 658.3 | — | — | — | 36.6 | — | — | — | 949 | — |
| | 1E-02 | — | — | 2224.1 | — | — | — | 98.1 | — | — | — | 3338 | — |
| elasticity-3d | full-rank | 156.7 | — | 118.6 | — | 153.0 | — | 103.6 | — | 1 | — | 11 | — |
| | 1E-10 | 110.2 | — | 82.3 | — | 95.5 | — | 72.1 | — | 2 | — | 10 | — |
| | 1E-08 | 96.3 | — | 68.6 | — | 91.6 | — | 56.8 | — | 5 | — | 10 | — |
| | 1E-06 | — | — | **57.9** | — | — | — | 44.1 | — | — | — | 13 | — |
| | 1E-04 | — | — | 125.3 | — | — | — | **39.0** | — | — | — | 121 | — |
| lfm_aug5M | full-rank | 536.2 | 254.5 | 269.3 | 353.6 | 312.0 | 157.0 | 187.5 | 157.0 | 1 | 4 | 5 | 46 |
| | 1E-10 | 313.3 | 199.8 | 210.0 | 230.9 | 240.0 | 141.0 | 147.6 | 144.0 | 2 | 4 | 7 | 37 |
| | 1E-08 | 260.2 | 119.2 | 130.1 | 162.3 | 218.0 | 112.0 | 116.0 | 116.0 | 3 | 4 | 9 | 60 |
| | 1E-06 | 223.2 | **100.4** | 110.1 | 131.3 | 199.0 | 107.0 | 107.0 | 107.0 | 4 | 4 | 9 | 47 |
| | 1E-04 | 212.3 | **95.8** | 105.4 | 124.7 | 200.0 | **101.0** | **103.0** | **103.0** | 22 | 20 | 19 | 65 |
| | 1E-02 | — | — | 482.6 | 1111.0 | — | — | **96.8** | **96.8** | — | — | 367 | 1763 |
| CarBody25M | full-rank | **62.9** | — | 109.8 | — | 77.6 | — | 54.3 | — | 1 | — | 24 | — |
| | 1E-10 | **63.3** | — | 90.8 | — | 65.5 | — | **44.0** | — | 3 | — | 23 | — |
| | 1E-08 | **68.9** | — | 91.3 | — | 64.8 | — | **41.8** | — | 6 | — | 23 | — |
| | 1E-06 | — | — | 299.4 | — | — | — | 55.8 | — | — | — | 140 | — |
| thmgas | full-rank | 97.6 | **65.4** | 79.8 | 79.6 | 192.0 | 97.7 | 141.7 | 97.7 | 1 | 4 | 7 | 10 |
| | 1E-10 | 88.9 | **63.7** | 75.8 | 69.5 | 137.0 | 70.9 | 110.5 | 71.0 | 2 | 4 | 7 | 7 |
| | 1E-08 | 81.3 | **59.5** | 66.1 | 66.7 | 131.0 | 67.5 | 92.1 | 67.6 | 3 | 4 | 7 | 7 |
| | 1E-06 | 85.1 | **61.4** | 65.6 | 70.8 | 118.0 | 61.4 | 70.4 | 61.5 | 8 | 8 | 9 | 13 |
| | 1E-04 | — | — | 147.5 | 131.4 | — | — | 53.7 | 53.7 | — | — | 53 | 48 |
| | 1E-02 | — | — | 1043.9 | 2380.8 | — | — | **45.5** | **45.5** | — | — | 523 | 1259 |
| | 1E-01 | — | — | 3340.5 | 3155.2 | — | — | 48.9 | **43.7** | — | — | 1399 | 1649 |

Table 6.5: Execution time of the factorizations, LU solves, and matrix–vector products of IR variants associated with the runs in Table 6.4.

| | Solver | LU | LU | GMRES | GMRES | LU | LU | GMRES | GMRES | LU | LU | GMRES | GMRES |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | $u_f$ | D | S | S | S | D | S | S | S | D | S | S | S |
| | $u_p{=}u_g$ | — | — | D | S | — | — | D | S | — | — | D | S |
| ID | $\tau_b$ | Factorization (s) | | | | Solve (s) | | | | SpMV (s) | | | |
| ElectroPhys10M | full-rank | 262.0 | 147.0 | 147.0 | 147.0 | 3.22 | 2.31 | 3.22 | 2.31 | 0.04 | 0.04 | 0.04 | 0.04 |
| | 1E-10 | 99.4 | 68.8 | 68.8 | 68.8 | 0.77 | 0.55 | 0.77 | 0.55 | 0.04 | 0.04 | 0.04 | 0.04 |
| | 1E-08 | 91.6 | 68.3 | 68.3 | 68.3 | 0.72 | 0.53 | 0.72 | 0.53 | 0.04 | 0.04 | 0.04 | 0.04 |
| | 1E-06 | 88.9 | 63.3 | 63.3 | 63.3 | 0.69 | 0.51 | 0.69 | 0.51 | 0.04 | 0.04 | 0.04 | 0.04 |
| | 1E-04 | 85.7 | 64.2 | 64.2 | 64.2 | 0.63 | 0.50 | 0.63 | 0.50 | 0.04 | 0.04 | 0.04 | 0.04 |
| | 1E-02 | 83.3 | 66.7 | 66.7 | 66.7 | 0.69 | 0.49 | 0.69 | 0.49 | 0.04 | 0.04 | 0.04 | 0.04 |
| | 1E-01 | 84.3 | 64.2 | 64.2 | 64.2 | 0.67 | 0.48 | 0.67 | 0.48 | 0.04 | 0.04 | 0.04 | 0.04 |
| DrivAer6M | full-rank | 89.6 | 57.5 | 57.5 | 57.5 | 2.21 | 1.65 | 2.21 | 1.65 | 0.04 | 0.04 | 0.04 | 0.04 |
| | 1E-10 | 53.6 | 38.5 | 38.5 | 38.5 | 0.99 | 0.83 | 0.99 | 0.83 | 0.04 | 0.04 | 0.04 | 0.04 |
| | 1E-08 | 50.3 | 37.1 | 37.1 | 37.1 | 0.96 | 0.81 | 0.96 | 0.81 | 0.04 | 0.04 | 0.04 | 0.04 |
| | 1E-06 | 48.2 | 35.5 | 35.5 | 35.5 | 0.92 | 0.78 | 0.92 | 0.78 | 0.04 | 0.04 | 0.04 | 0.04 |
| | 1E-04 | — | — | 33.8 | 33.8 | — | — | 0.86 | 0.76 | — | — | 0.04 | 0.04 |
| | 1E-02 | — | — | 32.7 | 32.7 | — | — | 0.84 | 0.74 | — | — | 0.04 | 0.04 |
| | 1E-01 | — | — | 32.4 | 32.4 | — | — | 0.83 | 0.73 | — | — | 0.04 | 0.04 |
| tminlet3M | full-rank | 293.0 | 130.0 | 130.0 | 130.0 | 1.49 | 0.82 | 1.49 | 0.82 | 0.08 | 0.08 | 0.08 | 0.08 |
| | 1E-10 | 231.0 | 155.0 | 155.0 | 155.0 | 0.93 | 0.47 | 0.93 | 0.47 | 0.08 | 0.08 | 0.08 | 0.08 |
| | 1E-08 | 202.0 | 146.0 | 146.0 | 146.0 | 0.91 | 0.46 | 0.91 | 0.46 | 0.08 | 0.08 | 0.08 | 0.08 |
| | 1E-06 | 175.0 | 85.2 | 85.2 | 85.2 | 0.73 | 0.37 | 0.73 | 0.37 | 0.08 | 0.08 | 0.08 | 0.08 |
| | 1E-04 | — | — | 63.4 | 63.4 | — | — | 0.59 | 0.29 | — | — | 0.08 | 0.08 |
| perf009ar | full-rank | 45.0 | 30.2 | 30.2 | 30.2 | 1.09 | 0.85 | 1.09 | 0.85 | 0.13 | 0.13 | 0.13 | 0.13 |
| | 1E-10 | 31.3 | 24.1 | 24.1 | 24.1 | 0.75 | 0.61 | 0.75 | 0.61 | 0.13 | 0.13 | 0.13 | 0.13 |
| | 1E-08 | 30.0 | 22.9 | 22.9 | 22.9 | 0.81 | 0.59 | 0.81 | 0.59 | 0.13 | 0.13 | 0.13 | 0.13 |
| | 1E-06 | — | — | 22.1 | 22.1 | — | — | 0.70 | 0.58 | — | — | 0.13 | 0.13 |
| | 1E-04 | — | — | 21.1 | — | — | — | 0.67 | — | — | — | 0.13 | — |
| | 1E-02 | — | — | 19.8 | — | — | — | 0.66 | — | — | — | 0.13 | — |
| elasticity-3d | full-rank | 154.0 | — | 84.1 | — | 2.73 | — | 2.73 | — | 0.03 | — | 0.03 | — |
| | 1E-10 | 108.0 | — | 69.0 | — | 1.07 | — | 1.07 | — | 0.03 | — | 0.03 | — |
| | 1E-08 | 91.5 | — | 57.2 | — | 0.94 | — | 0.94 | — | 0.03 | — | 0.03 | — |
| | 1E-06 | — | — | 45.7 | — | — | — | 0.82 | — | — | — | 0.03 | — |
| | 1E-04 | — | — | 39.4 | — | — | — | 0.70 | — | — | — | 0.03 | — |
| lfm_aug5M | full-rank | 532.0 | 245.0 | 245.0 | 245.0 | 4.22 | 2.36 | 4.22 | 2.36 | 0.03 | 0.03 | 0.03 | 0.03 |
| | 1E-10 | 310.0 | 196.0 | 196.0 | 196.0 | 1.66 | 0.94 | 1.66 | 0.94 | 0.03 | 0.03 | 0.03 | 0.03 |
| | 1E-08 | 256.0 | 116.0 | 116.0 | 116.0 | 1.39 | 0.77 | 1.39 | 0.77 | 0.03 | 0.03 | 0.03 | 0.03 |
| | 1E-06 | 218.0 | 97.4 | 97.4 | 97.4 | 1.28 | 0.72 | 1.28 | 0.72 | 0.03 | 0.03 | 0.03 | 0.03 |
| | 1E-04 | 186.0 | 82.3 | 82.3 | 82.3 | 1.17 | 0.65 | 1.17 | 0.65 | 0.03 | 0.03 | 0.03 | 0.03 |
| | 1E-02 | — | — | 70.7 | 70.7 | — | — | 1.12 | 0.59 | — | — | 0.03 | 0.03 |
| CarBody25M | full-rank | 60.8 | — | 47.4 | — | 2.07 | — | 2.07 | — | 0.31 | — | 0.31 | — |
| | 1E-10 | 57.4 | — | 46.6 | — | 1.77 | — | 1.77 | — | 0.31 | — | 0.31 | — |
| | 1E-08 | 56.6 | — | 45.4 | — | 1.79 | — | 1.79 | — | 0.31 | — | 0.31 | — |
| | 1E-06 | — | — | 45.2 | — | — | — | 1.78 | — | — | — | 0.31 | — |
| thmgas | full-rank | 95.1 | 55.6 | 55.6 | 55.6 | 2.51 | 2.37 | 2.51 | 2.37 | 0.10 | 0.10 | 0.10 | 0.10 |
| | 1E-10 | 83.6 | 55.4 | 55.4 | 55.4 | 2.59 | 1.99 | 2.59 | 1.99 | 0.10 | 0.10 | 0.10 | 0.10 |
| | 1E-08 | 74.8 | 49.5 | 49.5 | 49.5 | 2.10 | 2.43 | 2.10 | 2.43 | 0.10 | 0.10 | 0.10 | 0.10 |
| | 1E-06 | 67.6 | 45.1 | 45.1 | 45.1 | 2.10 | 1.95 | 2.10 | 1.95 | 0.10 | 0.10 | 0.10 | 0.10 |
| | 1E-04 | — | — | 41.8 | 41.8 | — | — | 1.97 | 1.86 | — | — | 0.10 | 0.10 |
| | 1E-02 | — | — | 38.7 | 38.7 | — | — | 1.92 | 1.86 | — | — | 0.10 | 0.10 |
| | 1E-01 | — | — | 37.9 | 37.9 | — | — | 2.36 | 1.89 | — | — | 0.10 | 0.10 |

variant with $u_g = u_p = \text{D}$ consumes more than 35% less memory than variant LU-IR3 with $u_f = \text{D}$ (41.8GB versus 64.8GB). It is worth pointing out that the value of $\tau_b$ for which LU-GMRES-IR5 achieves the lowest possible memory consumption is not always the largest value for which convergence is still possible. This is because for a large number of iterations the memory needed to store the Krylov basis may exceed the savings obtained with BLR. This problem can be overcome or mitigated by choosing an appropriate value for the $\tau_g$ threshold or, similarly, using a restarted GMRES method; we leave this analysis for future work.

We finally compare the two LU-GMRES-IR5 variants $u_g = u_p = \text{D}$ and $u_g = u_p = \text{S}$. When $u_g = u_p = \text{S}$, LU-GMRES-IR5 avoids the cast of the LU factors from single to double precision, and thus reduces memory consumption compared with $u_g = u_p = \text{D}$. However, as explained above, the relative weight of the factors with respect to the active memory is smaller as $\tau_b$ increases, and so the reduction achieved by LU-GMRES-IR5 with $u_g = u_p = \text{S}$ grows smaller until the point where both variants achieve a similar memory consumption. On our matrix set, for the values of $\tau_b$ where the LU-IR3 with $u_f = \text{S}$ does not converge, LU-GMRES-IR5 with $u_g = u_p = \text{S}$ does not achieve significant memory reductions compared with LU-GMRES-IR5 with $u_g = u_p = \text{D}$ (at best 7% on perf009ar, 18.6GB versus 20.0GB).

*6.4.5.2 BLR factorization with compressed active memory.* When using BLR, as explained in section 2.2.4.1, we have the flexibility to choose whether the contribution blocks are compressed or not. With the default settings, the MUMPS BLR factorization only compresses the factors; the active memory, composed of the contribution blocks, remains full-rank. If we choose to compress the contribution blocks as well, we reduce the active memory overhead during factorization and, then, we further reduce the peak memory consumption. However, compressing the contributions blocks does not reduce the operational complexity and even adds flop overhead. In addition, compressing the contribution blocks will also exchange efficient BLAS-3 kernels (i.e., dense matrix–matrix product) for slower operations (i.e., compression kernels). For these reasons, we can expect a higher execution time for sequential factorization. Consequently, compressing the active memory can reduce the memory peak, but at the cost of increasing the execution time. It is, therefore, particularly relevant for applications where memory is the constraint. Note that the increased execution time can be mitigated in highly parallelized settings where the application requires many communications that can be made faster by compressing the active memory.

We present, in Table 6.6, the corresponding execution time and memory consumption as in Table 6.4 but with the active memory compression activated.

In terms of the numerics, we expect that compressing the active memory will not change much the numerical behavior of the methods because with the contribution blocks compressed or not, the perturbation introduced will still be of order the compression threshold $\tau_b$, so the convergence conditions of Theorems 6.1 and 6.2 should not be affected. It is actually what we observed from our experiments. Indeed, Table 6.6 displays nearly identical convergence behavior as in the previous Table 6.4. That is, if a given variant was converging, it still converges, conversely if it was not converging it still does not; in addition, the number of LU solve calls is also relatively similar.

Table 6.6: Execution time, memory consumption and number of LU solve calls of IR variants as in Table 6.4 but with compressed active memory.

| | Solver | LU | LU | GMRES | GMRES | LU | LU | GMRES | GMRES | LU | LU | GMRES | GMRES |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | $u_f$ | D | S | S | S | D | S | S | S | D | S | S | S |
| | $u_p$=$u_g$ | — | — | D | S | — | — | D | S | — | — | D | S |
| ID | $\tau_b$ | Time (s) | | | | Memory (GB) | | | | Nb LU solves | | | |
| ElectroPhys10M | full-rank | 265.2 | 154.0 | 166.5 | 163.3 | 272.0 | 138.0 | 171.3 | 138.0 | 1 | 3 | 5 | 7 |
| | 1E-10 | 87.7 | 64.4 | 66.5 | 66.6 | 115.0 | 59.7 | 61.6 | 61.6 | 2 | 3 | 5 | 7 |
| | 1E-08 | 86.0 | **62.7** | 64.6 | 64.8 | 109.0 | 56.4 | 56.4 | 56.4 | 2 | 3 | 5 | 7 |
| | 1E-06 | 82.0 | **60.5** | **61.6** | 64.1 | 102.0 | 53.1 | 54.9 | 54.9 | 3 | 3 | 4 | 10 |
| | 1E-04 | 80.0 | **58.1** | **61.2** | **62.6** | 95.9 | **49.9** | **49.9** | **49.9** | 4 | 4 | 7 | 13 |
| | 1E-02 | 81.0 | **63.0** | 65.7 | 105.6 | 91.8 | **47.8** | **47.8** | **47.8** | 9 | 9 | 12 | 98 |
| | 1E-01 | 84.4 | 64.9 | 65.9 | 130.5 | 89.9 | **48.7** | **48.7** | **48.7** | 19 | 19 | 16 | 154 |
| DrivAer6M | full-rank | 91.8 | 67.6 | 77.9 | 77.4 | 81.6 | 41.7 | 52.9 | 41.7 | 1 | 6 | 8 | 12 |
| | 1E-10 | **66.4** | 70.4 | 72.4 | 74.4 | 44.2 | 24.6 | 25.3 | 24.6 | 2 | 7 | 7 | 12 |
| | 1E-08 | **63.5** | **64.6** | **67.2** | 69.3 | 40.2 | 22.5 | 22.6 | 22.6 | 4 | 6 | 7 | 12 |
| | 1E-06 | **65.8** | **65.6** | **61.5** | **62.8** | 39.5 | 20.7 | 20.9 | 20.9 | 14 | 16 | 9 | 13 |
| | 1E-04 | — | — | 75.7 | 163.6 | — | — | 19.0 | 19.0 | — | — | 45 | 168 |
| | 1E-02 | — | — | 246.9 | 410.2 | — | — | **17.6** | **17.6** | — | — | 256 | 505 |
| | 1E-01 | — | — | 472.5 | 909.7 | — | — | 22.5 | **17.2** | — | — | 531 | 1171 |
| tminlet3M | FR | 294.5 | 136.2 | 157.9 | 176.3 | 241.0 | 121.0 | 169.9 | 121.0 | 1 | 7 | 15 | 56 |
| | 1E-10 | 341.9 | 187.7 | 202.9 | 206.9 | 176.0 | 114.0 | 169.9 | 114.0 | 2 | 7 | 16 | 48 |
| | 1E-08 | 299.9 | 181.2 | 194.1 | 200.2 | 156.0 | 110.0 | 161.9 | 110.0 | 3 | 10 | 16 | 53 |
| | 1E-06 | 245.8 | **117.1** | **127.6** | 136.9 | 133.0 | 78.0 | 94.2 | 78.0 | 6 | 7 | 16 | 61 |
| | 1E-04 | — | — | **124.6** | 155.5 | — | — | **59.2** | **55.8** | — | — | 85 | 283 |
| perf009ar | full-rank | 46.1 | 57.5 | 52.0 | 110.0 | 55.6 | 28.9 | 38.1 | 28.9 | 1 | 28 | 16 | 92 |
| | 1E-10 | **36.0** | 44.0 | 41.0 | 85.6 | 35.6 | 19.4 | 25.6 | 19.4 | 2 | 23 | 16 | 93 |
| | 1E-08 | **36.0** | 42.5 | **39.0** | 101.6 | 33.9 | **18.2** | 22.8 | **18.2** | 4 | 23 | 16 | 115 |
| | 1E-06 | — | — | 43.7 | 125.6 | — | — | 20.0 | **16.9** | — | — | 25 | 173 |
| | 1E-04 | — | — | 896.4 | — | — | — | 36.7 | — | — | — | 1284 | — |
| | 1E-02 | — | — | 2262.4 | — | — | — | 98.1 | — | — | — | 3344 | — |
| elasticity-3d | full-rank | 156.7 | — | 118.6 | — | 153.0 | — | 103.6 | — | 1 | — | 11 | — |
| | 1E-10 | 126.3 | — | 93.5 | — | 91.1 | — | 72.1 | — | 3 | — | 10 | — |
| | 1E-08 | 107.8 | — | 76.7 | — | 84.0 | — | 56.5 | — | 5 | — | 10 | — |
| | 1E-06 | — | — | **64.5** | — | — | — | 43.8 | — | — | — | 13 | — |
| | 1E-04 | — | — | 127.8 | — | — | — | **32.1** | — | — | — | 121 | — |
| lfm_aug5M | full-rank | 536.2 | 254.5 | 269.3 | 353.6 | 312.0 | 157.0 | 187.5 | 157.0 | 1 | 4 | 5 | 46 |
| | 1E-10 | 383.4 | 234.8 | 250.2 | 263.6 | 221.0 | 135.0 | 148.6 | 135.0 | 2 | 4 | 10 | 35 |
| | 1E-08 | 325.2 | 143.2 | 152.7 | 170.9 | 204.0 | 105.0 | 105.0 | 105.0 | 3 | 4 | 8 | 40 |
| | 1E-06 | 222.2 | **97.3** | 107.1 | 142.8 | 181.0 | 91.5 | 91.5 | 91.5 | 4 | 4 | 9 | 69 |
| | 1E-04 | 208.0 | **92.0** | 106.4 | 117.0 | 160.0 | 81.1 | 81.2 | 81.2 | 23 | 20 | 21 | 59 |
| | 1E-02 | — | — | 413.8 | 510.4 | — | — | **72.2** | **72.2** | — | — | 339 | 732 |
| CarBody25M | full-rank | **62.9** | — | 109.8 | — | 77.6 | — | 54.3 | — | 1 | — | 24 | — |
| | 1E-10 | **65.7** | — | 93.4 | — | 64.1 | — | **44.0** | — | 3 | — | 24 | — |
| | 1E-08 | 72.8 | — | 91.6 | — | 63.1 | — | **41.8** | — | 7 | — | 24 | — |
| | 1E-06 | — | — | 448.4 | — | — | — | 55.8 | — | — | — | 214 | — |
| thmgas | full-rank | 97.6 | **65.4** | 79.8 | 79.6 | 192.0 | 97.7 | 141.7 | 97.7 | 1 | 4 | 7 | 10 |
| | 1E-10 | 87.9 | 71.0 | 79.9 | 76.9 | 137.0 | 70.9 | 110.5 | 70.9 | 2 | 4 | 7 | 7 |
| | 1E-08 | 81.3 | **65.4** | 72.3 | 72.3 | 131.0 | 67.5 | 92.1 | 67.5 | 4 | 4 | 7 | 7 |
| | 1E-06 | 85.1 | **64.3** | **68.6** | 73.5 | 118.0 | 61.4 | 70.4 | 61.4 | 8 | 8 | 9 | 13 |
| | 1E-04 | — | — | 146.0 | 135.2 | — | — | 48.4 | 43.8 | — | — | 52 | 49 |
| | 1E-02 | — | — | 1081.9 | 3149.2 | — | — | 36.6 | **34.6** | — | — | 522 | 1353 |
| | 1E-01 | — | — | 2709.2 | 4624.8 | — | — | 48.6 | **32.5** | — | — | 1399 | 2480 |

It is expected that using the active memory compression shall increase the execution time and decrease the memory consumption of LU-IR3. While it will also increase the execution time of LU-GMRES-IR5, it might not necessarily decrease its peak memory consumption. Indeed, the memory peak of LU-GMRES-IR5 can happen during either the factorization in precision $u_f$ or the refinement steps where the factors are fully cast in precision $u_p$, as illustrated in Figure 6.2. Actually, both are possible in practice depending on the MUMPS settings, the problem, and the parallelization. In the first scenario, where the peak is reached during the factorization, the active memory compression reduces the memory consumption of LU-GMRES-IR5. Though, it is not the case in the second scenario where the peak is reached during the refinement steps. In addition, as stated in section 6.4.5.1, the relative weight of the non-compressed active memory increases with BLR, which helps LU-GMRES-IR5 to be more competitive regarding memory consumption when compared with LU-IR3. If the active memory is compressed, its relative weight will be lowered, and we might lose this previous positive property for LU-GMRES-IR5.

We emphasize that matrices DrivAer6M, tminlet3M, perf009ar, and elasticity-3D behave as expected: the best time among all the variants and threshold $\tau_b$ is worse with the active memory compression, but the best memory consumption is better. To illustrate this point, by looking specifically at tminlet3M, we can observe that the execution time increases from 88.3s to 117.1s (×0.75), and the memory consumption decreases from 70.9Go to 55.8Go (×1.27).

Some of the matrices do not match the expected behavior. For example, with ElectroPhys10M, we reduce both memory and time. It may be due to the reduction of the communications, which, from logs not reported in this document, decrease from 8.2Go to 0.9Go. With lfm_aug5M, we observe that for small $\tau_b$ the execution time is increased as expected, but the tendency is reversed from $\tau_b = 10^{-6}$ and higher, where the execution time starts to decrease. It could also be due to the reduction of communications which, for $\tau_b = 10^{-4}$, decrease from 13.8Go to 1.9Go. For CarBody25M, the execution time is slightly worse, but the memory consumption is not significantly improved. The reason is probably that, as the matrix does not compress well, the active memory cannot be compressed enough to obtain a noticeable gain.

*6.4.5.3 Static pivoting factorization.* We now turn our attention to the use of static pivoting. We report in Table 6.7 the execution time and memory consumption of three iterative refinement variants for different values of the static pivoting threshold $\tau_s$. All variants are stopped when they reach a forward error on the solution equivalent to the one of DMUMPS. If $\tau_s$ = "partial", the factorization is run in standard MUMPS threshold partial pivoting. It should be noted that in this case the double precision factorization LU-IR3 variant is equivalent to DMUMPS.

Once again the observed convergence behaviors are in good agreement with Theorems 6.1 and 6.2 as explained below. In the case of static pivoting, the execution time of the factorization does not depend on $\tau_s$; in order to minimize the overall solution time, the goal is therefore to achieve the fastest possible convergence. This is a complex issue: a smaller perturbation $\tau_s$ does not always mean a faster convergence, because the value of

Table 6.7: Execution time (in seconds) and number of LU solve calls of IR variants for a subset of the matrices listed in Table 6.1 and depending on the perturbation $\tau_s$. $\underline{\rho_n} = \max\{\max|L|, \max|U|\}/\max|A|$ is a lower bound of the growth factor. We fix $u_r = u = \text{D}$.

| | Solver | LU | LU | GMRES | LU | LU | GMRES | |
|---|---|---|---|---|---|---|---|---|
| | $u_f$ | D | S | S | D | S | S | |
| | $u_p$=$u_g$ | — | — | D | — | — | D | |
| ID | $\tau_s$ | Time (s) | | | Nb LU solves | | | $\underline{\rho_n}$ |
| ElectroPhys10M | partial | 265.2 | **154.0** | 166.5 | 1 | 3 | 5 | 9E-1 |
| | 1E-10 | 244.9 | **145.1** | 160.7 | 1 | 3 | 5 | 9E-1 |
| | 1E-08 | 244.9 | **145.1** | 160.7 | 1 | 3 | 5 | 9E-1 |
| | 1E-06 | 244.9 | **145.1** | 161.1 | 1 | 3 | 5 | 9E-1 |
| | 1E-04 | 244.9 | **145.1** | 161.1 | 1 | 3 | 5 | 9E-1 |
| | 1E-02 | 244.9 | **145.1** | 161.6 | 1 | 3 | 5 | 9E-1 |
| tminlet3M | partial | 294.5 | 136.2 | 157.9 | 1 | 7 | 15 | 4E2 |
| | 1E-10 | 258.1 | **121.0** | 141.6 | 2 | 9 | 16 | 2E4 |
| | 1E-08 | 258.1 | **121.0** | 141.5 | 2 | 9 | 16 | 2E4 |
| | 1E-06 | 258.1 | **121.0** | 144.7 | 2 | 9 | 18 | 2E4 |
| | 1E-04 | — | — | 1659.9 | — | — | 985 | 4E3 |
| lfm_aug5M | partial | 536.2 | **254.5** | **269.3** | 1 | 4 | 5 | 5E4 |
| | 1E-10 | — | — | — | — | — | — | 2E9 |
| | 1E-08 | 508.0 | — | — | 7 | — | — | 2E7 |
| | 1E-06 | 490.3 | — | — | 3 | — | — | 2E5 |
| | 1E-04 | 499.2 | — | 773.2 | 5 | — | 124 | 2E3 |
| | 1E-02 | 1501.5 | 780.3 | 484.9 | 231 | 233 | 59 | 5E3 |
| thmgas | partial | 97.6 | **65.4** | 79.8 | 1 | 4 | 7 | 2E0 |
| | 1E-10 | 88.7 | **63.9** | 78.9 | 1 | 4 | 7 | 2E0 |
| | 1E-08 | 88.7 | **63.9** | 78.9 | 1 | 4 | 7 | 2E0 |
| | 1E-06 | 88.7 | **63.9** | 78.9 | 1 | 4 | 7 | 2E0 |
| | 1E-04 | 88.7 | **63.9** | 78.9 | 1 | 4 | 7 | 2E0 |
| | 1E-02 | 113.4 | 110.8 | 109.8 | 9 | 23 | 17 | 2E0 |

$\tau_s$ also directly impacts the growth factor $\rho_n$. Thus, there is an optimal value of $\tau_s$, which is clearly problem dependent, that leads to the fastest convergence by balancing the $u_f\rho_n$ and $\tau_s$ terms in the convergence condition. To confirm this, Table 6.7 reports $\underline{\rho_n}$, a lower bound on the true growth factor, that can be used as a cheap, but rough indicator of the behavior of $\rho_n$ (the true $\rho_n$ would be extremely expensive to compute for such large matrices). There is a clear trend of $\underline{\rho_n}$ decreasing as $\tau_s$ increases, which explains, for example, why on lfm_aug5M convergence is achieved for large $\tau_s$. For many matrices in our set, such as tminlet3M in Table 6.7, static pivoting slightly accelerates the factorization without excessively deteriorating the convergence, and so allows for modest time gains overall. However, for some matrices such as lfm_aug5M, static pivoting requires many iterations and can lead to significant slowdowns compared with partial pivoting. It is however interesting to note that, on lfm_aug5M, if the use of partial pivoting is impossible (for instance because the available solver does not support it), the LU-GMRES-IR5 variant provides the best overall execution time.

Table 6.8: Execution time (in seconds) and number of LU solve calls of IR variants for a subset of the matrices listed in Table 6.1 and depending on the perturbation $\tau_b$ for a fixed $\tau_s$. The chosen $\tau_s$ is specified for each matrices. We fix $u_r = u =$ D.

| | | Solver | LU | LU | GMRES | LU | LU | GMRES |
|---|---|---|---|---|---|---|---|---|
| | | $u_f$ | D | S | S | D | S | S |
| | | $u_p$=$u_g$ | — | — | D | — | — | D |
| ID | $\tau_s$ | $\tau_b$ | Time (s) | | | Nb LU solves | | |
| ElectroPhys10M | partial | 1E-10 | 101.0 | **70.5** | 73.7 | 2 | 3 | 5 |
| | | 1E-08 | 93.1 | **70.0** | 72.8 | 2 | 3 | 5 |
| | | 1E-06 | 91.1 | **64.9** | **68.2** | 3 | 3 | 6 |
| | | 1E-04 | 88.3 | **66.3** | **69.3** | 4 | 4 | 7 |
| | | 1E-02 | 89.8 | 71.4 | 75.0 | 9 | 9 | 11 |
| | $10^{-8}$ | 1E-10 | 70.0 | **54.3** | 57.5 | 2 | 3 | 5 |
| | | 1E-08 | 66.6 | **52.4** | **55.3** | 2 | 3 | 5 |
| | | 1E-06 | 64.9 | **51.4** | **53.3** | 3 | 3 | 4 |
| | | 1E-04 | 73.1 | **51.0** | **53.7** | 5 | 5 | 7 |
| | | 1E-02 | 69.6 | 56.3 | 58.8 | 11 | 11 | 12 |
| tminlet3M | partial | 1E-10 | 232.9 | 158.8 | 174.0 | 2 | 7 | 16 |
| | | 1E-08 | 204.9 | 149.7 | 165.3 | 3 | 7 | 17 |
| | | 1E-06 | 179.0 | **88.3** | 98.8 | 5 | 7 | 16 |
| | | 1E-04 | — | — | 105.6 | — | — | 69 |
| | $10^{-8}$ | 1E-10 | 196.9 | 139.9 | 152.2 | 2 | 9 | 17 |
| | | 1E-08 | 181.9 | 133.7 | 149.8 | 3 | 9 | 21 |
| | | 1E-06 | 137.9 | **70.1** | 80.0 | 6 | 12 | 18 |
| | | 1E-04 | — | — | 90.9 | — | — | 71 |
| lfm_aug5M | partial | 1E-10 | 313.3 | 199.8 | 210.0 | 2 | 4 | 7 |
| | | 1E-08 | 260.2 | 119.2 | 130.1 | 3 | 4 | 9 |
| | | 1E-06 | 223.2 | **100.4** | 110.1 | 4 | 4 | 9 |
| | | 1E-04 | 212.3 | **95.8** | 105.4 | 22 | 20 | 19 |
| | | 1E-02 | — | — | 482.6 | — | — | 367 |
| | $10^{-2}$ | 1E-10 | 592.9 | 353.8 | 218.2 | 231 | 233 | 59 |
| | | 1E-08 | 525.8 | 266.6 | 163.6 | 231 | 233 | 59 |
| | | 1E-06 | 456.1 | 247.3 | 138.1 | 231 | 233 | 59 |
| | | 1E-04 | 404.6 | 212.8 | **123.1** | 238 | 238 | 63 |
| | | 1E-02 | — | — | 879.1 | — | — | 838 |
| | $10^{-6}$ | 1E-10 | 253.5 | — | — | 3 | — | — |
| | | 1E-08 | 200.2 | — | — | 3 | — | — |
| | | 1E-06 | **157.2** | — | — | 4 | — | — |
| | | 1E-04 | **166.4** | — | — | 33 | — | — |

*6.4.5.4 BLR factorization with static pivoting.* Finally in Table 6.8 we present the execution time and memory consumption of three iterative refinement variants (the same as in section 6.4.5.3) for different values of the BLR compression threshold $\tau_b$ and a fixed value of the static pivoting perturbation $\tau_s$. All variants are stopped when they reach a forward error on the solution equivalent to the one of DMUMPS. If $\tau_s$ = partial, the factorization is run in standard MUMPS threshold partial pivoting and the results are then equivalent to

the BLR results of section 6.4.5.1.

Theorems 6.1 and 6.2 applied to the case where BLR and static pivoting are used together states that the convergence conditions should be affected by the largest perturbations $\max(\tau_s, \tau_b)$ and the term $\rho_n u_f$ which depends on the growth factor. Our experiments confirm this: values of $\tau_s$ or $\tau_b$ for which a given variant was not converging with BLR or static pivoting alone still do not converge when they are combined, and, conversely, variants that were converging for BLR and static pivoting alone still converge when these two approximations are used together. lfm_aug5M with $\tau_s = 10^{-6}$ illustrates an interesting point of the error bound $\max(\tau_s, \tau_b) + \rho_n u_f$: convergence is only achieved for the variant that uses a double precision factorization ($u_f = \mathrm{D}$), even for values of $\tau_b$ that are much larger than the unit roundoff of single precision. This shows that the rule of thumb that the factorizaton precision should be chosen as low as possible as long as $u_f \leq \tau_b$ is not true in presence of large element growth, since a smaller value of $u_f$ can be beneficial to absorb a particularly large $\rho_n$.

While the reductions in execution time obtained by using static pivoting instead of partial pivoting were modest for the full-rank factorization, they are larger for the BLR factorization. Taking tminlet3M as an example, in full-rank the single precision factorization LU-IR3 variant is only 1.12 (136s/121s) times faster after the activation of the static pivoting (see Table 6.7), whereas in BLR it is 1.26 (88s/70s) times faster than (see Table 6.4). These better reductions are explained by the fact that in the BLR factorization, static pivoting also allows the panel reduction operation to be processed with low-rank operations (Amestoy et al. [17]), which leads to a reduction of flops and thus a faster execution time.

### 6.4.6 Performance summary

To summarize the results presented in the previous sections, we report in Table 6.9 the best execution time and memory consumption amongst all the previously reviewed iterative refinement variants, for the industrial partners matrices and Queen_4147. All variants are stopped when they reach a forward error on the solution equivalent to the one of DMUMPS. We do not activate the active memory compression with BLR.

We obtain at best on lfm_aug5M a reduction of 5.6× in time and on thmgas a reduction of 4.4× in memory. A greater variability is observed in the speedup with respect to the memory gains. This is because numerous parameters affect the execution time which are related to the numerical properties of the problems as well as to the features of the computer architecture; in some extreme cases (such as CarBody25M) no speedup is observed at all. As the best memory saving is sometimes obtained for aggressive values of the BLR threshold, the execution time can be deteriorated due to a high number of iterations. We however note that a balance between the two use cases can be struck to obtain large memory savings while keeping a reasonable execution time: taking thmgas as an example, we can achieve a 3.6× memory reduction (compared with the 2.8× reduction of the “best in time” variant) while only leading to a 0.7× slowdown (compared with the 0.03× slowdown of the “best in memory” variant).

Table 6.9: Best execution time and memory consumption improvements in comparison to DMUMPS amongst all the presented IR variants (full-rank, BLR, static pivoting, and BLR with static pivoting) for the industrial partners matrices (bold in Table 6.1) and Queen_4147.

| ID | DMUMPS | | Best in time | | Best in memory | |
|---|---|---|---|---|---|---|
| | Time (s) | Memory (s) | Time (s) | Memory (s) | Time (s) | Memory (s) |
| ElectroPhys10M | 265.2 | 272.0 | 51.0 (5.2×) | 73.0 (3.7×) | 80.7 (3.3×) | 71.2 (3.8×) |
| DrivAer6M | 91.8 | 81.6 | 37.8 (2.4×) | 26.9 (3.0×) | 471.3 (0.2×) | 22.4 (3.6×) |
| Queen_4147 | 284.2 | 178.0 | 60.1 (4.7×) | 50.7 (3.5×) | 117.5 (2.4×) | 50.3 (3.5×) |
| tminlet3M | 294.5 | 241.0 | 70.1 (4.2×) | 82.4 (2.9×) | 105.6 (2.8×) | 70.9 (3.4×) |
| perf009ar | 46.1 | 55.6 | 30.8 (1.5×) | 38.6 (1.4×) | 187.8 (0.2×) | 18.6 (3.0×) |
| elasticity-3d | 156.7 | 153.0 | 56.0 (2.8×) | 41.9 (3.7×) | 125.3 (1.3×) | 39.0 (3.9×) |
| lfm_aug5M | 536.2 | 312.0 | 95.8 (5.6×) | 101.0 (3.1×) | 879.1 (0.6×) | 91.6 (3.4×) |
| CarBody25M | 62.9 | 77.6 | 62.9 (1.0×) | 77.6 (1.0×) | 91.3 (0.7×) | 41.8 (1.9×) |
| thmgas | 97.6 | 192.0 | 50.7 (1.9×) | 67.5 (2.8×) | 3155.2 (0.0×) | 43.7 (4.4×) |

### 6.4.7 Scalability and parallelism

Finally, in this section, we propose to study the performance of LU-IR3 and LU-GMRES-IR5 when we increase the parallelism, the computer resources, and the size of the problem. We will focus on standard LU factorization as in section 6.4.4, and we will not consider the approximation techniques covered in section 6.4.5.

*6.4.7.1 Multithreading.* The first source of parallelism in LU-IR3 and LU-GMRES-IR5 is the multithreading which allows for the use of multiple cores in each MPI process. The MUMPS solver uses multithreading to exploit node parallelism during the factorization (see section 2.2.3.6); that is, to parallelize the partial factorizations of the nodes of the assembly tree. With the advanced multithreading option of L'Excellent and Sid-Lakhdar [141], it can also be used to exploit tree parallelism; that is, to work concurrently on different tree nodes. Multithreading is also used for more negligible operations inside our iterative refinement algorithms, such as the SpMV, the explicit cast of the factors, or some other dense operations occurring in the GMRES solver.

We represent in Figure 6.6 the execution times of LU-IR3 and LU-GMRES-IR5 for 1, 9, and 18 threads per MPI for each matrix of our set and normalized by that of DMUMPS. Note that the execution times with the 18 threads configuration correspond to the experiments in Table 6.3.

We can observe that generally, for both LU-IR3 and LU-GMRES-IR5, when more and more threads are used, the speedup decreases. While on specific matrices, the difference between 1 and 18 threads is small (e.g., approximately 5% loss on Queen_4147), for many other matrices, it can be substantial (e.g., approximately 20% loss on thmgas). This observation originates from two combined phenomenons:

- The factorization operation has better scalability than the solve. Hence, as LU-IR3 and LU-GMRES-IR5 apply multiple solves during the refinement steps, the relative cost of the refinement steps increases as we increase the number of threads.

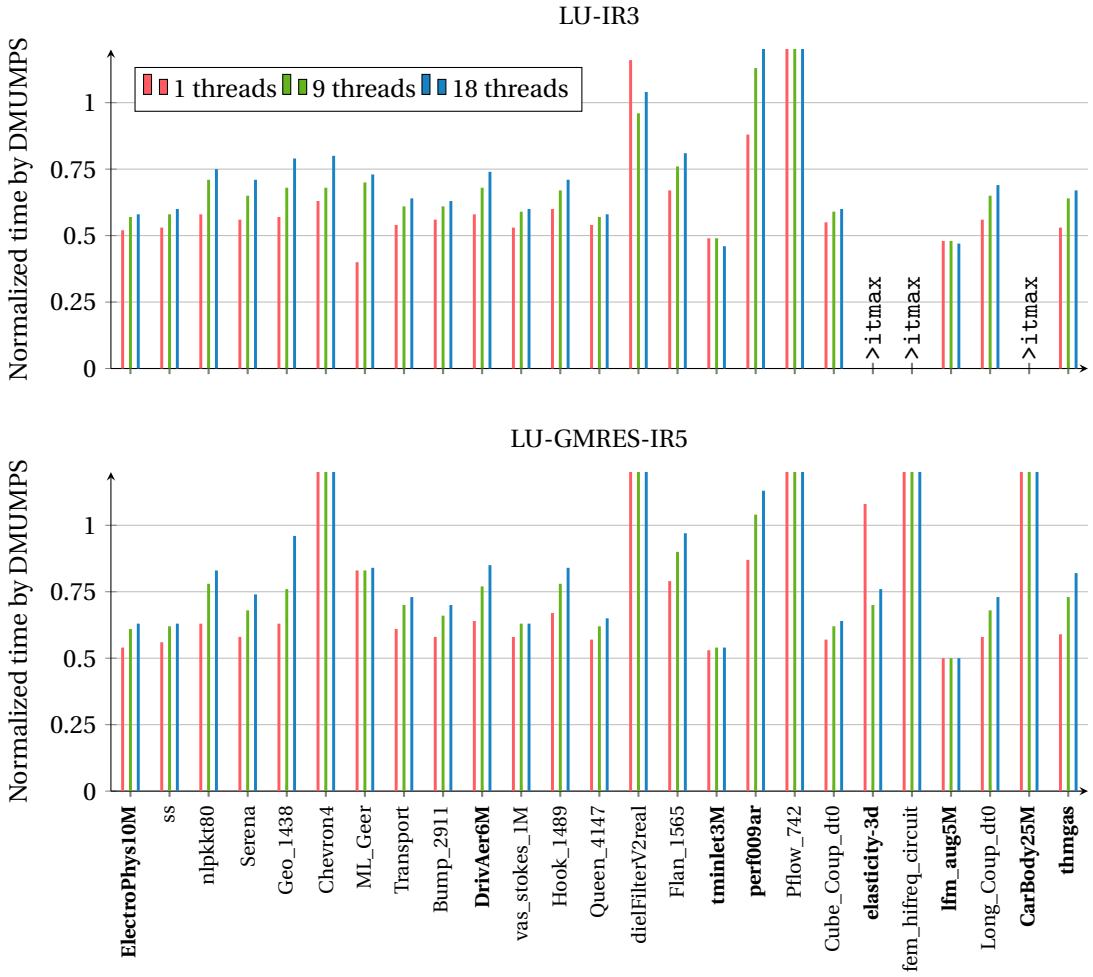



Figure 6.6: Execution time of LU-IR3 and LU-GMRES-IR5 with $u_f = \text{S}$ and $u = u_r = u_g = u_p = \text{D}$ normalized by that of DMUMPS for 1, 9, and 18 threads per MPI and for the whole set of matrices listed in Table 6.1. The forward error is equivalent to the one obtained with DMUMPS. The number of MPI used for each matrix is given in Table 6.1.

- The single precision factorization operation is less scalable than its double precision counterpart. Indeed, the factorization in double precision has approximately double the number of flops and a better granularity; it can therefore achieve better scalability. For this reason, the time cost of the single precision factorization relative to the double one increases as we increase the number of threads.

*6.4.7.2 Strong scaling.* The other source of parallelism is the use of multiple MPI processes. As our in-house implementations of the SpMV and the GMRES solver works only on the master MPI, this parallelism only concerns the MUMPS factorization, the MUMPS solve, and our explicit cast. The MPI parallelism in the MUMPS solver is used to exploit tree and node parallelism (see section 2.2.3.6); therefore, the more the MPI processes are used, the more fronts of the elimination tree are computed concurrently.

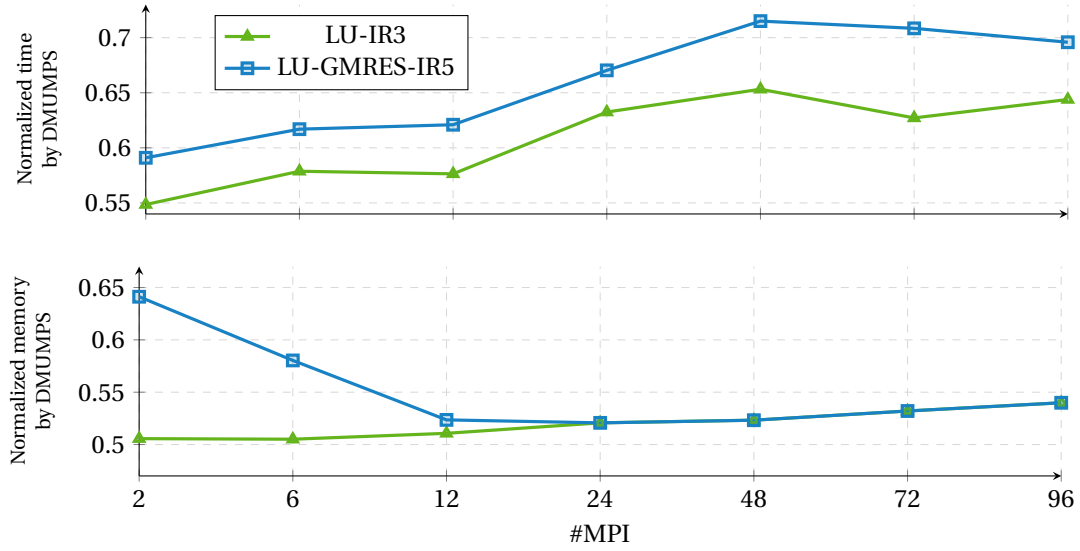



Figure 6.7: Execution time and memory consumption of LU-IR3 and LU-GMRES-IR5 with $u_f = \text{S}$ and $u = u_r = u_g = u_p = \text{D}$ normalized by that of DMUMPS with an increasing number of MPI for ElectroPhys10M. The forward error is equivalent to the one obtained with DMUMPS. Each MPI uses 6 threads.

In Figure 6.7, we present a strong scaling experiment on ElectroPhys10M, where we record the execution time and memory consumption of both LU-IR3 and LU-GMRES-IR5 using an increasing number of MPI (from 2 to 96). Each MPI uses 6 threads.

In terms of time, we can observe that, as for the multithreading, when we increase the number of MPI processes, our iterative refinement variants get less competitive compared with DMUMPS. Indeed, LU-GMRES-IR5 goes from 60% of the DMUMPS execution time to 70%. The same reasons mentioned above can explain it: double precision factorization is more scalable than its single precision counterpart, and the solve operation is less scalable than the factorization.

In terms of memory, we observe that from 24 MPI, LU-GMRES-IR5 becomes as efficient as LU-IR3. This situation arises because the active memory consumption increases when we increase the MPI parallelism in MUMPS. Indeed, processing more frontal matrices concurrently means that more contribution blocks will be stored concurrently. When the active memory overhead is high enough such that the single precision factorization peak is higher than the factors in double precision (and considering that the Krylov basis is negligible), LU-GMRES-IR5 will consume as much memory as LU-IR3. It is a very appealing property because, when parallelism increases, the difference of memory consumption between LU-GMRES-IR5 and LU-IR3, which is one of the main drawbacks of LU-GMRES-IR5, is expected to fade.

*6.4.7.3 Weak scaling.* We now want to observe how our iterative refinement variants scale when we increase the dimension of the problem and the number of MPI processes. We choose the number of MPI such that the average memory consumption per MPI is approx-

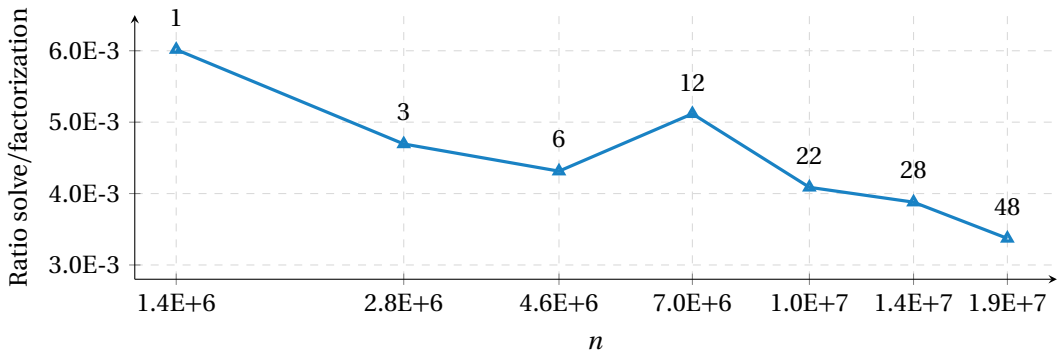



Figure 6.8: Ratio between the DMUMPS solve and factorization operations for increasing dimension of a cubic Helmholtz problem. The numbers on top of the points are the numbers of MPI used. Each MPI uses 18 threads. The memory consumption of DMUMPS per MPI is approximately constant.

imately constant.

In Figure 6.8, we present the evolution of the ratio between the solve and the factorization operations of DMUMPS for problems of increasing dimension. This ratio is a good indicator of the relative weight of the refinement steps in the iterative refinement algorithms. Therefore, the lower the ratio, the more negligible the refinement steps are and the better the execution time. We expect this ratio to get lower as the dimension increases since we know that the operational complexities of the factorization and solve operations for sparse 3D problems are respectively $\mathcal{O}(n^2)$ and $\mathcal{O}(n^{4/3})$ (see Table 2.2). Hence, as $n$ increases, the solve should be more and more negligible. It is what we observe on Figure 6.8; indeed, between the first and the last point of the figure (i.e., $n = 1.4\text{E+}6$ to $n = 1.9\text{E+}7$), the dimension is more than ten times larger, and the relative cost of the solve is reduced by two.

Therefore, our variants of iterative refinement are expected to perform better in time relative to the direct solver when the problem size increases. It is a very appealing property of these algorithms. Naturally, the relative weight of the refinement steps is also affected by the structure of the sparse problem. For example, for 2D problems, where the ratio between the factorization and the solve is expected to be smaller (resp. operational complexities are $\mathcal{O}(n^{3/2})$ and $\mathcal{O}(n \log n)$, see Table 2.2), the relative weight of the refinement steps will decrease more slowly than for 3D problems. Note that this still holds for data sparse solvers (see Table 2.3). We also point out that it is unclear how much the dimension of the problem affects the number of iterations. Naturally, if the number of iterations increases too much with the dimension, it might mitigate this desirable effect.

## 6.5 Conclusions

In this chapter, we have evaluated the potential of mixed precision iterative refinement to improve the solution of large sparse systems with direct methods. Compared with dense systems, sparse ones present some challenges but also, as we have explained, some specific

features that make iterative refinement especially attractive. In particular, the fact that the LU factors are much denser than the original matrix makes the computation of the residual in quadruple precision arithmetic affordable and leads to potentially significant memory savings compared with a standard direct solver. Moreover, iterative refinement can remedy potential instabilities in factorization, which modern sparse solvers often introduce by using numerical approximations and relaxed pivoting strategies. To assess these benefits, we propose to combine some of the most recent variants of iterative refinement (i.e., LU-IR3 and LU-GMRES-IR5 reviewed previously in this manuscript) with state-of-the-art approximate sparse factorizations employing low-rank approximations and static pivoting.

In the first instance, we derived, in Theorems 6.1 and 6.2, new bounds that take into account numerical approximations in the factorization as well as a possibly large element growth due to relaxed pivoting. These bounds better correspond to the typical use of sparse solvers, and we have observed them to be in good accordance with the experimental behavior, at least in the case of BLR approximations, static pivoting, and their combination.

We then provided an extensive experimental study of several iterative refinement variants combined with different types of approximate factorization using the multifrontal solver MUMPS. Our experiments demonstrate the potential of mixed precision iterative refinements to reduce the execution time and memory consumption of sparse direct solvers and shed light on important features of these methods. In particular, we have shown that LU-IR3 with a standard factorization in single precision can reduce time and memory by up to 2× compared with a double precision direct solver. We have found LU-GMRES-IR5 to be usually more expensive but also more robust than LU-IR3, which allows it to converge on very ill-conditioned problems and still achieve gains in memory, and sometimes even in time. Moreover, we have combined single precision arithmetic with BLR and static pivoting and analyzed how the convergence of iterative refinement depends on their threshold parameters. Overall, compared with the double precision direct solver, we have obtained reductions of up to 5.6× in time and 4.4× in memory, all while preserving double precision accuracy. Moreover, we have shown that memory consumption can be even further reduced at the expense of time by using LU-GMRES-IR5 with more aggressive approximations. Finally, we have discussed the scaling properties of our iterative refinement variants; we observed two interesting conflicting effects: increasing the parallelism might reduce the relative time gain while increasing the dimension might increase it. In addition, we also remarked that increasing the parallelism reduces the memory consumption gap between LU-IR3 and LU-GMRES-IR5 to some point where, on our example, LU-GMRES-IR5 becomes as efficient as LU-IR3.

These results open up promising avenues of research as half precision arithmetic becomes progressively available in hardware and supported by compilers.

Most of this chapter is the object of the preprint *"Combining sparse approximate factorizations with mixed precision iterative refinement"* (Amestoy et al. [19]).

# 7 Iterative refinement with preconditioned GMRES

So far, we have mainly been interested in improving sparse direct solvers with mixed precision iterative refinement algorithms. Another very popular way to solve sparse linear systems is using iterative methods (see section 2.3), particularly when the dimension of the problems is too high to be solved efficiently by direct approaches. In this chapter, we propose a new mixed precision iterative refinement algorithm for the GMRES iterative solver that covers most of the previous state-of-the-art mixed precision strategies for this algorithm and that we call GMRES-based iterative refinement with arbitrary preconditioner $M$ in six precisions (M-GMRES-IR6).

The content of this chapter is structured as follows: in section 7.1, we review most of the different mixed precision strategies used for Krylov subspace iterative solvers. In section 7.2, we study a new version of left-preconditioned MGS-GMRES in mixed precision. In section 7.3, we use this new version of MGS-GMRES to derive results for M-GMRES-IR6 that we validate with numerical experiments.

## 7.1 State-of-the-art mixed precision strategies for GMRES

Chapters 5 and 6 specifically focus on the improvement of direct solvers through mixed precision iterative refinement algorithms. However, the use of mixed precision strategies for the solution of the linear system (1.1) has been actively studied for several kinds of solvers: direct (e.g., Carson and Higham [45], Haidar et al. [103], Amestoy et al. [18]), Krylov based (e.g., Turner and Walker [205], Arioli and Duff [25], Gratton et al. [95], Carson et al. [48]), or multigrid (e.g., Göddeke et al. [87], Oo and Vogel [172], McCormick et al. [159]). This chapter focuses on improving Krylov based solvers with mixed precision, more particularly it

focuses on the GMRES solvers for the solutions of general square linear systems we covered in section 2.3.1. The use of mixed precision in these solvers has already been the topic of many papers. This is why, in this section, we propose to list a representative sample of them that covers most of the different mixed precision strategies inside Krylov subspace based solvers. We can classify the different state-of-the-art strategies based on three criteria.

The first criterion is whether the method uses a single level of iteration or multiple levels, such as inner–outer approaches which embed two levels. These approaches are based on using an inner solver applied with low precision inside an iterative solver applied with higher precision. The idea is that the most computationally intensive part is concentrated in the inner solver in low precision, and the outer solver in high precision serves to improve the accuracy of the computed solution cheaply. The first work on the topic is from Turner and Walker [205], who applied an outer iterative refinement in high precision combined with an inner GMRES solver for the solution of the correction equation in a lower precision. This specific approach is very popular and has been reused for CG and GMRES by Strzodka and Göddeke [198], Anzt et al. [21; 22], Lindquist et al. [146; 147], Loe et al. [150; 151]. Buttari et al. [42] proposed a strategy where the outer and inner solvers are both CG or GMRES; we can also see this setting as a GMRES preconditioned by a lower precision GMRES (resp. CG). To use GMRES as a preconditioner, we need to employ the flexible formulation of the right-preconditioned GMRES (i.e., FGMRES, see section 2.3.2.1) because the application of the inner GMRES results in a nonconstant operator $M^{-1}$ from an outer iteration to another.

The second criterion is to choose or not to run some or each kernel of the iterative solver (i.e., the orthonormalization, the SpMV, or the preconditioner) with different precisions. Some works apply all of these kernels in the same precision, such as Turner and Walker [205]; in this case, the mixed precision only comes from the variations between the inner and outer levels. However, many other works rely on using different precisions for different kernels: in the orthonormalization process (e.g., the work of Yamazaki et al. [215], Carson et al. [48], Balabanov and Grigori [30]), in the SpMV kernel (e.g., the work of Graillat et al. [94]), or in storing the basis (e.g., the work of Aliaga et al. [9], Agullo et al. [7]). In particular, many studies were interested in computing, storing, and applying the preconditioner of a preconditioned GMRES (see section 2.3.2) in various precisions. For instance, the computation of the preconditioner in low precision is particularly efficient for preconditioners based on a factorization of the matrix $A$. It is because, generally, the computation of the (approximate) factors, done once before the start of the iterations, represents a costly step in flops and memory. Of course, the LU preconditioner reviewed in section 2.3.2.2 belongs to this category, but for instance, this strategy is also valid for ILUT (see section 2.3.2.3) or block Jacobi (see section 2.3.2.4) which can be both based on some approximate factorizations of $A$. Actually, if the whole factors need to be computed, the factorization could certainly be the dominant operation and, therefore, offers a large avenue to save resources with low precision. For example, the computation and the application of the full factors in low precision have been studied by Arioli and Duff [25], Hogg and Scott [123] for FGMRES. The same strategy can be employed by replacing the low precision factorization with an approximate factorization, such as static pivoting (see section 2.2.4.2) as proposed by Arioli et al. [27]. Anzt et al. [23] used a block Jacobi preconditioner for CG where each block is

stored in low precision and cast on the fly to be applied at the same precision as the other operations of CG. In contrast to all these approaches, where the preconditioner is either applied with low precision or with the same precision as the other operations, the opposite strategy of applying it with extra precision has also been proposed. For instance, it is the approach employed by LU-GMRES-IR3 developed by Carson and Higham [44; 45], which computes the factors with low precision but applies them with high. We improved this strategy with LU-GMRES-IR5 proposed and studied in chapters 5 and 6. In particular, we have seen that such a strategy is more robust regarding the conditioning of the problem. Further improvements for this specific strategy were proposed. For example, by Oktay and Carson [171], which used a recycling variant of GMRES to reduce the number of iterations, or by Carson and Khan [46], which employs an approximate inverse preconditioner instead of the LU preconditioner, showing that this strategy also works for preconditioners that are not necessarily based on the factorization of $A$.

Finally, the third criterion is whether the choice of precisions is static throughout all the iterations or is allowed to change as the iterations go. It has been explored by Gratton et al. [95], who demonstrated that it was possible to reduce the precision on the matrix–vector products and the scalar products (i.e., steps 3 and 5 of the Arnoldi Algorithm 2.5) as we lose orthogonality on the basis due to rounding errors (theory of inexact Krylov). Another approach is by Oktay and Carson [170], who showed that it is possible to increase progressively the internal precision $u_g$ and $u_p$ of LU-GMRES-IR5 if the accuracy on the solution stagnates or diverges. Hence, the method starts with low precision $u_g$ and $u_p$; if it converges appropriately, the method does not switch the precision; otherwise, it does, and it repeats the process in such a way that the problem is guaranteed to be solved with the lowest precisions possible. Our last example of strategy is by Loe et al. [150; 151], who proposed to restart the whole GMRES from low precision to higher precision at some point in the iterations. This strategy comes from the intuition that we can use a cheaper low precision GMRES to produce a low precision accuracy solution, and restart in more costly full high precision GMRES to reach high precision accuracy.

We summarize in Table 7.1 most of the previously mentioned contributions; we do not consider mixed precision in the orthonormalization process or SpMV kernel. We classify them by preconditioner used (LU, ILU, Jacobi, and others), by way of applying the preconditioner (left- or right-preconditioning), and finally, if they are using an outer iterative refinement process or not.

In the wide range of studies listed in Table 7.1, we can identify one critical issue: every proposition is specialized for one kind of preconditioner, one way of applying it, and uses or not an iterative refinement outer solver. Unfortunately, this substantial number of different and (for most of them) seemingly unrelated approaches can be confusing and leads to the following reasonable practical questions: how to choose one of these propositions? How are they linked and different? Are they consistent with each other? Etc. In other words, we lack a general theoretical framework that covers an extensive collection of preconditioners, their different ways of application, and the use or not of outer iterative refinement.

In this chapter, we develop a framework that aims to gather most of these approaches under the same coherent analysis from which we can compare and decide which mixed

Table 7.1: Summary of existing scientific papers using mixed precision with GMRES or CG classified by preconditioner used and if it is combined or not with a refinement process. We display in red the papers using left-preconditioning, in blue the papers using right-preconditioning, and in black the papers which are not using any preconditioner.

| | LU | ILU | Jacobi | Polynomial | Approximate inverse | GMRES /CG | None |
|---|---|---|---|---|---|---|---|
| IR | [44] [45] [18] [19] [170] [171] [123] | [146] [147] | [147] [150] [151] | [150] [151] | [46] | | [205] [198] [21] [22] |
| No IR | [25] | | [23] | | | [42] | [95] |

precision strategy is the best for a given use case. Accordingly, the chapter is organized into two main parts. First, we study a new preconditioned MGS-GMRES in mixed precision that uses four independent precisions. With this new algorithm, we essentially propose to decouple the applications of an arbitrary preconditioner $M^{-1}$ and the original matrix $A$ into possibly two different precisions. Second, we add on top of MGS-GMRES in mixed precision an iterative refinement process to obtain an algorithm called M-GMRES-IR6. This new algorithm uses six independent precisions and can compute a high-accuracy solution of a potentially ill-conditioned linear system, making it robust, efficient, and versatile. By using an arbitrary preconditioner, an iterative refinement process, and decoupling the precision of the preconditioned matrix–vector products, M-GMRES-IR6 covers most of the mixed precision strategies listed in Table 7.1. In addition, it enables new mixed precision strategies that have not been covered in previous work, and that can achieve improvements in time and memory consumption.

With M-GMRES-IR6, we restrict ourselves to a fixed set of precisions that do not change throughout the iterations. However, we believe that the adaptive precisions strategies presented previously can be nicely extended on top of our algorithms. Moreover, while we will present the right-preconditioned versions of our algorithms, this chapter's error analyses and numerical experiments are only centered on the left-preconditioned versions. Studying the right-preconditioned version is certainly of interest, but because the analysis on the left-preconditioned case is already dense and cannot be extended straightforwardly to the right case which needs its own analysis, we leave it for future work. Finally, we mainly focus our analysis and experiments on the GMRES solver. However, we believe that the framework we are proposing can be extended to some extent to other Krylov subspace based iterative solvers.

## 7.2 Left-preconditioned MGS-GMRES in mixed precision

We describe in Algorithm 7.1 a left- and right-preconditioned MGS-GMRES applying its operations in four independent precisions $u_f$, $u_g$, $u_m$, and $u_a$ that we refer to as *left or right MGS-GMRES in mixed-precision* in this chapter. Two points should be noted. First,

Algorithm 7.1 is based on the MGS orthonormalization, but this mixed precision strategy can be adapted to other sorts of orthonormalization processes such as CGS or Householder (see section 2.3.1.2). Second, the preconditioner $M$ is arbitrary, meaning that Algorithm 7.1 can be specialized for any kind of preconditioner, in particular, with the ones used in the previous mixed precision GMRES approaches listed in Table 7.1.

**Algorithm 7.1** Left-preconditioned MGS-GMRES in mixed precision

**Input:** an $n \times n$ matrix $A$ and a preconditioner $M$, a right-hand side $b$, and a number of iteration $k$.
**Output:** a computed solution to $Ax = b$.

| | | | | | |
|---|---|---|---|---|---|
| 1: | Compute $M$ (Optional) | $(u_f)$ | 1: | Compute $M$ (Optional) | $(u_f)$ |
| 2: | Initialize $x_0$ | | 2: | Initialize $x_0$ | |
| 3: | $r_0 = Ax_0 - b$ | $(u_a)$ | 3: | $r_0 = Ax_0 - b$ | $(u_a)$ |
| 4: | $\boldsymbol{s_0 = M^{-1} r_0}$ | $(u_m)$ | 4: | | |
| 5: | $\beta = \|s_0\|, \quad v_1 = s_0/\beta, \quad$ j=0 | $(u_g)$ | 5: | $\beta = \|r_0\|, \quad v_1 = r_0/\beta, \quad$ j=0 | $(u_g)$ |
| 6: | **for** $j = 1 : k$ **do** | | 6: | **for** $j = 1 : k$ **do** | |
| 7: | $z_j = Av_j$ | $(u_a)$ | 7: | $\boldsymbol{z_j = M^{-1} v_j}$ | $(u_m)$ |
| 8: | $\boldsymbol{w_j = M^{-1} z_j}$ | $(u_m)$ | 8: | $w_j = Az_j$ | $(u_a)$ |
| 9: | **for** $l = 1 : j$ **do** | | 9: | **for** $l = 1 : j$ **do** | |
| 10: | $h_{l,j} = v_l^T w_j$ | $(u_g)$ | 10: | $h_{l,j} = v_l^T w_j$ | $(u_g)$ |
| 11: | $w_j = w_j - h_{l,j} v_l$ | $(u_g)$ | 11: | $w_j = w_j - h_{l,j} v_l$ | $(u_g)$ |
| 12: | **end for** | | 12: | **end for** | |
| 13: | $h_{j+1,j} = \|w_j\|$ | $(u_g)$ | 13: | $h_{j+1,j} = \|w_j\|$ | $(u_g)$ |
| 14: | $v_{j+1} = w_j / h_{j+1,j}$ | $(u_g)$ | 14: | $v_{j+1} = w_j / h_{j+1,j}$ | $(u_g)$ |
| 15: | **end for** | | 15: | **end for** | |
| 16: | $y_k = \arg\min_y \|\beta e_1 - H_k y\|$ | $(u_g)$ | 16: | $y_k = \arg\min_y \|\beta e_1 - H_k y\|$ | $(u_g)$ |
| 17: | $x_k = x_0 + V_k y_k$ | $(u_g)$ | 17: | $\boldsymbol{x_k = x_0 + Z_k y_k}$ | $(u_m)$ |

The different precisions are defined as follows. The precision $u_a$ is used to compute the matrix–vector products with the matrix $A$, the precision $u_m$ is used for the applications of the preconditioner $M^{-1}$, and the precision $u_g$ is used to compute the rest of the operations. The precision $u_f$ is the precision at which the preconditioner is computed at step 1. This step is optional since every preconditioner does not require some precomputation before being applied. It is particularly relevant for preconditioners based on the (approximate) factorization of the matrix $A$. In this framework, the precisions $u_f$, $u_g$, $u_m$, and $u_a$ are fixed for every iteration of GMRES, unlike adaptive strategies where the precisions evolve as the iterations go, see the work by Gratton et al. [95], Loe et al. [150], and Oktay and Carson [171].

In this section, we develop a rounding error analysis of the left MGS-GMRES in mixed precision. We first demonstrate the backward stability of this algorithm and introduce two essential quantities $\rho_A$ and $\rho_M$ in section 7.2.1. By studying these quantities, we explain why it makes sense to differentiate the precision $u_a$ and $u_m$ in two independent precisions in section 7.2.2. Finally, we assess our theoretical analysis by numerical experiments in section 7.2.3.

### 7.2.1 Backward stability of left MGS-GMRES in mixed precision

To demonstrate the backward stability of left MGS-GMRES in mixed precision, we can use Theorem 5.2 for the backward stability of MGS-GMRES with an arbitrary matrix–vector product. To use this theorem, we need a bound on the error generated by the preconditioned matrix–vector product kernel (steps 7 to 8) of Algorithm 7.1.

We suppose in the following that the left-preconditioned matrix–vector product kernel $M^{-1}Av_j$ is composed of a standard matrix–vector product with the matrix $A$ in precision $u_a$, and the application of the preconditioner $M^{-1}$ on the resulting vector in precision $u_m$. We have

$$\mathrm{fl}(M^{-1}Av_j) = (M^{-1} + \Delta M_j)(A + \Delta A_j)v_j, \tag{7.1a}$$
$$|\Delta A_j| \leq \gamma_n^a |A|, \quad |\Delta M_j| \leq f(n)u_m E_m, \tag{7.1b}$$

where $\Delta M_j$ models a generic error on the application of the preconditioner bounded by the precision $u_m$ and a matrix $E_m$ with positive entries. Therefore, noting $\widetilde{A} = M^{-1}A$, the error $f$ on the kernel satisfies

$$\mathrm{fl}(M^{-1}Av_j) \approx M^{-1}Av_j + f, \qquad f = M^{-1}\Delta A_j v_j + \Delta M_j A v_j, \tag{7.2a}$$
$$\|f\|_2 \leq (u_a\rho_A + u_m\rho_M)\|\widetilde{A}\|_F \|v_j\|_2, \tag{7.2b}$$

with

$$\rho_A = \max_j(\rho_{A,j}) \quad \text{and} \quad \rho_M = \max_j(\rho_{M,j}) \tag{7.3}$$

where

$$u_a\rho_{A,j} \equiv \frac{\|M^{-1}\Delta A_j v_j\|_2}{\|\widetilde{A}\|_F\|v_j\|_2} \quad \text{and} \quad u_m\rho_{M,j} \equiv \frac{\|\Delta M_j A v_j\|_2}{\|\widetilde{A}\|_F\|v_j\|_2}. \tag{7.4}$$

Note that we implicitly supposed that the term $\Delta M_j \Delta A_j v_j$ is a second order term. We will make such implicit assumptions on second order terms at different moment of the analysis of this chapter while acknowledging that it would require a more careful check.

From (7.2) we can apply Theorem 5.2 and straightforwardly derive a backward stability result for left MGS-GMRES in mixed precision.

**Theorem 7.1.** *Let (1.1) be solved by a left preconditioned MGS-GMRES using a preconditioner $M$ and applying its operations in precision $u_g$, except the matrix–vector product with $A$ which is applied in precision $u_a$ and the application of the preconditioner $M$ which is applied in precision $u_m$. Provided that the application of $A$ and $M^{-1}$ to a vector $v$ satisfies*

$$\mathrm{fl}(Av) = (A + \Delta A)v, \quad |\Delta A| \leq \gamma_n^a |A|, \tag{7.5a}$$
$$\mathrm{fl}(M^{-1}v) = (M^{-1} + \Delta M)v, \quad |\Delta M| \leq f(n)u_m E_m, \tag{7.5b}$$

*where $E_m$ is a matrix with positive entries, and that*

$$\sigma_{\min}(\widetilde{A}) \gg \max(u_a\rho_A, u_m\rho_M, u_g)\|\widetilde{A}\|_F, \tag{7.6}$$

*then there is a step $k \leq n$ such that the algorithm produces a computed $\widehat{x}_k$ of the left-preconditioned system $\widetilde{A}x = \widetilde{b}$, with $\widetilde{A} = M^{-1}A$ and $\widetilde{b} = M^{-1}b$, satisfying*

$$(\widetilde{A} + \Delta\widetilde{A})\widehat{x}_k = \widetilde{b} + \Delta\widetilde{b}, \tag{7.7a}$$

$$\|\Delta\widetilde{A}\|_F \lesssim f(n,k)(u_a\rho_A + u_m\rho_M + u_g)\|\widetilde{A}\|_F, \tag{7.7b}$$

$$\|\Delta\widetilde{b}\|_2 \lesssim \widetilde{\gamma}^g_{kn}\|\widetilde{b}\|_2, \tag{7.7c}$$

*with*

$$u_a\rho_A = \max_j \frac{\|M^{-1}\Delta A_j v_j\|_2}{\|\widetilde{A}\|_F\|v_j\|_2} \quad \textit{and} \quad u_m\rho_M = \max_j \frac{\|\Delta M_j A v_j\|_2}{\|\widetilde{A}\|_F\|v_j\|_2}. \tag{7.8}$$

The quantities $\rho_A$ and $\rho_M$ of Theorem 7.1 are made of the terms $A$, $\Delta A_j$, $M$ and $\Delta M_j$, and can be further determined when more information is available on the preconditioner and how it is applied.

We now check that we can recover the result of Paige et al. [175] for an unpreconditioned MGS-GMRES in uniform precision. We choose $M = I$ and $u_g = u_m = u_a$, and evaluate $u_a\rho_A$ and $u_m\rho_M$. For $u_a\rho_A$, we have

$$u_a\rho_A = \max_j \frac{\|M^{-1}\Delta A_j v_j\|_2}{\|\widetilde{A}\|_F\|v_j\|_2} \leq \frac{\|\Delta A_j\|_2}{\|A\|_2} \leq \gamma^g_n. \tag{7.9}$$

As the application of the identity $I$ does not generate any computing error, $\Delta M_j = 0$ for all $j \leq n$ and, so, $u_m\rho_M = 0$. Hence, from (7.7) we recover the original result

$$(A + \Delta A_j)\widehat{x}_k = b + \Delta b, \quad \|\Delta A_j\|_F \lesssim \widetilde{\gamma}^g_{kn}\|A\|_F, \quad \|\Delta b\|_2 \lesssim \widetilde{\gamma}^g_{kn}\|b\|_2. \tag{7.10}$$

Note that Theorem 7.1 is valid regardless of the stopping criterion chosen as long as the algorithm does not stop before the specific $k$th iteration (mentioned in the theorem). However, in practice, it is important to note that for left-preconditioned GMRES we cannot have costless access to the residual of the original system (1.1). Instead, we have access to the residual of the left-preconditioned system (2.41), which can be arbitrarily smaller and lead to a too-early stop of the algorithm if the stopping criterion is based on it.

### 7.2.2 Differentiating the precisions $\mathbf{u_a}$ and $\mathbf{u_m}$

An important conclusion of Theorem 7.1 is that the interest of setting $u_a \neq u_m$ depends on the dominance of the quantity $\rho_A$ or $\rho_M$ over the other. Indeed, if $\rho_A \approx \rho_M$ we should set $u_a = u_m$, as for LU-GMRES-IR5 analyzed and used in the previous chapters. However, if for example $\rho_A \gg \rho_M$ we can set $u_m \gg u_a$, which, as long as $u_a\rho_A \geq u_m\rho_M$, is not expected to affect much the convergence. This section is dedicated to studying these two quantities.

*7.2.2.1 On the quantity $\rho_{A,j}$.* From (7.4) we have for all $j \leq n$

$$u_a\rho_{A,j} = \frac{\|M^{-1}\Delta A_j v_j\|_2}{\|\widetilde{A}\|_F\|v_j\|_2} \leq \widetilde{\gamma}^a_n \frac{\|M^{-1}\|_F\|A\|_F}{\|\widetilde{A}\|_F}, \tag{7.11}$$

then

$$\rho_{A,j} \leq f(n)\frac{\|M^{-1}\|_F\|M\widetilde{A}\|_F}{\|\widetilde{A}\|_F} \leq f(n)\kappa_F(M) \tag{7.12}$$

or

$$\rho_{A,j} \leq f(n)\frac{\|M^{-1}AA^{-1}\|_F\|A\|_F}{\|\widetilde{A}\|_F} \leq f(n)\kappa_F(A), \tag{7.13}$$

and so

$$\rho_{A,j} \leq f(n)\min(\kappa_F(A),\kappa_F(M)). \tag{7.14}$$

With the reasonable assumption that the condition number of the preconditioner is smaller than the condition number of $A$, that is $\kappa(M) \leq \kappa(A)$, then, for all $j \leq n$ we have $\rho_{A,j} \leq f(n)\kappa_F(M)$. Therefore, $\rho_{A,j}$ is at most of order $\kappa_F(M)$. In addition, if this bound is descriptive, we can expect $\rho_{A,j}$ to potentially increase as the condition number of the preconditioner increases.

We now make the simple observation that more the preconditioner reduce the condition number of the preconditioned matrix, more its condition number is close to the one of $A$. We have

$$\kappa(A) = \kappa(MM^{-1}A) \leq \kappa(M)\kappa(\widetilde{A}), \tag{7.15a}$$

$$\kappa(M) = \kappa(AA^{-1}M) \leq \kappa(A)\kappa(\widetilde{A}), \tag{7.15b}$$

which gives

$$\frac{\kappa(A)}{\kappa(\widetilde{A})} \leq \kappa(M) \leq \kappa(A)\kappa(\widetilde{A}). \tag{7.16}$$

For simplicity, we will consider that a "good" preconditioner $M$ is a preconditioner that brings $\kappa(\widetilde{A})$ close to 1. While this is not true in general since every nonincreasing GMRES convergence curve is possible for a given spectrum of $\widetilde{A}$ (see Greenbaum et al. [96]), it is, however, a tendency that is often observed in practice. Therefore, accounting for the previous comment on $\rho_{A,j}$, if $\kappa(A)$ is large and the preconditioner is "good", $\rho_{A,j}$ is expected to be large.

For the following, we denote the bound (7.11) on $\rho_{A,j}$ by

$$\overline{\rho}_{A,j} \equiv f(n)\frac{\|M^{-1}\|_F\|A\|_F}{\|\widetilde{A}\|_F}. \tag{7.17}$$

*7.2.2.2 Explicit construction of $M^{-1}$.* The application of a preconditioner $M$ can take different forms. In this section, we suppose that $M^{-1}$ is computed and formed explicitly, so that the application of $M^{-1}$ to the vectors $Av_j$ is a standard matrix–vector product. In this case $E_m \equiv |M^{-1}|$ in (7.1).

As for the previous bound on $\rho_{A,j}$, we denote the bound on $\rho_{M,j}$ by

$$\overline{\rho}_{M,j} \equiv f(n)\frac{\|M^{-1}\|_F\|Av_j\|_2}{\|\widetilde{A}\|_F\|v_j\|_F}. \tag{7.18}$$

In this case, we have $\overline{\rho}_{M,j} \leq \overline{\rho}_{A,j}$ for all $j \leq n$, where the gap between $\overline{\rho}_{A,j}$ and $\overline{\rho}_{M,j}$ can be quantified by the ratio $\overline{\rho}_{A,j}/\overline{\rho}_{M,j} = f(n)\|A\|_F\|v_j\|_2\|Av_j\|_2$. This ratio can be potentially large in the presence of heavy cancellations in the product $Av_j$, that is, $\|Av_j\|_2 \ll \|A\|_F\|v_j\|_2$.

If we assume that $\overline{\rho}_{A,j}$ and $\overline{\rho}_{M,j}$ are sharp bounds of, respectively, $\rho_{A,j}$ and $\rho_{M,j}$, then, our conclusions on $\overline{\rho}_{A,j}$ and $\overline{\rho}_{M,j}$ apply to $\rho_{A,j}$ and $\rho_{M,j}$. This assumption seems to be reasonable if we view the error matrices $\Delta A_j$ and $\Delta M_j$ as random perturbations that do not carry significant cancellations (i.e., $|\Delta A_j v_j| \approx |\Delta A_j||v_j|$). In particular, we can expect $\rho_{M,j} \leq \rho_{A,j}$.

*7.2.2.3 Implicit construction of $M^{-1}$ via LU factorization.* In this section, we suppose that $M^{-1}$ is applied by an implicit LU inversion; that is, we have computed or we have access to the LU factors of M, which are noted $L$ and $U$, and we apply $M^{-1}$ by applying the two consecutive triangular solves with $L$ and $U$ in precision $u_m$. For instance, this kind of preconditioner application can be used with block Jacobi or ILU preconditioners. Thus, the preconditioned matrix–vector product kernel verifies

$$\mathrm{fl}(M^{-1}Av_j) = (U+\Delta U)^{-1}(L+\Delta L)^{-1}(A+\Delta A_j)v_j, \tag{7.19a}$$
$$|\Delta U| \leq \gamma_n^m |U|, \quad |\Delta L| \leq \gamma_n^m |L|, \tag{7.19b}$$

where we identify

$$\Delta M_j = (U+\Delta U)^{-1}(L+\Delta L)^{-1} - M^{-1} \tag{7.20a}$$
$$\approx (U^{-1}+U^{-1}\Delta U U^{-1})(L^{-1}+L^{-1}\Delta L L^{-1}) - M^{-1} \tag{7.20b}$$
$$\approx -U^{-1}\Delta U M^{-1} - M^{-1}\Delta L L^{-1} \tag{7.20c}$$

by using the formulation of the inverse of a sum of matrices by Henderson and Searle [104]. Next, to bound $\rho_{M,j}$, we form

$$\Delta M_j Av_j \approx -U^{-1}\Delta U_j M^{-1}Av_j - M^{-1}\Delta L_j L^{-1}Av_j \tag{7.21}$$

which can be bounded such that

$$\|\Delta M_j Av_j\|_2 \lesssim \gamma_n^m \left[ \| |U^{-1}||U||M^{-1}Av_j| \|_2 + \| |M^{-1}||L||L^{-1}Av_j| \|_2 \right] \tag{7.22a}$$
$$\leq \gamma_n^m \big[ \mathrm{cond}(U)\|M^{-1}Av_j\|_2 \tag{7.22b}$$
$$+ \mathrm{cond}(L)\min(\|U^{-1}\|_F\|L^{-1}Av_j\|_2, \|M^{-1}\|_F\|Av_j\|_2) \big]. \tag{7.22c}$$

We know from [113, lem 8.6] that $\mathrm{cond}(L)$ and $\mathrm{cond}(U)$ are of modest size when partial pivoting is used, in addition $\|M^{-1}Av_j\|_2 \leq \min(\|U^{-1}\|_F\|L^{-1}Av_j\|_2, \|M^{-1}\|_F\|Av_j\|_2)$, so if we gather all the constants under the term $f(n)$ we obtain

$$\rho_{M,j} \lesssim \overline{\rho}_{M,j} \equiv f(n)\frac{\min(\|U^{-1}\|_F\|L^{-1}Av_j\|_2, \|M^{-1}\|_F\|Av_j\|_2)}{\|\widetilde{A}\|_F\|v_j\|_2}. \tag{7.23}$$

The ratio becomes, for all $j \leq n$,

$$\overline{\rho}_{A,j}/\overline{\rho}_{M,j} = f(n)\frac{\|M^{-1}\|_F\|A\|_F\|v_j\|_2}{\min(\|U^{-1}\|_F\|L^{-1}Av_j\|_2, \|M^{-1}\|_F\|Av_j\|_2)}. \tag{7.24}$$

Therefore, if there are heavy cancellations in the products $Av_j$ or $L^{-1}Av_j$, that is,

$$\|M^{-1}\|_F\|Av_j\|_2 \ll \|M^{-1}\|_F\|A\|_F\|v_j\|_2 \text{ or } \|U^{-1}\|_F\|L^{-1}Av_j\|_2 \ll \|M^{-1}\|_F\|A\|_F\|v_j\|_2, \tag{7.25}$$

the ratio can be large.

Supposing as previously that the products with $\Delta L_j$ and $\Delta U_j$ do not carry significant cancellations, we can expect the bound (7.23) to be sharp. It tends to assess that it is very likely to observe $\rho_{M,j} \leq \rho_{A,j}$, where possibly $\rho_{M,j} \ll \rho_{A,j}$ if the ratio is large. It is the same conclusion as the previous section 7.2.2.2 about the explicit construction of $M^{-1}$.

*7.2.2.4 Discussion on $\rho_{A,j}$ and $\rho_{M,j}$.* We emphasize the two significant observations of the previous analysis. First, $\rho_A$ is potentially large in cases where $A$ is ill-conditioned and where the preconditioner reduces $\kappa(\widetilde{A})$ close to 1, that is, when $\kappa(M)$ is large too. In such cases, it is interesting to use high precision for $u_a$ to compensate for the generated error. Second, we have shown that the ratio between the bounds $\overline{\rho}_{A,j}$ and $\overline{\rho}_{M,j}$ can be potentially large, specifically when there are heavy cancellations in the products $Av_j$ or in the triangular solve $L^{-1}Av_j$. Therefore, assuming these bounds are reasonably descriptive of the actual behavior of the effective quantities $\rho_{A,j}$ and $\rho_{M,j}$, we may expect the ratio $\rho_{A,j}/\rho_{M,j}$ to also be potentially large. In this case, $\rho_M \ll \rho_A$ and it becomes meaningful (as defined in section 2.1.3.3) to set $u_m \gg u_a$. Note that we have not shown that these bounds are descriptive, so this last conclusion constitutes more an intuition than a strong assessment. Conversely, as we showed that it is very unlikely to have $\rho_A \leq \rho_M$, we know that it is not meaningful to set $u_m \ll u_a$. We now turn to numerical experiments to assess to what extent these theoretical conclusions are met in practice.

### 7.2.3 Numerical experiments

We are interested in validating through numerical experiments the possibility to set $u_m \gg u_a$, which has been the main conclusion of the previous section. We use a generator to create dense matrices $A$ and their associated preconditioners $M$, where both condition numbers $\kappa(A)$ and $\kappa(M)$ can be manually set. We emphasize that this generator is not meant to be of any practical relevance but is rather used to fully control the conditionings of $A$ and $M$. We combine it with a Julia implementation of the left-preconditioned MGS-GMRES to compare the quantities $\rho_{A,j}$ and $\rho_{M,j}$ over the first iterations of GMRES.

*7.2.3.1 Preconditioner generator.* For the numerical validation of our rounding error analysis, we use the following random generator of preconditioned linear systems $M^{-1}Ax = M^{-1}b$. Given a logarithmic distribution of the eigenvalues $\{1 = \lambda_1 > \lambda_2 > \cdots > \lambda_n > 0\}$, we

build $A$ with a target condition number $\kappa(A)$ such that

$$A = Q^T \operatorname{diag}(\lambda_1, \lambda_2, \ldots, \lambda_n) Q, \tag{7.26}$$

where $Q \in \mathbb{R}^{n\times n}$ is a randomly generated orthogonal matrix. Doing so, we have $\kappa(A) = 1/\lambda_n$. Then we build its preconditioner $M$ with a target condition number $\kappa(M) \leq \kappa(A)$ by truncating the previous distribution of eigenvalues at the first $i$ such that $1/\lambda_i > \kappa(M)$, it provides

$$M = Q^T \operatorname{diag}(\lambda_1, \ldots, \lambda_{i-1}, \ldots, \lambda_{i-1}) Q. \tag{7.27}$$

In exact arithmetic, this setup guarantees that

$$\kappa(\widetilde{A}) = \kappa(M^{-1}A) = \frac{\lambda_{i-1}}{\lambda_n} \approx \frac{\kappa(A)}{\kappa(M)}. \tag{7.28}$$

Therefore, with this generator a preconditioner $M$ reducing $\kappa(\widetilde{A})$ close to 1 satisfies $\kappa(M) \approx \kappa(A)$ and, conversely, a preconditioner providing $\kappa(\widetilde{A}) \gg 1$ satisfies $\kappa(M) \ll \kappa(A)$. We assume that a "good" preconditioner reduces $\kappa(\widetilde{A})$, even though it is not necessary true as explained in section 7.2.2.1. With this assumption, $M$ becomes a better preconditioner as $\kappa(M)$ gets closer to $\kappa(A)$. In our experiments, both $A$ and $M$ are generated in double precision and then converted to the arithmetic required by the algorithm.

*7.2.3.2 Comparison of $\rho_{A,j}$ and $\rho_{M,j}$.* We plot in Figures 7.1 and 7.2 the evolution of $\rho_{A,j}$ and $\rho_{M,j}$ over the first 15 iterations of MGS-GMRES with, respectively, preconditioners $M$ with increasing $\kappa(M)$ and $A$ with fixed $\kappa(A) = 10^{10}$, and preconditioners $M$ with fixed $\kappa(M) = 10^2$ and $A$ with increasing $\kappa(A)$. To generate the different matrices $A$ and $M$ of dimension $50\times 50$, we use the previous random dense generator presented in section 7.2.3.1. For these experiments, we are not interested in the accuracy of the solution of the GMRES throughout the iterations, but we only focus on the values of $\rho_{A,j}$ and $\rho_{M,j}$. We are also not interested in studying different combinations of $u_g$, $u_m$, and $u_a$; these precisions are all fixed to double precision, which is the precision at which $A$ and $M$ are generated.

We now comment on Figure 7.1. In this figure, the quality of the preconditioner $M$ increases with $\kappa(M)$ until it reaches $\kappa(M) = \kappa(A)$. For $\kappa(M) = 10^2$, we can observe that, over the iterations, $\rho_{A,j}$ is just slightly higher than $\rho_{M,j}$ (the ratio $\rho_{A,j}/\rho_{M,j}$ is about $10^1$). In this case, we are not in a configuration where $\rho_{A,j} \gg \rho_{M,j}$, and so where we can set $u_m \gg u_a$. However, we remark that when we increase $\kappa(M)$, this ratio is increasing; that is, the gap between $\rho_{A,j}$ and $\rho_{M,j}$ is widening. For example, for $\kappa(M) = 10^4$ the ratio is approximately $10^2$, for $\kappa(M) = 10^6$ it is $10^4$, for $\kappa(M) = 10^8$ it is $10^6$, and for $\kappa(M) = 10^{10}$ it is $10^8$. Therefore, when $\kappa(M)$ gets larger, we have more room to set $u_m \gg u_a$. This experimental observation validates our theoretical conclusion of section 7.2.2.4 stating that $\rho_{A,j}$ tends to increase with $\kappa(M)$, and that $\rho_{M,j}$ can be far smaller than $\rho_{A,j}$. More precisely, with these experiments, we observe that the ratio $\rho_{A,j}/\rho_{M,j}$ is getting larger for high $\kappa(A)$ and small $\kappa(\widetilde{A})$, that is, when $\kappa(M)$ is high. This configuration is favorable for setting $u_m \ll u_a$, and it is likely that this observation stays true for other, more practical, classes of preconditioners.

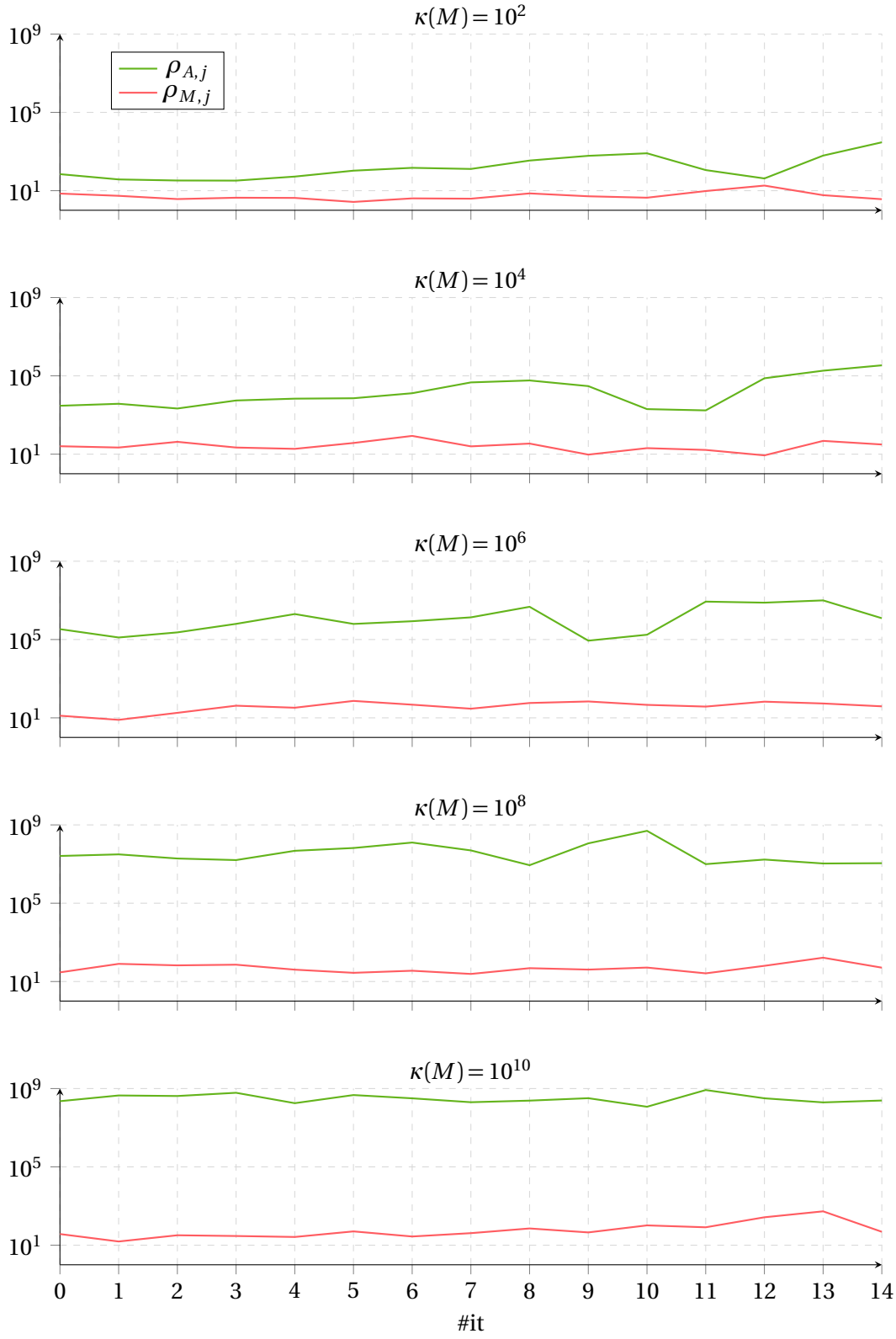



Figure 7.1: Evolution of $\rho_A$ and $\rho_M$ over the 15 first iterations of GMRES for preconditioners $M$ of a matrix $A \in \mathbb{R}^{50\times 50}$ with various condition numbers. We fix $\kappa(A) = 10^{10}$.

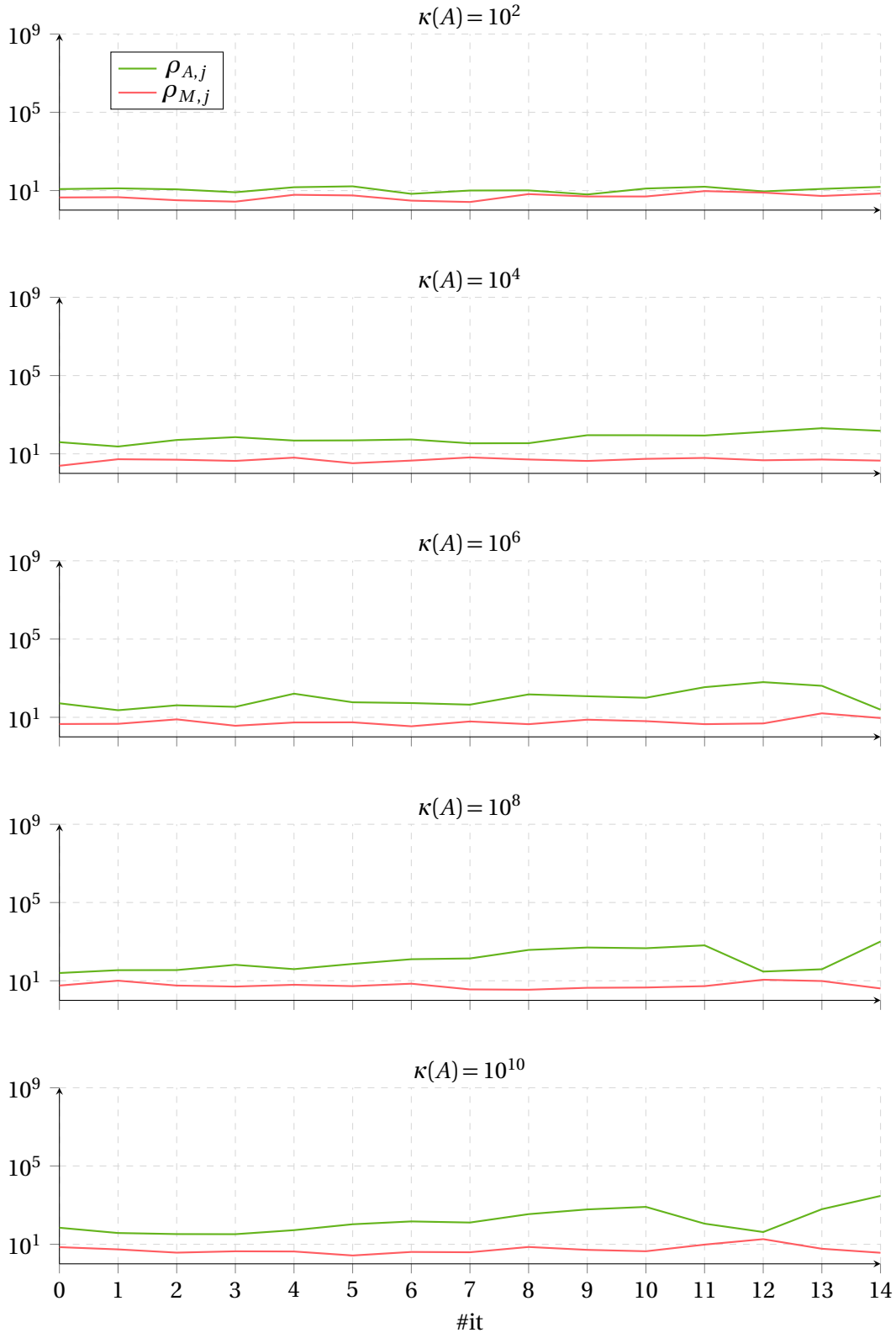



Figure 7.2: Evolution of $\rho_A$ and $\rho_M$ over the 15 first iterations of GMRES for matrices $A \in \mathbb{R}^{50\times 50}$ with various condition numbers. We fix $\kappa(M) = 10^2$.

Next, we comment on Figure 7.2. On this figure, $\kappa(M)$ is fixed and $\kappa(A)$ increases. Interestingly, from $\kappa(A) = 10^2$ to $\kappa(A) = 10^{10}$, the ratio between $\rho_{A,j}$ and $\rho_{M,j}$ seems to stay constant at about $10^1$, and does not evolve when $\kappa(A)$ is increasing. This observation also comes to strengthen our conclusions of section 7.2.2.4. Specifically, it confirms that the quantities $u_a\rho_A$ and $u_m\rho_M$ of (7.7) are driven by $\kappa(M)$ and not $\kappa(A)$. It is an interesting property because when $\kappa(M)$ is small, both $\rho_{A,j}$ and $\rho_{M,j}$ are small, no matter the value of $\kappa(A)$. In such cases, having $u_m$ and $u_a$ in higher precision than $u_g$ is meaningless, and we should set $u_m = u_a = u_g$.

## 7.3 M-GMRES-IR6

Algorithm 7.2 describes an iterative refinement variant based on the preconditioned MGS-GMRES in mixed precision studied in section 7.2, and applying its operations in six independent precisions $u_f$, $u$, $u_r$, $u_g$, $u_m$, and $u_a$ that we call *left or right GMRES-based iterative refinement with arbitrary preconditioner $M$ in six precisions (M-GMRES-IR6)*. The precision $u$ and $u_r$ are the precisions for, respectively, the computation of the update and the residual; these operations are equivalent to step 3 and step 5 of the generalized iterative refinement described by Algorithm 4.1. The last four precisions $u_f$, $u_g$, $u_m$, and $u_a$ are the precisions used by the inner preconditioned MGS-GMRES in mixed precision studied previously, and are the same as in Algorithm 7.1. Note that in Algorithm 7.2, the iterative refinement process is written in the form of a restarting process; we explain the equivalence between the two in more detail in the coming section 7.3.1.

Written as such, left and right M-GMRES-IR6 cover almost all the algorithms proposed by the papers listed in Table 7.1; the only exceptions are the strategies where the precision varies adaptively during the execution. However, even for these strategies, we believe that M-GMRES-IR6 can be easily extended to be used with them. In addition, it can achieve new meaningful combinations of precisions that have never been covered before, particularly variants where $u_g \geq u_m \gg u_a$. These new kinds of variants, which apply the preconditioner with lower precision than the matrix $A$, can be attractive in terms of performance if the application of the matrix $A$ is negligible compared with the application of the preconditioner. This configuration is actually relatively standard for sparse systems. For example, solving a sparse problem with an ILU preconditioner can lead to computing approximate factors with a higher number of nonzeros than $A$ because of the fill-in (see section 2.2.3.2); the solves are therefore expected to be more costly than the matrix–vector products with $A$.

There are two major differences between M-GMRES-IR6 and LU-GMRES-IR5 previously described and studied in chapter 5. First, the preconditioner is not necessarily an LU preconditioner but is rather arbitrary, such that we can derive results for a wide class of preconditioners rather than for one specialized preconditioner. Second, we split the application of the preconditioned matrix–vector product in two precisions $u_a$ and $u_m$, where formerly it was fully applied in precision $u_p$.

This section is interested in studying left M-GMRES-IR6 and, specifically, in deriving convergence conditions on this algorithm's forward and backward errors by using the previous error analysis on left MGS-GMRES in mixed precision. To do so, we first explain in

**Algorithm 7.2** M-GMRES-IR6

**Input:** an $n \times n$ matrix $A$, a preconditioner $M$, and right-hand side $b$.
**Output:** a computed solution to $Ax = b$.

| | | | | | |
|---|---|---|---|---|---|
| 1: | Compute $M$ (Optional) | $(u_f)$ | 1: | Compute $M$ (Optional) | $(u_f)$ |
| 2: | Initialize $x_0$ | | 2: | Initialize $x_0$ | |
| 3: | **repeat** | | 3: | **repeat** | |
| 4: | $r_i = Ax_i - b$ | $(u_r)$ | 4: | $r_i = Ax_i - b$ | $(u_r)$ |
| 5: | $\boldsymbol{s_i = M^{-1} r_i}$ | $(u_m)$ | 5: | | |
| 6: | $\beta = \|s_i\|, \quad v_1 = s_i/\beta, \quad$ j=0 | $(u_g)$ | 6: | $\beta = \|r_i\|, \quad v_1 = r_i/\beta, \quad$ j=0 | $(u_g)$ |
| 7: | **repeat** | | 7: | **repeat** | |
| 8: | $j = j + 1$ | | 8: | $j = j + 1$ | |
| 9: | $z_j = Av_j$ | $(u_a)$ | 9: | $\boldsymbol{z_j = M^{-1} v_j}$ | $(u_m)$ |
| 10: | $\boldsymbol{w_j = M^{-1} z_j}$ | $(u_m)$ | 10: | $w_j = Az_j$ | $(u_a)$ |
| 11: | **for** $l = 1 : j$ **do** | | 11: | **for** $l = 1 : j$ **do** | |
| 12: | $h_{l,j} = v_l^T w_j$ | $(u_g)$ | 12: | $h_{l,j} = v_l^T w_j$ | $(u_g)$ |
| 13: | $w_j = w_j - h_{l,j} v_l$ | $(u_g)$ | 13: | $w_j = w_j - h_{l,j} v_l$ | $(u_g)$ |
| 14: | **end for** | | 14: | **end for** | |
| 15: | $h_{j+1,j} = \|w_j\|$ | $(u_g)$ | 15: | $h_{j+1,j} = \|w_j\|$ | $(u_g)$ |
| 16: | $v_{j+1} = w_j / h_{j+1,j}$ | $(u_g)$ | 16: | $v_{j+1} = w_j / h_{j+1,j}$ | $(u_g)$ |
| 17: | $y_j = \arg\min_y \|\beta e_1 - H_j y\|$ | $(u_g)$ | 17: | $y_j = \arg\min_y \|\beta e_1 - H_j y\|$ | $(u_g)$ |
| 18: | **until** $\|\beta e_1 - H_j y_j\| \leq \tau_g$ | | 18: | **until** $\|\beta e_1 - H_j y_j\| \leq \tau_g$ | |
| 19: | $x_{i+1} = x_i + V_j y_j$ | $(u)$ | 19: | $\boldsymbol{x_{i+1} = x_i + Z_j y_j}$ | $(u)$ |
| 20: | $i = i + 1$ | | 20: | $i = i + 1$ | |
| 21: | **until** convergence | | 21: | **until** convergence | |

section 7.3.1 why restarted GMRES is a form of iterative refinement. It allows us to use the generalized iterative refinement results of section 4.2 to carry out the rounding error analysis of left M-GMRES-IR6 in section 7.3.2. Finally, we validate our theoretical result with randomly generated and real-life matrices, and evaluate the potential of new combinations of precisions in section 7.3.3.

### 7.3.1 Equivalence between restarted GMRES and iterative refinement

M-GMRES-IR6, as described by Algorithm 7.2, is a form of restarted GMRES. However, it can also be interpreted as an iterative refinement algorithm based on the preconditioned MGS-GMRES in mixed precision covered in section 7.2. This statement is essential for the following error analysis since it means that major stability results of iterative refinement can apply to M-GMRES-IR6, particularly Theorems 4.1 and 4.2.

We now explain in which case the equivalence is true. The difference between a restarted GMRES and iterative refinement combined with a non-restarted GMRES is that, in the first case, the GMRES solver solves $Ax = b$, and in the second, it solves the correction equation $Ad_i = r_i$. Presented differently, the restarted GMRES builds the next iterate of the solution $x_{i+1}$ in the subspaces $x_i + K_j(A, b - Ax_i)$ with $j \leq n$ and where the notation $K_j(A, b - Ax_i)$ denotes a Krylov subspace (see section 2.3.1). The GMRES-IR builds the next correction

$d_i$ in the subspaces $d_{i,0} + K_j(A,(b-Ax_i)-Ad_{i,0})$, where $d_{i,0}$ is a first guess on $d_i$, and so the next iterate $x_{i+1}$ is in $x_i + d_{i,0} + K_j(A,(b-Ax_i)-Ad_{i,0})$. Therefore, restarted GMRES is equivalent to a GMRES-IR where the first guess of the solution $d_{i,0}$ is always initialized to the zero vector.

Hence, M-GMRES-IR6 can be viewed as both, a restarted GMRES using six precisions as described in Algorithm 7.2, or as an iterative refinement based on the preconditioned MGS-GMRES in mixed precision represented by Algorithm 7.1.

As we have explained in chapter 5, M-GMRES-IR6 is not backward stable if GMRES restarts after a fixed maximum number of iterations, as it is often done in practice. This is why we do not assume such a restarting criterion in the following, and GMRES is allowed to iterate on the full dimension of the problem. We discussed in section 5.4 a stopping criterion $\tau_g$ for LU-GMRES-IR5 on which we can derive a backward stability result. It is easily extendable to M-GMRES-IR6; though, we do not consider it in the following analysis but use it in our numerical experiments.

### 7.3.2 Error analysis and convergence conditions

In the following, we carry out the error analysis of M-GMRES-IR6. The structure is similar to the analysis of LU-GMRES-IR5 in section 5.2, our aim is to rewrite (4.9) and (4.16) to apply Theorems 4.1 and 4.2 to obtain specialized conditions for the convergence of the forward and backward errors of M-GMRES-IR6 to their respective limiting accuracies (4.10) and (4.17). To achieve that, we proceed like in section 5.2.2: we bound the error in $\widehat{s}_i = \mathrm{fl}(M^{-1}\widehat{r}_i)$, then we use our analysis of left MGS-GMRES in mixed precision made in section 7.2 to derive bounds on the forward and backward errors of the solution of $\widetilde{A}d_i = s_i$, finally we identify the terms $u_s$, $E_i$, $c_1$ and $c_2$ from (4.9) and (4.16) and we conclude on the convergence conditions.

We first evaluate the error on computing the preconditioned right-hand side $s_i = M^{-1}\widehat{r}_i$. Using the assumption (7.1) we have

$$\widehat{s}_i = \mathrm{fl}(M^{-1}\widehat{r}_i) = (M^{-1} + \Delta M_s)\widehat{r}_i, \quad |\Delta M_s| \leq f(n)u_m E_m, \tag{7.29}$$

giving

$$\|s_i - \widehat{s}_i\|_\infty = \|\Delta M_s \widehat{r}_i\|_\infty \leq \|\Delta M_s M\|_\infty \|s_i\|_\infty. \tag{7.30}$$

We know from Theorem 7.1 that the computed solution $\widehat{d}_i$ of $\widetilde{A}d_i = \widehat{s}_i$ by MGS-GMRES in three precisions satisfies

$$(\widetilde{A} + \Delta\widetilde{A})\widehat{d}_i = \widehat{s}_i + \Delta s_i, \tag{7.31a}$$

$$\|\Delta\widetilde{A}\|_F \lesssim f(n,k)(u_a\rho_A + u_m\rho_M + u_g)\|\widetilde{A}\|_F, \tag{7.31b}$$

$$\|\Delta s_i\|_2 \lesssim \widetilde{\gamma}^g_{kn}\|\widehat{s}_i\|_2 \lesssim n^{1/2}\widetilde{\gamma}^g_{kn}\|s_i\|_\infty. \tag{7.31c}$$

Rewriting (7.31a) as

$$s_i - \widetilde{A}\widehat{d}_i = \Delta\widetilde{A}\widehat{d}_i - (\widehat{s}_i - s_i) - \Delta\widehat{s}_i \tag{7.32}$$

and using (7.31b), (7.30), and (7.31c) to bound the three terms on the right-hand side, we

obtain

$$\|s_i - \widetilde{A}\widehat{d}_i\|_\infty \leq \|\Delta\widetilde{A}\|_\infty\|\widehat{d}_i\|_\infty + \|\widehat{s}_i - s_i\|_\infty + \|\Delta\widehat{s}_i\|_\infty \tag{7.33a}$$

$$\lesssim f(n,k)\big[(u_a\rho_A + u_m\rho_M + u_g)\|\widetilde{A}\|_\infty\|\widehat{d}_i\|_\infty \tag{7.33b}$$

$$+ \|\Delta M_s M\|_\infty\|s_i\|_\infty + u_g\|s_i\|_\infty\big] \tag{7.33c}$$

$$\leq f(n,k)(u_a\rho'_A + u_m\rho'_M + u_g)(\|\widetilde{A}\|_\infty\|\widehat{d}_i\|_\infty + \|s_i\|_\infty), \tag{7.33d}$$

where with (7.3), by switching from the 2-norm to the $\infty$-norm, and by taking into account the term $\|\Delta M_j M\|_\infty$, we have

$$u_a\rho'_A \equiv \max_j \frac{\|M^{-1}\Delta A_j v_j\|_\infty}{\|\widetilde{A}\|_\infty\|v_j\|_\infty} \quad \text{and} \quad u_m\rho'_M \equiv \max_j \frac{\|\Delta M_j A v_j\|_\infty}{\|\widetilde{A}\|_\infty\|v_j\|_\infty} + \|\Delta M_s M\|_\infty. \tag{7.34}$$

Therefore, the normwise relative backward error of the preconditioned system $\widetilde{A}\widehat{d}_i = s_i$ is bounded by

$$\frac{\|s_i - \widetilde{A}\widehat{d}_i\|_\infty}{\|\widetilde{A}\|_\infty\|\widehat{d}_i\|_\infty + \|s_i\|_\infty} \lesssim f(n,k)(u_a\rho'_A + u_m\rho'_M + u_g) \tag{7.35}$$

and the forward error of the computed $\widehat{d}_i$ therefore satisfies

$$\frac{\|d_i - \widehat{d}_i\|_\infty}{\|d_i\|_\infty} \lesssim f(n,k)(u_a\rho'_A + u_m\rho'_M + u_g)\kappa_\infty(\widetilde{A}). \tag{7.36}$$

By rewriting (7.32) as

$$\widehat{r}_i - A\widehat{d}_i = M(\Delta\widetilde{A}\widehat{d}_i - (\widehat{s}_i - s_i) - \Delta\widehat{s}_i), \tag{7.37}$$

and reusing the reasoning of (7.33a) we obtain

$$\|\widehat{r}_i - A\widehat{d}_i\|_\infty \lesssim f(n,k)(u_a\rho'_A + u_m\rho'_M + u_g)(\|M\|_\infty\|\widetilde{A}\|_\infty\|\widehat{d}_i\|_\infty + \|M\|_\infty\|s_i\|_\infty) \tag{7.38a}$$

$$\leq f(n,k)(u_a\rho'_A + u_m\rho'_M + u_g)(\frac{\|M\|_\infty\|\widetilde{A}\|_\infty}{\|A\|_\infty}\|A\|_\infty\|\widehat{d}_i\|_\infty \tag{7.38b}$$

$$+ \kappa_\infty(M)\|\widehat{r}_i\|_\infty) \tag{7.38c}$$

which gives a bound on the normwise backward error of the original system $A\widehat{d}_i = \widehat{r}_i$.

From (7.36) and (7.38) we can identify

$$u_s \equiv (u_a\rho'_A + u_m\rho'_M + u_g), \quad \|E_i\|_\infty \equiv f(n,k)\kappa_\infty(\widetilde{A}), \tag{7.39a}$$

$$c_1 \equiv f(n,k)\frac{\|M\|_\infty\|\widetilde{A}\|_\infty}{\|A\|_\infty}, \quad c_2 \equiv f(n,k)\kappa_\infty(M), \tag{7.39b}$$

and by using Theorems 4.1 and 4.2 we obtain the following theorem.

**Theorem 7.2** (Convergence of M-GMRES-IR6)**.** *Let* (1.1) *be solved by M-GMRES-IR6 (Algorithm 7.2). If $u_g \geq u$, and that the application of $A$ and $M^{-1}$ to a vector $v$ satisfies*

$$\mathrm{fl}(Av) = (A + \Delta A)v, \quad |\Delta A| \leq \gamma_n^a|A|, \tag{7.40a}$$

$$\mathrm{fl}(M^{-1}v) = (M^{-1} + \Delta M)v, \quad |\Delta M| \le f(n)u_m E_m, \tag{7.40b}$$

*where $E_m$ is a matrix with positive entries, then the forward and backward errors will reach their respective limiting accuracies*

$$p\,u_r\,\mathrm{cond}(A,x) + u \text{ (forward)} \qquad \text{and} \qquad p\,u_r + u \text{ (backward)}, \tag{7.41}$$

*provided that*

$$(u_a\rho_A' + u_m\rho_M' + u_g)\kappa_\infty(\widetilde{A}) \ll 1 \text{ (forward)} \tag{7.42}$$

*and*

$$(u_a\rho_A' + u_m\rho_M' + u_g)(\frac{\|M\|_\infty\|\widetilde{A}\|_\infty}{\|A\|_\infty}\kappa(A) + \kappa(M)) \ll 1 \text{ (backward)}, \tag{7.43}$$

*where*

$$u_a\rho_A' = \max_j \frac{\|M^{-1}\Delta A_j v_j\|_\infty}{\|\widetilde{A}\|_\infty\|v_j\|_\infty} \quad \text{and} \quad u_m\rho_M' = \max_j \frac{\|\Delta M_j A v_j\|_\infty}{\|\widetilde{A}\|_\infty\|v_j\|_\infty} + \|\Delta M_s M\|_\infty. \tag{7.44}$$

In order to assess the consistency of our new analysis, we demonstrate that we can recover the forward error convergence condition (5.2) of LU-GMRES-IR5 in Theorem 5.1 with the convergence condition (7.42) of M-GMRES-IR6. We suppose that $u_a = u_m = u_p$ and that $M = \widehat{L}\widehat{U}$ where $\widehat{L}$ and $\widehat{U}$ are the computed LU factors of $A$ in precision $u_f$. We first evaluate the term $\rho_A'$, we have

$$u_p\rho_A' \le \frac{\|M^{-1}\Delta A_j\|_\infty}{\|\widetilde{A}\|_\infty} = \frac{\|M^{-1}AA^{-1}\Delta A_j\|_\infty}{\|\widetilde{A}\|_\infty} \le \|A^{-1}\Delta A_j\|_\infty \le \gamma_n^p\kappa_\infty(A). \tag{7.45}$$

We now evaluate the term $\rho_M'$, replacing $\Delta M_j$ as in (7.20), using Theorem 2.2, and dropping second order terms give

$$\begin{aligned} u_p\rho_M' &\le \frac{\|\Delta M_j A\|_\infty}{\|\widetilde{A}\|_\infty} + \|\Delta M_s M\|_\infty \le \frac{\|\Delta M_j M M^{-1}A\|_\infty}{\|\widetilde{A}\|_\infty} + \|\Delta M_s M\|_\infty && (7.46)\\ &\le \|\Delta M_j M\|_\infty + \|\Delta M_s M\|_\infty \lesssim 2\|\widehat{U}^{-1}\Delta U\|_\infty + 2\|\widehat{U}^{-1}\widehat{L}^{-1}\Delta L\widehat{U}\|_\infty && (7.47)\\ &\lesssim f(n)u_p\kappa_\infty(A). && (7.48) \end{aligned}$$

Bounding $\kappa_\infty(\widetilde{A})$ in (7.42) with (5.35) provides the forward error convergence condition (5.2) of LU-GMRES-IR5.

Similarly, we show that we can recover the backward error convergence condition (5.3) of LU-GMRES-IR5 with (7.43). We have already worked on the terms $u_p\rho_M'$ and $u_p\rho_A'$, so we only need to express $\kappa_\infty(M)$ and $\|M\|_\infty\|\widetilde{A}\|_\infty/\|A\|_\infty$. First, using Theorem 2.2 and dropping second order terms provides

$$\kappa_\infty(M) = \|\widehat{L}\widehat{U}\|_\infty\|\widehat{U}^{-1}\widehat{L}^{-1}\|_\infty \approx \|A\|_\infty\|A^{-1}\|_\infty = \kappa(A), \tag{7.49}$$

and

$$\frac{\|M\|_\infty \|\widetilde{A}\|_\infty}{\|A\|_\infty} \approx \frac{\|A\|_\infty \|\widetilde{A}\|_\infty}{\|A\|_\infty} = \|\widetilde{A}\|_\infty. \tag{7.50}$$

Hence, we can recover the backward error convergence condition (5.38) of LU-GMRES-IR5 from (7.43).

We now comment on the new forward error convergence condition (7.42) of M-GMRES-IR6. The two significant differences with the previous forward error convergence condition (5.2) derived for LU-GMRES-IR5 in chapter 5 are:

- The term $u_p\kappa(A)$ becomes $u_a\rho'_A + u_m\rho'_M$ because, first, the preconditioner is applied in two independent precisions $u_m$ and $u_a$ instead of being fully applied in precision $u_p$ and, second, without information on the preconditioner $u_m$ and $u_a$ are multiplied by the quantities $\rho'_A$ and $\rho'_M$ instead of $\kappa(A)$.

- The term $1+\kappa(A)^2 u_f^2$ becomes $\kappa(\widetilde{A})$ because $\kappa(\widetilde{A})$ cannot be simplified without further information on the preconditioner.

The analysis of the meaningful combinations of LU-GMRES-IR5 in section 5.3 is naturally inherited to the six precisions $u_f$, $u$, $u_r$, $u_g$, $u_m$, and $u_a$. In particular, from condition (7.42) we require $u_a, u_m \leq u_g$. In addition, with the reasonable assumption made in section 7.2.2, we have $\rho_M \leq \rho_A$, so we require $u_m \geq u_a$. Note that the precision $u_f$ does not directly appear in the condition (7.42) unlike the condition (5.2) for LU-GMRES-IR5. This precision is hidden in the term $\kappa(\widetilde{A})$ because it only affects the quality of the preconditioned matrix and does not play a role in the rounding errors generated by the computation of the preconditioned matrix–vector product.

Applying Theorem 7.2 for $M = I$ provides a backward stability result for an unpreconditioned restarted GMRES. Revisiting bound (7.9) directly gives $u_a\rho_A \leq \widetilde{\gamma}_n^a$ and, as discussed in section 7.2.1, we have $u_m\rho_M = 0$. Replacing these quantities accordingly in the bounds (7.42) and (7.43) shows that having $u_a < u_g$ is not meaningful. Then, $u_a = u_g$, and the convergence conditions for both forward and backward errors become $u_g\kappa(A) \ll 1$. Therefore, as long as the matrix $A$ is nonsingular relative to the precision $u_g$, Theorem 7.2 ensures that the forward and backward errors of the solution will converge to their respective limiting accuracies $p\,u_r\,\mathrm{cond}(A,x) + u$ and $p\,u_r + u$.

Note that the analysis of this chapter, and also the previous analyses of chapters 5 and 6, do not consider a phenomenon that possibly has a noticeable effect on the convergence of our different algorithms: for a given matrix $A$, if we note $A_m = \mathrm{fl}_m(A)$ which is $A$ cast into a lower precision $u_m$, then possibly $\kappa(A_m) \neq \kappa(A)$. We call this phenomenon "regularization" and, in practice, we often observe that $\kappa(A_m)$ satisfies $\min(\kappa(A), u_m^{-1}) \leq \kappa(A_m) \leq \kappa(A)$. With M-GMRES-IR6, the matrices $A$ and $M$ are applied respectively in precision $u_a$ and $u_m$, therefore, if the original precisions of these matrices are higher, the condition numbers of the resulting cast matrices can be lowered and it would affect the convergence conditions (7.42) and (7.43).

### 7.3.3 Numerical experiments

In the following numerical experiments, we are aiming at both validating the convergence condition on the forward error (7.42) of Theorem 7.2 and evaluating the potential of applying the preconditioner in a lower precision than the matrix–vector product with $A$ (i.e., $u_g \geq u_m \gg u_a$) on real-life problems from the SuiteSparse collection (Davis and Hu [55]). This combination of precisions has not been explored in the literature so far and can be of interest, particularly when the application of the preconditioner is more costly than the application of $A$. To perform these experiments, we implemented Algorithm 7.2 in Julia.

To refer to a specific arithmetic, we will use the symbols B, H, S, D, and Q listed in Table 2.1. In addition, throughout this section, each variant of M-GMRES-IR6 will be presented in the form of a triplet $(u_g, u_m, u_a)$, so BSD means $u_g = \text{B}$, $u_m = \text{S}$, and $u_a = \text{D}$.

*7.3.3.1 Random dense matrices.* We first use random dense matrices to assess the validity of the convergence condition (7.42). Mainly, we want to observe the impact of the choice of precisions $u_g$, $u_m$, and $u_a$ on the convergence of M-GMRES-IR6 according to the properties of the problem $\rho'_A$, $\rho'_M$ and $\kappa(\widetilde{A})$. Unfortunately, it is difficult to generate problems for given $\rho'_A$ and $\rho'_M$; this is why we rather use the dense matrix generator described in section 7.2.3.1 that randomly generates $A$ and a preconditioner $M$ in double precision with given $\kappa(A)$ and $\kappa(M)$. With this class of matrices and preconditioners, we know that the generated $A$ and $M$ in double precision satisfies $\kappa(\widetilde{A}) = \kappa(A)/\kappa(M)$ and that the ratio $\rho'_A/\rho'_M$ is tightly related to the quantity $\kappa(A)$ and $\kappa(M)$, that is, the larger is $\kappa(A)$ and the closer is $\kappa(M)$ to $\kappa(A)$, the larger $\rho'_A/\rho'_M$ can be (see discussion in section 7.2.2 and numerical experiments in section 7.2.2.4). Throughout these experiments, we fix the precisions $u = \text{D}$ and $u_r = \text{Q}$ to guarantee that the limiting accuracy of the forward error does not depend on the condition number $\text{cond}(A, x)$ (see the limiting accuracy (7.41) of M-GMRES-IR6). We only focus on the precisions $u_g$, $u_m$, and $u_a$, and we will not attempt to study the effect of precision $u_f$.

We take $\kappa(A) = 10^c$ and $\kappa(M) = 10^c$, for $c = 0\colon 16$ and, for each couple $(\kappa(A), \kappa(M))$, we generate 100 random $50 \times 50$ matrices $A$ and $M$ of corresponding condition numbers with the generator described in section 7.2.3.1. Then, we run M-GMRES-IR6 for five different values of $\tau_g$ ($10^{-10}$, $10^{-8}$, $10^{-6}$, $10^{-4}$, and $10^{-2}$) on the 100 generated $A$ and $M$ and we keep for each of them the best number of cumulated GMRES iterations to achieve a forward error $\|x - \hat{x}\|_2 / \|x\|_2 \leq 1 \times 10^{-10}$.

Figure 7.3 displays for each couple $(\kappa(A), \kappa(M))$ the average number of iterations over the 100 randomly generated matrices. Each couple is represented by a tile with a specific color tone; the lighter the color, the higher the number of iterations. There are nine heatmaps associated with nine combinations of precisions $u_g$, $u_m$, and $u_a$. A blank tile is a couple $(\kappa(A), \kappa(M))$ for which M-GMRES-IR6 does not achieve the expected forward error. Note that the upper triangular parts of the plots are always blank because we do not allow $\kappa(M) > \kappa(A)$. For instance, with variant BBB, we observe that the convergence is limited to $\kappa(A), \kappa(M) \leq 10^3$ because the tiles corresponding to $\kappa(A) > 10^3$ are blank. In addition, we observe that the color tone is darker near the diagonal (i.e., $\kappa(A) \approx \kappa(M)$) and gets lighter below, meaning that we do fewer iterations when $\kappa(A) \approx \kappa(M)$; for example, see the tile $(\kappa(A) = 10^2, \kappa(M) = 10^2)$ compared with the tile $(\kappa(A) = 10^2, \kappa(M) = 10^0)$.

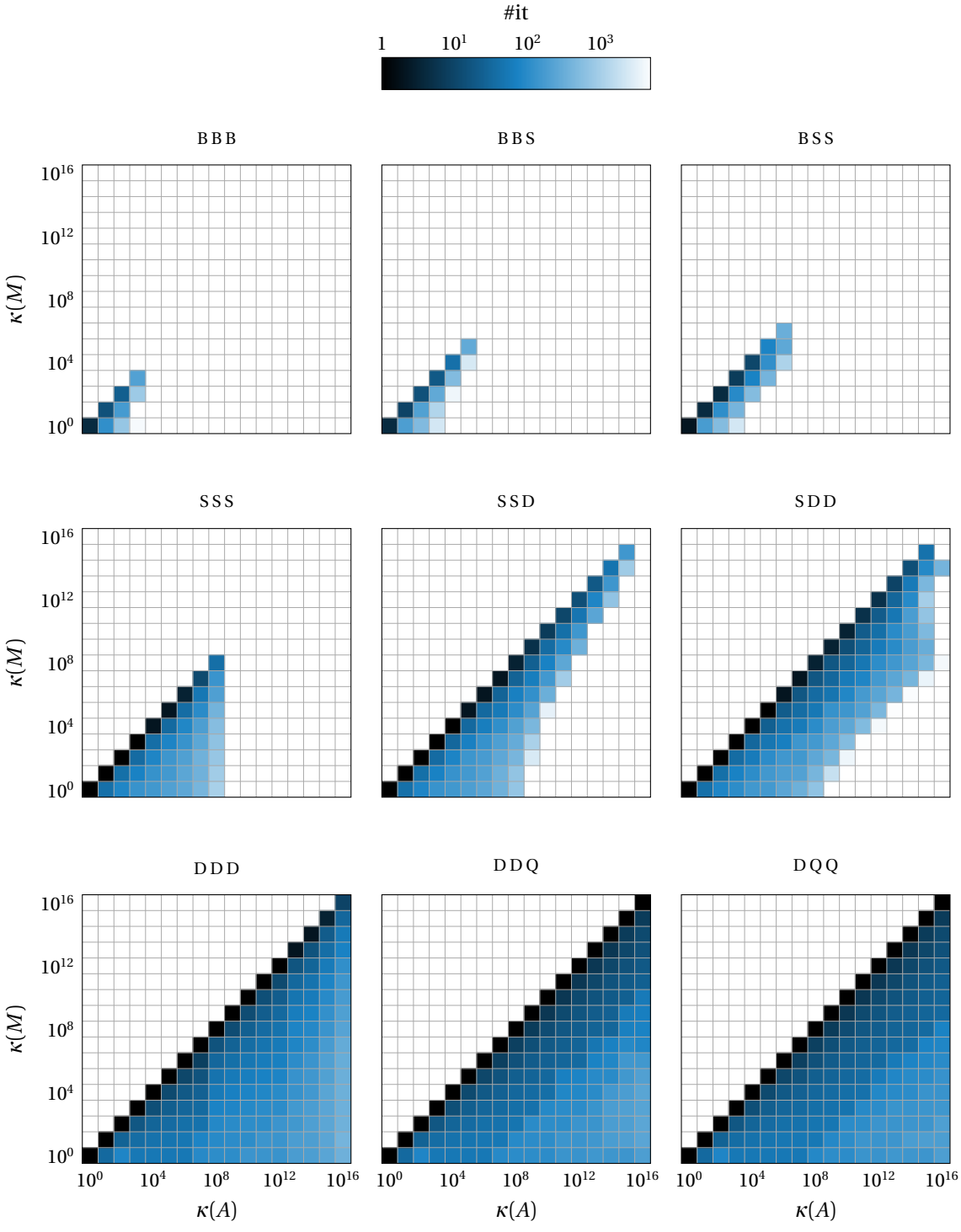



Figure 7.3: Average number of iterations in M-GMRES-IR6 according to $\kappa(A)$ and $\kappa(M)$ for different combinations of precisions $u_g$, $u_m$, and $u_a$. The iterations are stopped when we converged to $\|x - \hat{x}\|_2/\|x\|_2 \leq 1 \times 10^{-10}$. The preconditioner is built as described in section 7.2.3.1. We fix $u = \text{D}$ and $u_r = \text{Q}$.

We first focus on the first row of Figure 7.3, that is, variants BBB, BBS, and BSS:

- Switching from BBB to BSS - Compared with BBB, BSS is a combination where $u_a = u_m \ll u_g$; we observe that switching to this configuration allows converging on more tiles. These additional tiles verify $\kappa(A), \kappa(M) \leq 10^6$ and are located near the diagonal, that is, $\kappa(A) \approx \kappa(M)$. We can explain this effect with the convergence condition (7.42) of M-GMRES-IR6. Because the term $\kappa(\widetilde{A}) = \kappa(A)/\kappa(M)$ is kept relatively small and constant near the diagonal, and the term $u_a\rho'_A + u_m\rho'_M$, which is dominant compared with the term $u_g$ for variant BBB, is reduced with variant BSS, variant BSS achieves better convergence conditions than BBB near the diagonal. Far from the diagonal, BSS does not converge because $\kappa(\widetilde{A})$ grows too high due to the difference between $\kappa(A)$ and $\kappa(M)$.

- Switching from BBB to BBS - A major observation of this section is that BBS, which is a combination where $u_a \ll u_m = u_g$, is also converging on more tiles than variant BBB; this is even more noticeable from variant SSS to variant SSD. These additional tiles are located near the diagonal. This effect can also be explained by the convergence condition (7.42) of M-GMRES-IR6. In addition of the previous argument stating that the term $u_a\rho'_A + u_m\rho'_M$ is dominant compared with the term $u_g$ near the diagonal, we demonstrated in sections 7.2.2 and 7.2.3 that the term $u_a\rho'_A$ should be dominant over the term $u_m\rho'_M$ when $\kappa(A) \approx \kappa(M) \gg 1$. Therefore, increasing the precision $u_a$ only also improves the convergence condition near the diagonal.

- Switching from BBS to BSS - As BSS converges on more tiles than BBS, we can conclude that increasing the precision $u_m$ also positively affects the convergence. We can explain such a result from the convergence condition (7.42). Indeed, when $u_a \ll u_m = u_g$, we expect the term $u_m\rho_M\kappa(\widetilde{A})$ to be dominant; consequently, increasing the precision $u_m$ should reduce this term and improve the convergence condition.

  Variant BBS presents experimental convergence conditions that are better than variant BBB but worse than variant BSS. For this reason, BBS can be seen as a trade-off between these two variants.

The same behavior can be observed for the group of variants (SSS, SSD, SDD) on the second row. On the third row, for the group (DDD, DDQ, DQQ), variant DDD already converges on all the possible tiles. However, switching from DDD to DDQ and from DDQ to DQQ further reduces the number of iterations on tiles where we were already able to converge.

Finally, we comment on what happens when we switch from variant BSS to SSS. While SSS does not converge on much higher $\kappa(A)$ than BSS, we observe, however, that we can converge on the formerly blank tiles of the subdiagonal part. On the subdiagonal part, we know from our previous discussion in sections 7.2.2.4 and 7.2.3 that the terms $\rho'_A$ and $\rho'_M$ are smaller because $\kappa(M)$ is smaller. Consequently, the convergence condition (7.42) is expected to be dominated by the term $u_g\kappa(\widetilde{A})$. This is why increasing $u_g$ allows convergence on the subdiagonal tiles.

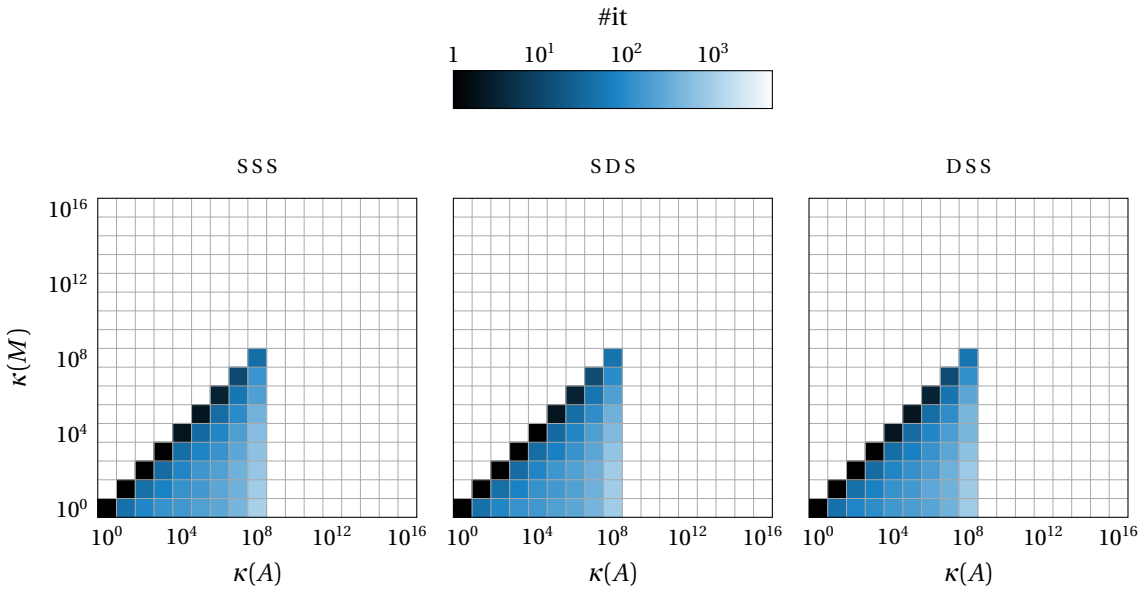



Figure 7.4: Supplement for Figure 7.3. Average number of iterations in M-GMRES-IR6 according to $\kappa(A)$ and $\kappa(M)$ for SSS, SDS, and DSS.

Figure 7.4 compares variants SDS and DSS with variant SSS. Both SDS and DSS use higher precisions but do not improve the convergence compared with SSS; that is, they are not converging on more tiles, and they do not reduce the number of iterations noticeably. This observation is not surprising since we concluded in section 7.3.2 that these variants are not theoretically meaningful. In the case of SDS, we have already explained that $u_m \ll u_a$ is not meaningful because it is very likely that $\rho_M \leq \rho_A$. In the case of DSS, having $u_g \ll u_m, u_a$ is not meaningful as well because it is very likely that $\rho_A \geq 1$.

This set of experiments demonstrated two main things. First, that our convergence condition (7.42) on the forward error seems to appropriately describe practical results obtained with M-GMRES-IR6. Second, that variants using a lower precision on the preconditioner application than the precision at which the matrix $A$ is applied (i.e., $u_g \geq u_m \gg u_a$) realize a trade-off between variants where $u_g = u_m = u_a$ and variants where $u_g \gg u_m = u_a$.

*7.3.3.2 Real-life matrices from SuiteSparse.* In this section, we seek to evaluate the interest of applying the preconditioner in a lower precision than the matrix–vector product with $A$, that is, $u_g \geq u_m \gg u_a$. This configuration has never been explored in previous studies and can potentially significantly enhance the performance of the solver, in particular, when the application of $M^{-1}$ is more costly than the application of $A$. It is the case, for instance, for sparse LU and ILUT preconditioners (see section 2.3.2), where the fill-in generates factors with a higher number of entries than $A$.

For these experiments we use LU and ILUT (with threshold $10^{-6}$) as preconditioners for M-GMRES-IR6. We compare variants from two groups: (SSS, SSD, SDD) and (HHH, HHS, HSS), where SSD and HHS are the newly proposed variants. For each variant, we run M-GMRES-IR6 for $\tau_g = 10^c$ where $c = -10 : -1$, and we keep the $\tau_g$ leading to the least

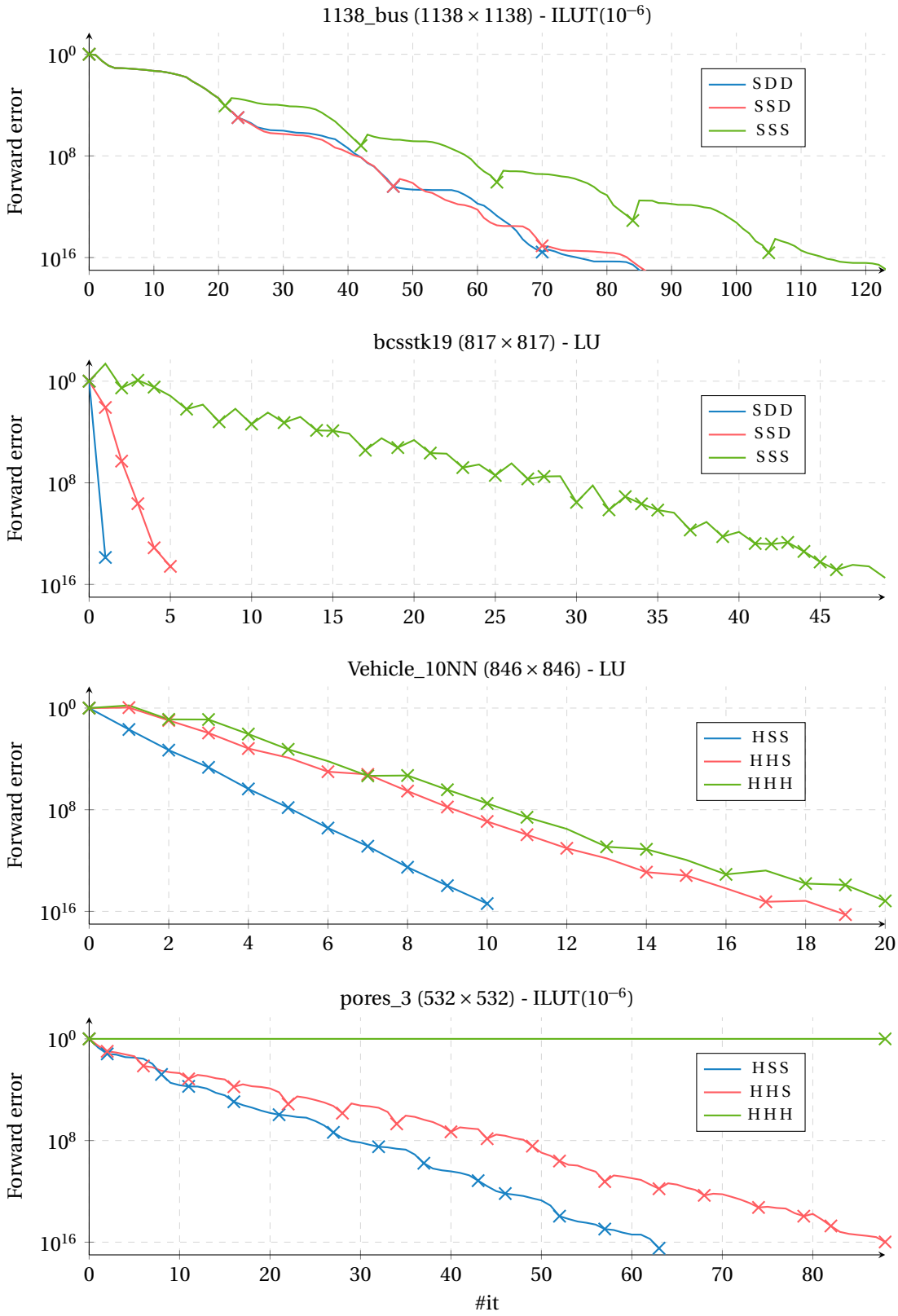



Figure 7.5: Evolution of the forward error for different variants of M-GMRES-IR6 on four SuiteSparse matrices.

number of cumulated GMRES iterations to reach a forward error with double precision accuracy.

We display in Figure 7.5 the evolution of the forward error (of the original system (1.1)) of the different variants throughout the iterations for four SuiteSparse matrices. On each plot, we specify the name of the matrix, its dimension, and the preconditioner used. A cross means that M-GMRES-IR6 has been restarted at this iteration. The new variants SSD and HHS are plotted in red, variants SSS and HHH with $u_g = u_m = u_a$ are plotted in green, and variants SDD and HSS with $u_g \gg u_m = u_a$ are plotted in blue. We now comment on the results obtained for each of these four matrices.

1138_bus with the ILUT($10^{-6}$) preconditioner is a very favorable case for variant SSD because SSD converges in less iterations than SSS and with the same number of iterations as SDD. Therefore, as SSD applies the preconditioner in a lower precision than SDD while converging at the same rate, it is surely more interesting to use SSD over SDD for this specific case.

For bcsstk19 with an LU preconditioner, variant SSD converges in fewer iterations than variant SSS but in more iterations than variant SDD; therefore, SSD is a trade-off between the two. To extrapolate on the potential relative performance of each variant, we can consider that single precision operations are twice faster as double precision ones. Consequently, we can expect the execution time of SSD to be shorter than the one of SSS in achieving double precision accuracy; however, we expect it to be higher than the one of SDD which converges in just one iteration. On the other hand, SSD does not need to cast the LU factors in double precision and might consume less memory than SDD for just a small time overhead.

Vehicle_10NN with an LU preconditioner is an example of an unfavorable case for variant HHS. For this matrix, the convergence of HHS is very similar to variant HHH, which is applying $A$ in precision $u_a = \mathrm{H}$ instead of $u_a = \mathrm{S}$ as for HHS. Hence, one iteration of HHH is less costly than one iteration of HHS and, as they do approximately the same number of iterations, HHH is expected to be faster in computing a double precision accuracy solution. Therefore, in this case, increasing the precision for the application of $A$ alone is not attractive, and we would prefer variant HHH over variant HHS.

Finally, pores_3 with ILUT($10^{-6}$) preconditioner is an example of matrix where variant HHH does not converge. In this case, we can only choose between variants HHS and HSS. As HHS does less than two times more iterations than HSS, HHS can be faster than HSS if the application of the preconditioner is the dominant operation in the GMRES iterations. For instance, with ILUT, the application of $A$ can become negligible compared with the application of the preconditioner if we authorize ILUT to generate substantial fill-in.

## 7.4 Conclusion

In this chapter, we have proposed a new mixed precision algorithm for GMRES that we call M-GMRES-IR6. Compared with LU-GMRES-IR5 presented in chapter 5, this algorithm splits the precision at which the preconditioned matrix–vector product is computed in two precisions $u_m$ and $u_a$ for respectively the application of the preconditioner and the

multiplication with the matrix $A$. As M-GMRES-IR6 uses an arbitrary preconditioner $M$ and can be left- or right-preconditioned, it works as a framework that contains most of the mixed precision strategies for GMRES present in the literature. It even proposes new meaningful combinations of precisions that have never been considered before and can be of practical interest. Specifically, applying the preconditioner in lower precision than the matrix $A$ can offer substantial gains in performance if the application of the preconditioner is the dominant operation in the GMRES iterations. Note that even if we described the right-preconditioned formulations in Algorithms 7.1 and 7.2, we exclusively focused on the left-preconditioned case.

To analyse this algorithm, we proceeded in two main steps. First, we derived a backward stability result for left MGS-GMRES in mixed precision and studied the left-preconditioned matrix–vector product kernel to assess the relevance of decoupling the precisions $u_m$ and $u_a$. We validated our theoretical observations with experiments on random dense matrices. Second, we used these previous results to derive convergence conditions on the forward and normwise backward errors of M-GMRES-IR6. We then demonstrated the validity of the forward error convergence condition by numerical experiments with randomly generated and real-life sparse matrices. In addition, we illustrated on real-life problems that applying the preconditioner with lower precision than the matrix–vector product with $A$ can potentially improve the performance.

The work described in this chapter is an ongoing work that has not been submitted for publication yet.

# 8 Conclusion

## 8.1 Summary

In this manuscript, we have investigated the use of mixed precision iterative refinement for improving the solution of large sparse linear systems of the form (1.1). We showed that we could significantly reduce the resource consumption of sparse direct and iterative solvers while still guaranteeing robustness and accuracy.

In chapter 2, we first summarized the different notions and tools used throughout this document. In particular, we covered the basics of floating-point rounding error analysis and recalled major results on LU and QR direct solvers. We also covered the basics of sparse direct factorization and GMRES iterative solvers for the solution of (1.1).

Then, we provided in chapters 3 and 4 an in-depth overview of iterative refinement, which is the central algorithm of this manuscript. These chapters cover essential results upon which the contributions of this manuscript are built; they can also be used for an up-to-date comprehensive introduction to iterative refinement. Chapter 3 is a chronological survey listing the different research studies on iterative refinement. In particular, it depicts the different evolutions of this algorithm, gives a better insight on state-of-the-art iterative refinement, and explains how it reflects the current context and needs of present-day scientific computing. Chapter 4 focuses on listing the major technical results on state-of-the-art iterative refinement. It especially summarizes the recent work of Carson and Higham [44; 45] that played a major role in the renewed interest this algorithm recently received and on which most of our contributions are based.

The first primary concern of this document was to improve sparse direct solvers for the solution of (1.1). It was done in two steps.

First, in chapter 5, we addressed the central issue of LU-GMRES-IR3, that is, the application of the LU factors in extra-precision $u^2$. When the working precision $u$ is double precision, the factors need to be applied in quadruple precision, which might not be supported by the hardware and, hence, might be highly inefficient. In addition, many high-end sparse direct solver software does not provide a quadruple precision implementation for applying the LU solves. For these reasons, LU-GMRES-IR3 is not viable for many applications as it is. Our approach was to relax the precisions used inside the GMRES solver in two

independent precisions: $u_p \geq u^2$ (for the computation of the preconditioned matrix–vector product) and $u_g \geq u$ (for the rest of the operations). We did a rounding error analysis of the resulting algorithm called LU-GMRES-IR5 to derive new conditions for the convergence of the forward and backward errors of the solution of (1.1). A key step of this analysis was to prove the backward stability of a mixed precision MGS-GMRES method. We then identified the subset of meaningful combinations of precisions for this algorithm and assessed the relevance of our theoretical results with numerical experiments on a wide set of matrices.

Second, in chapter 6, we showed that state-of-the-art iterative refinements, that is, LU-IR3 and LU-GMRES-IR5, can greatly reduce the time and memory consumption of sparse direct solvers for the solution of (1.1) while preserving accuracy and robustness. We demonstrated that, combined with high-end approximate factorizations often used in state-of-the-art sparse direct solvers, we could achieve a reduction up to 5.6× in time and 4.4× in memory on our set of matrices coming from industrial applications. We supported this study with a new rounding error analysis of LU-IR3 and LU-GMRES-IR5 that covers the use of approximations and which better fits a realistic common use of sparse direct solvers.

Finally, we turned our attention to the improvement of the GMRES iterative solver through mixed precision in chapter 7. It was the other primary concern of this manuscript. From the observation that many mixed precision approaches for GMRES are based on the application of an iterative refinement process or/and the mixed precision application of various preconditioners, we proposed a new framework that gathers all these approaches under a shared analysis. It took the form of a new algorithm that we call M-GMRES-IR6, which is an iterative refinement variant using an arbitrary preconditioned GMRES for the solution of the correction equation and composed of six independent precision parameters. We carried out a rounding error analysis and demonstrated that it was relevant, from both a numerical and performance standpoint, to apply the preconditioner with lower precision than the matrix–vector product with $A$.

## 8.2 Future work

We now briefly discuss a few remaining challenges and open questions that could be the object of future work.

LU-GMRES-IR3 from Carson and Higham [44; 45] and LU-GMRES-IR5 introduced in chapter 5 are using a GMRES iterative solver for the solution of the correction equation. However, the same approach can be straightforwardly adapted to many other iterative solvers (e.g., CG, BiCG, or MINRES). The main reasons for this specific choice of iterative solver instead of others are that:

- To use Theorems 4.1 and 4.2 to determine convergence conditions for a given specialization of iterative refinement, we need the linear solver for the solution of the correction equation to satisfy the stability properties (4.4)–(4.6). It is because left-preconditioned MGS-GMRES is backward stable that we can carry out the analysis. On the other hand, for example, preconditioned CG is not; therefore, we cannot derive convergence conditions for this specific choice of solver.

- GMRES solves general square linear systems and is not confined, for example, to symmetric problems as for MINRES.

- GMRES is very popular, is widely known and used, and has been the object of many research studies and improvements.

Regardless of these different reasons, it can still be of interest to explore the use of other iterative solvers and compare their practical efficiency, even if a numerical analysis does not cover them.

We observed, in the numerical experiments of section 5.5 concerning the convergence of LU-GMRES-IR5, that the experimental forward error convergence condition is better than the theoretical one (5.2). It tends to reveal that the convergence conditions of Theorems 4.4 and 5.1 might be pessimistic to some extent. We believe that the reason for this discrepancy might come from the regularization phenomenon, discussed in section 7.3.2, that can occur when $A$ is cast in precision $u_f$ for the computation of the LU factors. From this observation, it might be interesting to try incorporating the regularization in our analysis to better capture its effect.

Concerning the parallel implementations of LU-IR3 and LU-GMRES-IR5 for the solution of large sparse systems that we covered in chapter 6, a few directions and challenges remain to be explored.

A direct and natural improvement of the experiments done in chapter 6 is to target half precision factorization. Because sparse direct solvers cannot exploit GPUs as efficiently as dense solvers due to the lower granularity of the operations, it is difficult to proceed to an efficient full half precision factorization. However, the increasing availability of half precision in the CPUs will greatly simplify the exploitation of this arithmetic for the direct solution of sparse linear systems. Hence, half precision in CPUs would provide reliable access to this arithmetic that does not depend on if the solver can efficiently use the GPU or not. However, the promise of a full half precision sparse factorization comes with a few pending questions:

- The first is to deal with overflow and underflow. The scaling approach in section 4.7 prevents overflow and reduces underflows when $A$ is cast in precision $u_f$. However, the entries of the computed LU factors in precision $u_f$ might still severely underflow because a large pivot can generate really small values, and still overflow because, even with GEPP, the growth factor is not guaranteed to be small.

- The second is the fact that many real-life and industrial applications present large condition number. For instance, the median condition number of our set of large sparse problems in chapter 6 listed in Table 6.1 is $\kappa(A) = 4 \times 10^6$, while the condition for convergence of LU-IR3 is $\kappa(A) \leq 2 \times 10^3$. Consequently, LU-IR3 would potentially be unable to compute most of our problems. Moreover, even though LU-GMRES-IR5 can achieve better convergence conditions, it is unclear how many iterations the algorithm would need to converge and if it can stay competitive with a variant using higher precision in the factorization.

The promise of easier half precision access in computers also raises questions about implementing maintainable high performance mixed precision algorithms. In our case, we used the Fortran 2018 standard for our parallel implementations of LU-IR3 and LU-GMRES-IR5. As Fortran does not provide templating features, two versions of the same routine, one in single precision and the other in double (say), must be written twice for both precisions. The main issue is that, with four different arithmetics available (e.g., half, single, double, and quadruple), a mixed precision algorithm can potentially be derived in many combinations of its precisions. Necessarily, writing a specific code for each of these combinations is not feasible because, with a high combinatory on the precisions of the algorithm, it would lead to an unmaintainable amount of code. It leads to questioning the best practices for coding efficient mixed precision algorithms that can offer a wide variety of combinations. For our own implementation, we built a source-to-source precompiler that generates every required combination of precisions of a given routine. While this solution is viable, it can generate a heavy load of source code due to the high combinatory, which would slow down the compilation. Other approaches are possible, such as relying on the object-oriented programming features of Fortran or using programming languages supporting templates (e.g., C++). A last promising option is using languages like Julia that are doing Just In Time compilation to offer both genericity to the code and performance. It is the language we use for the experiments of chapters 5 and 7; it would be interesting to evaluate to which extent it can be used to run high performance computing applications.

Lastly, we mentioned in section 6.4.3 two possibilities for the cast of the factors from precision $u_f$ to precision $u_p$ in LU-GMRES-IR5. We can either make an explicit copy of the factors or cast them on the fly during forward and backward substitutions. We chose the explicit copy approach that led to the least amount of time overhead, but cast on the fly remains very interesting from a memory standpoint. The promising results of Anzt et al. [24], Flegar et al. [73], which demonstrated that it was possible to even save time on memory-bound applications by casting on the fly from low to high precision, definitely raised our interest in implementing an efficient cast on the fly LU solve for the MUMPS solver.

Finally, a few major ways of improvement need to be considered to complete the work of chapter 7 on M-GMRES-IR6:

- A crucial missing piece is an error analysis of the right-preconditioned case. While the cost of an inner iteration of left and right M-GMRES-IR6 is equivalent, it is of interest to understand if the right M-GMRES-IR6 presents major numerical differences or not compared with the left M-GMRES-IR6.

- We know that the rounding errors made in the preconditioned matrix–vector product kernels depend on the preconditioner and the way it is applied. So far, we have considered explicit and implicit construction via LU factorization. However, it could be interesting to conduct analyses on other preconditioners, such as polynomial or when an iterative solver is used as a preconditioner.

- In section 7.3.3, we evaluated how the different precisions $u_m$, $u_a$, and $u_g$ (resp. the precision for the application of the preconditioner, for the matrix–vector product with $A$, and for the rest of the GMRES operations) affect the convergence. However,

we did not consider the fourth precision $u_f$ at which the preconditioner is computed and which is also expected to affect the convergence.

- We showed cases where applying the preconditioner in a lower precision than the matrix–vector products with $A$ for the solution of (1.1) was relevant numerically. We extrapolate that it might have some performance benefits for the solution of large sparse linear systems. However, to assess it undoubtedly, we need to make a performance study of a parallel implementation of M-GMRES-IR6.

# Scientific communications

## 8.2 Journal articles

[J2] Patrick R. Amestoy, Alfredo Buttari, Nicholas J. Higham, Jean-Yves L'Excellent, Théo Mary, and Bastien Vieublé. Combining sparse approximate factorizations with mixed precision iterative refinement. To appear in *ACM Transactions on Mathematical Software*, October 2022.

[J1] Patrick R. Amestoy, Alfredo Buttari, Nicholas J. Higham, Jean-Yves L'Excellent, Théo Mary, and Bastien Vieublé. Five-precision GMRES-based iterative refinement. Preprint, April 2021.

## 8.2 Talks in conferences

[C8] Patrick R. Amestoy, Alfredo Buttari, Nicholas J. Higham, Jean-Yves L'Excellent, Théo Mary, and Bastien Vieublé. Mixed precision strategies for preconditioned GMRES. *Sparse Days (In-Person)*, St Girons (France), June 2022.

[C7] Patrick R. Amestoy, Alfredo Buttari, Nicholas J. Higham, Jean-Yves L'Excellent, Théo Mary, and Bastien Vieublé. Combining sparse approximate factorizations with mixed precision iterative refinement. *ISC High Performance (ISC 22) (In-Person)*, Hamburg (Germany), May 2022.

[C6] Patrick R. Amestoy, Alfredo Buttari, Nicholas J. Higham, Jean-Yves L'Excellent, Théo Mary, and Bastien Vieublé. Mixed Precision Iterative Refinement with Approximate Factorization for the Solution of Large Sparse Systems. *SIAM Conference on Parallel Processing for Scientific Computing (SIAM PP 22) (Virtual)*, February 2022.

[C5] Patrick R. Amestoy, Alfredo Buttari, Nicholas J. Higham, Jean-Yves L'Excellent, Théo Mary, and Bastien Vieublé. Modern iterative refinement methods for the solution of large sparse linear systems. *Congrès des Jeunes Chercheuses et Chercheurs en Mathématiques Appliquées (CJC-MA) (In-Person)*, Paris (France), October 2021.

[C4] Patrick R. Amestoy, Alfredo Buttari, Nicholas J. Higham, Jean-Yves L'Excellent, Théo Mary, and Bastien Vieublé. Mixed precision iterative refinement for the solution of large sparse linear systems. *12èmes Rencontres Arithmétique de l'Informatique Mathématique (RAIM 21) (Virtual)*, May 2021.

[C3] Patrick R. Amestoy, Alfredo Buttari, Nicholas J. Higham, Jean-Yves L'Excellent, Théo Mary, and Bastien Vieublé. GMRES-based iterative refinement in up to five precisions. *SIAM Conference on Computational Science and Engineering (SIAM CSE 21) (Virtual)*, March 2021.

[C2] Patrick R. Amestoy, Alfredo Buttari, Nicholas J. Higham, Jean-Yves L'Excellent, Théo Mary, and Bastien Vieublé. Mixed precision iterative refinement for the solution of large sparse linear systems. *World Congress on Computational Mechanics and European Congress on Computational Methods in Applied Sciences and Engineering (WCCM-ECCOMAS 20) (Virtual)*, January 2021.

[C1] Patrick R. Amestoy, Alfredo Buttari, Nicholas J. Higham, Jean-Yves L'Excellent, Théo Mary, and Bastien Vieublé. Mixed precision iterative refinement for the solution of large sparse linear systems. *Sparse Days (Virtual)*, November 2020.

# Bibliography


[1] Bfloat16—hardware numerics definition. *Intel Corporation*, (338302-001US), 2018. URL https://www.intel.com/content/dam/develop/external/us/en/documents/bf16-hardware-numerics-definition-white-paper.pdf. 9

[2] TOP500, 2022. URL https://www.top500.org/lists/top500/2022/06/. 1, 9, 52

[3] Ahmad Abdelfattah, Hartwig Anzt, Erik G. Boman, Erin Carson, Terry Cojean, Jack Dongarra, Alyson Fox, Mark Gates, Nicholas J. Higham, Xiaoye S. Li, Jennifer Loe, Piotr Luszczek, Srikara Pranesh, Siva Rajamanickam, Tobias Ribizel, Barry F. Smith, Kasia Swirydowicz, Stephen Thomas, Stanimire Tomov, Yaohung M. Tsai, and Ulrike Meier Yang. A survey of numerical linear algebra methods utilizing mixed-precision arithmetic. *The International Journal of High Performance Computing Applications*, 35(4):344–369, March 2021. doi: 10.1177/10943420211003313. URL https://doi.org/10.1177/10943420211003313. 12, 53

[4] Emmanuel Agullo, James W. Demmel, Jack Dongarra, Bilel Hadri, Jakub Kurzak, Julien Langou, Hatem Ltaief, Piotr Luszczek, and Stanimire Tomov. Numerical linear algebra on emerging architectures: The PLASMA and MAGMA projects. *Journal of Physics: Conference Series*, 180:012037, July 2009. doi: 10.1088/1742-6596/180/1/012037. URL https://doi.org/10.1088/1742-6596/180/1/012037. 49

[5] Emmanuel Agullo, Patrick R. Amestoy, Alfredo Buttari, Abdou Guermouche, Jean-Yves L'Excellent, and François-Henry Rouet. Robust memory-aware mappings for parallel multifrontal factorizations. *SIAM Journal on Scientific Computing*, 38(3):C256–C279, 2016. doi: 10.1137/130938505. URL https://doi.org/10.1137/130938505. 133

[6] Emmanuel Agullo, Luc Giraud, Stojce Nakov, and Jean Roman. Hierarchical hybrid sparse linear solver for multicore platforms. Research Report RR-8960, INRIA Bordeaux, October 2016. URL https://hal.inria.fr/hal-01379227. 2

[7] Emmanuel Agullo, Franck Cappello, Sheng Di, Luc Giraud, Xin Liang, and Nick Schenkels. Exploring variable accuracy storage through lossy compression techniques in numerical linear algebra: a first application to flexible GMRES. Research

Report RR-9342, Inria Bordeaux Sud-Ouest, May 2020. URL https://hal.inria.fr/hal-02572910. 150

[8] Hussam Al Daas, Laura Grigori, Pierre Jolivet, and Pierre-Henri Tournier. A Multilevel Schwarz Preconditioner Based on a Hierarchy of Robust Coarse Spaces. 43(3):A1907–A1928, 2021. URL https://github.com/prj-/aldaas2019multi. 123

[9] José I. Aliaga, Hartwig Anzt, Thomas Grützmacher, Enrique S. Quintana-Ortí, and Andrés E. Tomás. Compressed basis GMRES on high-performance graphics processing units. *The International Journal of High Performance Computing Applications*, page 109434202211151, August 2022. doi: 10.1177/10943420221115140. URL https://doi.org/10.1177/10943420221115140. 150

[10] Patrick Amestoy, Olivier Boiteau, Alfredo Buttari, Matthieu Gerest, Fabienne Jézéquel, Jean-Yves L'Excellent, and Theo Mary. Mixed precision low-rank approximations and their application to block low-rank LU factorization. *IMA Journal of Numerical Analysis*, August 2022. doi: 10.1093/imanum/drac037. URL https://doi.org/10.1093/imanum/drac037. 29

[11] Patrick R. Amestoy, Timothy A. Davis, and Iain S. Duff. An Approximate Minimum Degree Ordering Algorithm. *SIAM Journal on Matrix Analysis and Applications*, 17(4):886–905, October 1996. doi: 10.1137/s0895479894278952. URL https://doi.org/10.1137%2Fs0895479894278952. 23

[12] Patrick R. Amestoy, Iain S. Duff, Jean-Yves L'Excellent, and Jacko Koster. A fully asynchronous multifrontal solver using distributed dynamic scheduling. *SIAM Journal on Matrix Analysis and Applications*, 23(1):15–41, 2001. doi: 10.1137/S0895479899358194. URL https://doi.org/10.1137/S0895479899358194. 26, 46, 120

[13] Patrick R. Amestoy, Iain S. Duff, Jean-Yves L'Excellent, and Xiaoye S. Li. Analysis and comparison of two general sparse solvers for distributed memory computers. *ACM Transactions on Mathematical Software*, 27(4):388–421, December 2001. doi: 10.1145/504210.504212. URL https://doi.org/10.1145%2F504210.504212. 46

[14] Patrick R. Amestoy, Cleve Ashcraft, Olivier Boiteau, Alfredo Buttari, Jean-Yves L'Excellent, and Clément Weisbecker. Improving multifrontal methods by means of block low-rank representations. *SIAM Journal on Scientific Computing*, 37(3):A1451–A1474, 2015. doi: 10.1137/120903476. URL https://doi.org/10.1137/120903476. 27, 120

[15] Patrick R. Amestoy, Romain Brossier, Alfredo Buttari, Jean-Yves L'Excellent, Theo Mary, Ludovic Métivier, Alain Miniussi, and Stephane Operto. Fast 3D frequency-domain full-waveform inversion with a parallel block low-rank multifrontal direct solver: Application to OBC data from the North Sea. *GEOPHYSICS*, 81(6):R363–R383, November 2016. doi: 10.1190/geo2016-0052.1. URL https://doi.org/10.1190%2Fgeo2016-0052.1. 27

[16] Patrick R. Amestoy, Alfredo Buttari, Jean-Yves L'Excellent, and Theo Mary. On the complexity of the block low-rank multifrontal factorization. *SIAM Journal on Scientific Computing*, 39(4):A1710–A1740, 2017. doi: 10.1137/16M1077192. URL https://doi.org/10.1137/16M1077192. 27, 28

[17] Patrick R. Amestoy, Alfredo Buttari, Jean-Yves L'Excellent, and Theo Mary. Performance and scalability of the block low-rank multifrontal factorization on multicore architectures. *ACM Transactions on Mathematical Software*, 45(1), February 2019. ISSN 0098-3500. doi: 10.1145/3242094. URL https://doi.org/10.1145/3242094. 27, 120, 143

[18] Patrick R. Amestoy, Alfredo Buttari, Nicholas J. Higham, Jean-Yves L'Excellent, Théo Mary, and Bastien Vieublé. Five-precision GMRES-based iterative refinement. Preprint, April 2021. URL http://eprints.maths.manchester.ac.uk/id/eprint/2807. 56, 57, 65, 110, 149, 152

[19] Patrick R. Amestoy, Alfredo Buttari, Nicholas J. Higham, Jean-Yves L'Excellent, Théo Mary, and Bastien Vieublé. Combining sparse approximate factorizations with mixed precision iterative refinement. To appear in *ACM Transactions on Mathematical Software*, January 2022. URL https://hal.archives-ouvertes.fr/hal-03536031. 56, 57, 65, 148, 152

[20] Bernard Van Antwerpen, Yves Detandt, Diego Copiello, Eveline Rosseel, and Eloi Gaudry. *Performance improvements and new solution strategies of Actran/TM for nacelle simulations*. 2014. doi: 10.2514/6.2014-2315. URL https://arc.aiaa.org/doi/abs/10.2514/6.2014-2315. 123

[21] Hartwig Anzt, Vincent Heuveline, and Björn Rocker. An Error Correction Solver for Linear Systems: Evaluation of Mixed Precision Implementations. In *Lecture Notes in Computer Science*, pages 58–70. Springer Berlin Heidelberg, 2011. doi: 10.1007/978-3-642-19328-6_8. URL https://doi.org/10.1007/978-3-642-19328-6_8. 48, 57, 150, 152

[22] Hartwig Anzt, Vincent Heuveline, and Björn Rocker. Mixed Precision Iterative Refinement Methods for Linear Systems: Convergence Analysis Based on Krylov Subspace Methods. In *Applied Parallel and Scientific Computing*, pages 237–247. Springer Berlin Heidelberg, 2012. doi: 10.1007/978-3-642-28145-7_24. URL https://doi.org/10.1007/978-3-642-28145-7_24. 48, 57, 150, 152

[23] Hartwig Anzt, Jack Dongarra, Goran Flegar, Nicholas J. Higham, and Enrique S. Quintana-Ortí. Adaptive precision in block-jacobi preconditioning for iterative sparse linear system solvers. *Concurrency and Computation: Practice and Experience*, 31(6):e4460, March 2018. doi: 10.1002/cpe.4460. URL https://doi.org/10.1002/cpe.4460. 150, 152

[24] Hartwig Anzt, Jack Dongarra, Goran Flegar, Nicholas J. Higham, and Enrique S. Quintana-Ortí. Adaptive precision in block-jacobi preconditioning for iterative

sparse linear system solvers. *Concurrency and Computation: Practice and Experience*, 31(6):e4460, 2019. doi: https://doi.org/10.1002/cpe.4460. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/cpe.4460. e4460 cpe.4460. 12, 120, 127, 178

[25] Mario Arioli and Iain S. Duff. Using FGMRES to obtain backward stability in mixed precision. *ETNA. Electronic Transactions on Numerical Analysis [electronic only]*, 33: 31–44, 2008. URL http://eudml.org/doc/130614. 4, 36, 48, 56, 149, 150, 152

[26] Mario Arioli, James W. Demmel, and Iain S. Duff. Solving Sparse Linear Systems with Sparse Backward Error. *SIAM Journal on Matrix Analysis and Applications*, 10(2):165–190, April 1989. doi: 10.1137/0610013. URL https://doi.org/10.1137/0610013. 44, 45, 56, 112, 116, 119

[27] Mario Arioli, Iain S. Duff, Serge Gratton, and Stéphane Pralet. A Note on GMRES Preconditioned by a Perturbed $LDL^T$ Decomposition with Static Pivoting. *SIAM Journal on Scientific Computing*, 29(5):2024–2044, 2007. doi: 10.1137/060661545. URL https://doi.org/10.1137/060661545. 36, 48, 56, 150

[28] Walter E. Arnoldi. The principle of minimized iterations in the solution of the matrix eigenvalue problem. *Quarterly of Applied Mathematics*, 9(1):17–29, 1951. doi: 10.1090/qam/42792. URL https://doi.org/10.1090%2Fqam%2F42792. 31

[29] Marc Baboulin, Alfredo Buttari, Jack Dongarra, Jakub Kurzak, Julie Langou, Julien Langou, Piotr Luszczek, and Stanimire Tomov. Accelerating scientific computations with mixed precision algorithms. *Computer Physics Communications*, 180(12):2526–2533, December 2009. doi: 10.1016/j.cpc.2008.11.005. URL https://doi.org/10.1016/j.cpc.2008.11.005. 3, 48, 56, 112

[30] Oleg Balabanov and Laura Grigori. Randomized Gram–Schmidt Process with Application to GMRES. *SIAM Journal on Scientific Computing*, 44(3):A1450–A1474, June 2022. doi: 10.1137/20m138870x. URL https://doi.org/10.1137%2F20m138870x. 150

[31] Friedrich L. Bauer. Elimination with Weighted Row Combinations for Solving Linear Equations and Least Squares Problems. In *Handbook for Automatic Computation*, pages 119–133. Springer Berlin Heidelberg, 1971. doi: 10.1007/978-3-642-86940-2_9. URL https://doi.org/10.1007/978-3-642-86940-2_9. 43, 57, 71

[32] Åke Björck. Iterative refinement of linear least squares solutions I. *BIT*, 7(4):257–278, December 1967. doi: 10.1007/bf01939321. URL https://doi.org/10.1007/bf01939321. 43, 45, 47, 50, 52, 56, 57, 72

[33] Åke Björck. Comment on the iterative refinement of least-squares solutions. *Journal of the American Statistical Association*, 73(361):161–166, 1978. ISSN 01621459. URL http://www.jstor.org/stable/2286538. 43

[34] Åke Björck and Gene H. Golub. Iterative refinements of linear least squares solutions by Householder transformations. Technical report, January 1968. URL http://i.stanford.edu/pub/cstr/reports/cs/tr/68/83/CS-TR-68-83.pdf. 43, 57

[35] Pierre Blanchard, Nicholas J. Higham, Florent Lopez, Theo Mary, and Srikara Pranesh. Mixed precision block fused multiply-add: Error analysis and application to GPU tensor cores. *SIAM Journal on Scientific Computing*, 42(3):C124–C141, January 2020. doi: 10.1137/19m1289546. URL https://doi.org/10.1137/19m1289546. 52

[36] Daniel Boley, Gene H. Golub, Samy Makar, Nirmal Saxena, and Edward J. McCluskey. Floating point fault tolerance with backward error assertions. *IEEE Transactions on Computers*, 44(2):302–311, 1995. doi: 10.1109/12.364541. URL https://doi.org/10.1109/12.364541. 45, 56, 57

[37] Hilary J. Bowdler, Roger S. Martin, G. Peters, and James H. Wilkinson. Solution of real and complex systems of linear equations. *Numerische Mathematik*, 8(3):217–234, May 1966. doi: 10.1007/bf02162559. URL https://doi.org/10.1007/bf02162559. 42, 56

[38] Hilary J. Bowdler, Roger S. Martin, G. Peters, and James H. Wilkinson. Solution of real and complex systems of linear equations. In *Handbook for Automatic Computation*, pages 93–110. Springer Berlin Heidelberg, 1971. doi: 10.1007/978-3-642-86940-2_7. URL https://doi.org/10.1007/978-3-642-86940-2_7. 42, 56

[39] Peter Businger and Gene H. Golub. Linear least squares solutions by Householder transformations. *Numerische Mathematik*, 7(3):269–276, June 1965. doi: 10.1007/bf01436084. URL https://doi.org/10.1007/bf01436084. 43, 57

[40] Alfredo Buttari. *Scalability of parallel sparse direct solvers: methods, memory and performance*. Habilitation à diriger des recherches, Toulouse INP, September 2018. URL https://hal.archives-ouvertes.fr/tel-01913033. 21

[41] 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, November 2007. doi: 10.1177/1094342007084026. URL https://doi.org/10.1177/1094342007084026. 47, 56

[42] Alfredo Buttari, Jack Dongarra, Jakub Kurzak, Piotr Luszczek, and Stanimir Tomov. Using Mixed Precision for Sparse Matrix Computations to Enhance the Performance while Achieving 64-bit Accuracy. *ACM Transactions on Mathematical Software*, 34(4):1–22, July 2008. doi: 10.1145/1377596.1377597. URL https://doi.org/10.1145/1377596.1377597. 3, 4, 48, 56, 112, 150, 152

[43] Alfredo Buttari, Julien Langou, Jakub Kurzak, and Jack Dongarra. A class of parallel tiled linear algebra algorithms for multicore architectures. *Parallel Computing*, 35(1):38–53, January 2009. doi: 10.1016/j.parco.2008.10.002. URL https://doi.org/10.1016%2Fj.parco.2008.10.002. 17

[44] 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, January 2017. doi: 10.1137/17m1122918. URL https://doi.org/10.1137/17m1122918. 3, 51, 56, 57, 64, 68, 69, 81, 82, 83, 92, 99, 114, 151, 152, 175, 176, 205

[45] 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, January 2018. doi: 10.1137/17m1140819. URL https://doi.org/10.1137/17m1140819. 3, 12, 36, 51, 52, 56, 57, 61, 62, 63, 65, 66, 68, 69, 80, 81, 83, 88, 89, 91, 92, 110, 114, 149, 151, 152, 175, 176, 205

[46] Erin Carson and Noaman Khan. Mixed Precision Iterative Refinement with Sparse Approximate Inverse Preconditioning. 2022. doi: 10.48550/ARXIV.2202.10204. URL https://arxiv.org/abs/2202.10204. 52, 56, 57, 65, 70, 151, 152

[47] 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, January 2020. doi: 10.1137/20m1316822. URL https://doi.org/10.1137/20m1316822. 52, 56, 57, 65, 73, 74, 75, 80, 105, 109

[48] Erin Carson, Tomáš Gergelits, and Ichitaro Yamazaki. Mixed precision s-step Lanczos and conjugate gradient algorithms. *Numerical Linear Algebra with Applications*, 29 (3), November 2021. doi: 10.1002/nla.2425. URL https://doi.org/10.1002/nla.2425. 149, 150

[49] Jack Choquette, Wishwesh Gandhi, Olivier Giroux, Nick Stam, and Ronny Krashinsky. NVIDIA A100 Tensor Core GPU: Performance and Innovation. *IEEE Micro*, 41(2):29–35, March 2021. doi: 10.1109/mm.2021.3061394. URL https://doi.org/10.1109%2Fmm.2021.3061394. 9

[50] Michael P. Connolly, Nicholas J. Higham, and Theo Mary. Stochastic rounding and its probabilistic backward error analysis. *SIAM Journal on Scientific Computing*, 43 (1):A566–A585, January 2021. doi: 10.1137/20m1334796. URL https://doi.org/10.1137/20m1334796. 14

[51] Jack Copeland. *Alan Turing's Automatic Computing Engine*. 2005. URL https://oxford.universitypressscholarship.com/view/10.1093/acprof:oso/9780198565932.001.0001/acprof-9780198565932. 40

[52] Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. Training deep neural networks with low precision multiplications. 2014. doi: 10.48550/ARXIV.1412.7024. URL https://arxiv.org/abs/1412.7024. 10

[53] Timothy A. Davis. Algorithm 832: UMFPACK V4.3—an Unsymmetric-Pattern Multifrontal Method$^2$. *ACM Transactions on Mathematical Software*, 30(2):196–199, June 2004. ISSN 0098-3500. doi: 10.1145/992200.992206. URL https://doi.org/10.1145/992200.992206. 26

[54] Timothy A. Davis. *Direct Methods for Sparse Linear Systems*. Society for Industrial and Applied Mathematics, January 2006. doi: 10.1137/1.9780898718881. URL https://doi.org/10.1137%2F1.9780898718881. 21

[55] Timothy A. Davis and Yifan Hu. The university of Florida sparse matrix collection. *ACM Transactions on Mathematical Software*, 38(1):1–25, November 2011. doi: 10.1145/2049662.2049663. URL https://doi.org/10.1145/2049662.2049663. 100, 122, 168

[56] Jeffrey Dean. The Deep Learning Revolution and Its Implications for Computer Architecture and Chip Design. 2019. doi: 10.48550/ARXIV.1911.05289. URL https://arxiv.org/abs/1911.05289. 10

[57] James W. Demmel. The probability that a numerical analysis problem is difficult. *Mathematics of Computation*, 50(182):449–480, 1988. ISSN 00255718, 10886842. URL http://www.jstor.org/stable/2008617. 84

[58] James W. Demmel. *Applied Numerical Linear Algebra*. Society for Industrial and Applied Mathematics, January 1997. doi: 10.1137/1.9781611971446. URL https://doi.org/10.1137/1.9781611971446. 46, 56

[59] James W. Demmel, Stanley C. Eisenstat, John R. Gilbert, Xiaoye S. Li, and Joseph W. H. Liu. A supernodal approach to sparse partial pivoting. *SIAM Journal on Matrix Analysis and Applications*, 20(3):720–755, 1999. doi: 10.1137/S0895479895291765. URL https://doi.org/10.1137/S0895479895291765. 46, 120

[60] James W. Demmel, Yozo Hida, William Kahan, Xiaoye S. Li, Sonil Mukherjee, and Jason Riedy. Error bounds from extra-precise iterative refinement. *ACM Transactions on Mathematical Software*, 32(2):325–351, June 2006. doi: 10.1145/1141885.1141894. URL https://doi.org/10.1145/1141885.1141894. 50, 56, 75, 76

[61] James W. Demmel, Yozo Hida, Jason Riedy, and Xiaoye S. Li. Extra-Precise Iterative Refinement for Overdetermined Least Squares Problems. *ACM Transactions on Mathematical Software*, 35(4):1–32, February 2009. doi: 10.1145/1462173.1462177. URL https://doi.org/10.1145/1462173.1462177. 50, 57

[62] ARM Developer. Half-precision floating-point number format. URL https://developer.arm.com/documentation/100067/0607/Other-Compiler-specific-Features/Half-precision-floating-point-number-format. [Online; accessed 7-July-2022]. 9

[63] Victorita Dolean, Pierre Jolivet, and Frédéric Nataf. *An Introduction to Domain Decomposition Methods*. Society for Industrial and Applied Mathematics, November 2015. doi: 10.1137/1.9781611974065. URL https://doi.org/10.1137%2F1.9781611974065. 2

[64] Jack Dongarra. Preface: Basic Linear Algebra Subprograms Technical (Blast) Forum Standard. *The International Journal of High Performance Computing Applications*, 16(1):1–1, 2002. doi: 10.1177/10943420020160010101. URL https://doi.org/10.1177/10943420020160010101. 49

[65] Jack Dongarra, Cleve B. Moler, and James H. Wilkinson. Improving the Accuracy of Computed Eigenvalues and Eigenvectors. *SIAM Journal on Numerical Analysis*, 20 (1):23–45, February 1983. doi: 10.1137/0720002. URL https://doi.org/10.1137/0720002. 44, 56

[66] Jack Dongarra, Victor Eijkhout, and Piotr Łuszczek. Recursive Approach in Sparse Matrix LU Factorization. *Scientific Programming*, 9(1):51–60, 2001. doi: 10.1155/2001/569670. URL https://doi.org/10.1155/2001/569670. 44, 56, 112

[67] Craig C. Douglas, Jan Mandel, and Willard L. Miranker. Fast Hybrid Solution of Algebraic Systems. *SIAM Journal on Scientific and Statistical Computing*, 11(6):1073–1086, November 1990. doi: 10.1137/0911060. URL https://doi.org/10.1137%2F0911060. 46, 56, 57

[68] Jitka Drkošová, Anne Greenbaum, Miroslav Rozložník, and Zdeněk Strakoš. Numerical stability of GMRES. *BIT Numerical Mathematics*, 35(3):309–330, September 1995. doi: 10.1007/bf01732607. URL https://doi.org/10.1007/bf01732607. 33

[69] Iain S. Duff and John K. Reid. The Multifrontal Solution of Indefinite Sparse Symmetric Linear. *ACM Transactions on Mathematical Software*, 9(3):302–325, September 1983. doi: 10.1145/356044.356047. URL https://doi.org/10.1145%2F356044.356047. 21

[70] Iain S. Duff and John K. Reid. The Multifrontal Solution of Unsymmetric Sets of Linear Equations. *SIAM Journal on Scientific and Statistical Computing*, 5(3):633–641, September 1984. doi: 10.1137/0905045. URL https://doi.org/10.1137%2F0905045. 21

[71] Iain S. Duff, Ronan Guivarch, Daniel Ruiz, and Mohamed Zenadi. The Augmented Block Cimmino Distributed Method. *SIAM Journal on Scientific Computing*, 37(3): A1248–A1269, January 2015. doi: 10.1137/140961444. URL https://doi.org/10.1137%2F140961444. 2

[72] Iain S. Duff, Albert M. Erisman, and John K. Reid. *Direct Methods for Sparse Matrices*. Oxford University Press, 01 2017. ISBN 9780198508380. doi: 10.1093/acprof:oso/9780198508380.001.0001. URL https://doi.org/10.1093/acprof:oso/9780198508380.001.0001. 21, 26, 46, 120

[73] Goran Flegar, Hartwig Anzt, Terry Cojean, and Enrique S. Quintana-Ortí. Adaptive precision block-jacobi for high performance preconditioning in the ginkgo linear algebra software. *ACM Transactions on Mathematical Software*, 47(2), April 2021. ISSN 0098-3500. doi: 10.1145/3441850. URL https://doi.org/10.1145/3441850. 127, 178

[74] R. Fletcher. On the Iterative Refinement of Least Squares Solutions. *Journal of the American Statistical Association*, 70(349):109–112, March 1975. doi: 10.1080/01621459.1975.10480270. URL https://doi.org/10.1080/01621459.1975.10480270. 43, 57

[75] R. Fletcher. Conjugate gradient methods for indefinite systems. In *Lecture Notes in Mathematics*, pages 73–89. Springer Berlin Heidelberg, 1976. doi: 10.1007/bfb0080116. URL https://doi.org/10.1007%2Fbfb0080116. 37

[76] Leslie Fox, Harry D. Huskey, and James H. Wilkinson. NOTES ON THE SOLUTION OF ALGEBRAIC LINEAR SIMULTANEOUS EQUATIONS. *The Quarterly Journal of Mechanics and Applied Mathematics*, 1(1):149–173, January 1948. doi: 10.1093/qjmam/1.1.149. URL https://doi.org/10.1093/qjmam/1.1.149. 41, 56

[77] Melina A. Freitag, Patrick Kürschner, and Jennifer Pestana. GMRES Convergence Bounds for Eigenvalue Problems. *Computational Methods in Applied Mathematics*, 18(2):203–222, June 2017. doi: 10.1515/cmam-2017-0017. URL https://doi.org/10.1515%2Fcmam-2017-0017. 93

[78] Keith O. Geddes and Wei Wei Zheng. Exploiting fast hardware floating point in high precision computation. In *Proceedings of the 2003 international symposium on Symbolic and algebraic computation - ISSAC '03*. ACM Press, 2003. doi: 10.1145/860854.860886. URL https://doi.org/10.1145/860854.860886. 48, 56

[79] Alan George. Nested dissection of a regular finite element mesh. *SIAM Journal on Numerical Analysis*, 10(2):345–363, 1973. doi: 10.1137/0710032. URL https://doi.org/10.1137/0710032. 23, 25

[80] Pieter Ghysels, Xiaoye S. Li, François-Henry Rouet, Samuel Williams, and Artem Napov. An Efficient Multicore Implementation of a Novel HSS-Structured Multifrontal Solver Using Randomized Sampling. *SIAM Journal on Scientific Computing*, 38(5):S358–S384, January 2016. doi: 10.1137/15m1010117. URL https://doi.org/10.1137%2F15m1010117. 27

[81] Philip E. Gill, Michael A. Saunders, and Joseph R. Shinnerl. On the Stability of Cholesky Factorization for Symmetric Quasidefinite Systems. *SIAM Journal on Matrix Analysis and Applications*, 17(1):35–46, January 1996. doi: 10.1137/s0895479893252623. URL https://doi.org/10.1137/s0895479893252623. 44, 56, 112

[82] Luc Giraud, Julien Langou, Miroslav Rozložník, and Jasper van den Eshof. Rounding error analysis of the classical Gram-Schmidt orthogonalization process. *Numerische Mathematik*, 101(1):87–100, May 2005. doi: 10.1007/s00211-005-0615-4. URL https://doi.org/10.1007/s00211-005-0615-4. 32

[83] Stefan L. Glimberg, Allan P. Engsig-Karup, and Morten G. Madsen. A Fast GPU-Accelerated Mixed-Precision Strategy for Fully Nonlinear Water Wave Computations.

In *Numerical Mathematics and Advanced Applications 2011*, pages 645–652. Springer Berlin Heidelberg, November 2012. doi: 10.1007/978-3-642-33134-3_68. URL https://doi.org/10.1007/978-3-642-33134-3_68. 48, 49, 57

[84] GNU. libquadmath. URL https://gcc.gnu.org/onlinedocs/libquadmath/#toc-GNU-Free-Documentation-License-1. 8, 49

[85] Fritz Göbel, Thomas Grützmacher, Tobias Ribizel, and Hartwig Anzt. Mixed precision incomplete and factorized sparse approximate inverse preconditioning on GPUs. In *Euro-Par 2021: Parallel Processing*, pages 550–564. Springer International Publishing, 2021. doi: 10.1007/978-3-030-85665-6_34. URL https://doi.org/10.1007/978-3-030-85665-6_34. 12

[86] Dominik Goddeke and Robert Strzodka. Cyclic Reduction Tridiagonal Solvers on GPUs Applied to Mixed-Precision Multigrid. *IEEE Transactions on Parallel and Distributed Systems*, 22(1):22–32, January 2011. doi: 10.1109/tpds.2010.61. URL https://doi.org/10.1109/tpds.2010.61. 48, 49, 57

[87] Dominik Göddeke, Robert Strzodka, and Stefan Turek. Performance and accuracy of hardware-oriented native-, emulated- and mixed-precision solvers in FEM simulations. *International Journal of Parallel, Emergent and Distributed Systems*, 22(4):221–256, August 2007. doi: 10.1080/17445760601122076. URL https://doi.org/10.1080/17445760601122076. 2, 48, 57, 66, 149

[88] Gene H. Golub. Numerical methods for solving linear least squares problems. *Numerische Mathematik*, 7(3):206–216, June 1965. doi: 10.1007/bf01436075. URL https://doi.org/10.1007/bf01436075. 43, 56, 57, 71

[89] Gene H. Golub. Matrix decompositions and statistical calculations. In *Statistical Computation*, pages 365–397. Elsevier, 1969. doi: 10.1016/b978-0-12-498150-8.50021-5. URL https://doi.org/10.1016/b978-0-12-498150-8.50021-5. 43, 56, 57

[90] Gene H. Golub and Charles F. van Loan. *Matrix Computations*. JHU Press, third edition, 1996. URL http://www.cs.cornell.edu/cv/GVL4/golubandvanloan.htm. 45, 46, 56

[91] Gene H. Golub and James H. Wilkinson. Note on the iterative refinement of least squares solution. *Numerische Mathematik*, 9(2):139–148, December 1966. doi: 10.1007/bf02166032. URL https://doi.org/10.1007/bf02166032. 42, 43, 57, 71

[92] Willy Govaerts. *Numerical Methods for Bifurcations of Dynamical Equilibria*. Society for Industrial and Applied Mathematics, January 2000. doi: 10.1137/1.9780898719543. URL https://doi.org/10.1137/1.9780898719543. 44

[93] Willy Govaerts and John D. Pryce. Block elimination with one refinement solves bordered linear systems accurately. *BIT*, 30(3):490–507, September 1990. doi: 10.1007/bf01931663. URL https://doi.org/10.1007/bf01931663. 44, 56

[94] Stef Graillat, Fabienne Jézéquel, Théo Mary, and Roméo Molina. Adaptive precision matrix-vector product. working paper or preprint, February 2022. URL https://hal.archives-ouvertes.fr/hal-03561193. 150

[95] Serge Gratton, Ehouarn Simon, David Titley-Peloquin, and Philippe Toint. Exploiting variable precision in GMRES. 2019. doi: 10.48550/ARXIV.1907.10550. URL https://arxiv.org/abs/1907.10550. 4, 12, 149, 151, 152, 153

[96] 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. doi: 10.1137/S0895479894275030. URL https://doi.org/10.1137/S0895479894275030. 95, 156

[97] Mårten Gulliksson. Iterative refinement for constrained and weighted linear least squares. *BIT*, 34(2):239–253, June 1994. doi: 10.1007/bf01955871. URL https://doi.org/10.1007/bf01955871. 47, 57

[98] Anshul Gupta. A Shared- and distributed-memory parallel general sparse direct solver. *Applicable Algebra in Engineering, Communication and Computing*, 18(3):263–277, March 2007. doi: 10.1007/s00200-007-0037-x. URL https://doi.org/10.1007%2Fs00200-007-0037-x. 26

[99] Dominik Göddeke, Robert Strzodka, and Stefan Turek. Accelerating double precision fem simulations with gpus. 10 2005. 49, 57

[100] 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*, ScalA '17, New York, NY, USA, 2017. Association for Computing Machinery. ISBN 9781450351256. doi: 10.1145/3148226.3148237. URL https://doi.org/10.1145/3148226.3148237. 52, 56, 57

[101] Azzam Haidar, Ahmad Abdelfattah, Mawussi Zounon, Panruo Wu, Srikara Pranesh, Stanimire Tomov, and Jack Dongarra. The Design of Fast and Energy-Efficient Linear Solvers: On the Potential of Half-Precision Arithmetic and Iterative Refinement Techniques. In *Lecture Notes in Computer Science*, pages 586–600. Springer International Publishing, 2018. doi: 10.1007/978-3-319-93698-7_45. URL https://doi.org/10.1007/978-3-319-93698-7_45. 3, 52, 56, 57, 70, 80, 112

[102] Azzam Haidar, Stanimire Tomov, Jack Dongarra, and Nicholas 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*. IEEE, November 2018. doi: 10.1109/sc.2018.00050. URL https://doi.org/10.1109/sc.2018.00050. 3, 12, 52, 56, 57, 70, 80, 91, 112

[103] Azzam Haidar, Harun Bayraktar, Stanimire Tomov, Jack Dongarra, and Nicholas J. Higham. Mixed-precision iterative refinement using tensor cores on GPUs to accelerate solution of linear systems. *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 476(2243):20200110, November 2020. doi: 10.1098/rspa.2020.0110. URL https://doi.org/10.1098/rspa.2020.0110. 3, 52, 56, 57, 70, 80, 91, 93, 112, 149

[104] Harold V. Henderson and Shayle R. Searle. On Deriving the Inverse of a Sum of Matrices. *SIAM Review*, 23(1):53–60, January 1981. doi: 10.1137/1023004. URL https://doi.org/10.1137%2F1023004. 87, 157

[105] John L Hennessy and David A Patterson. *Computer architecture: a quantitative approach*. Elsevier, 2011. 1

[106] Pascal Hénon, Pierre Ramet, and Jean Roman. PaStiX: a high-performance parallel direct solver for sparse symmetric positive definite systems. *Parallel Computing*, 28(2):301–321, February 2002. doi: 10.1016/s0167-8191(01)00141-7. URL https://doi.org/10.1016%2Fs0167-8191%2801%2900141-7. 26

[107] Magnus R. Hestenes and Eduard Stiefel. Methods of conjugate gradients for solving linear systems. *Journal of Research of the National Bureau of Standards*, 49(6):409, December 1952. doi: 10.6028/jres.049.044. URL https://doi.org/10.6028%2Fjres.049.044. 37

[108] Desmond J. Higham, Nicholas J. Higham, and Srikara Pranesh. Random matrices generating large growth in LU factorization with pivoting. 42(1):185–201, 2021. doi: 10.1137/20M1338149. 97

[109] Nicholas J. Higham. Fast Solution of Vandermonde-Like Systems Involving Orthogonal Polynomials. *IMA Journal of Numerical Analysis*, 8(4):473–486, 1988. doi: 10.1093/imanum/8.4.473. URL https://doi.org/10.1093/imanum/8.4.473. 44, 56

[110] Nicholas J. Higham. Stability Analysis of Algorithms for Solving Confluent Vandermonde-Like Systems. *SIAM Journal on Matrix Analysis and Applications*, 11 (1):23–41, January 1990. doi: 10.1137/0611002. URL https://doi.org/10.1137/0611002. 44, 56

[111] Nicholas J. Higham. Iterative refinement enhances the stability of QR factorization methods for solving linear equations. *BIT*, 31(3):447–468, September 1991. doi: 10.1007/bf01933262. URL https://doi.org/10.1007/bf01933262. 45, 56, 57

[112] Nicholas J. Higham. Iterative refinement for linear systems and LAPACK. *IMA Journal of Numerical Analysis*, 17(4):495–509, October 1997. doi: 10.1093/imanum/17.4.495. URL https://doi.org/10.1093/imanum/17.4.495. 45, 51, 56, 57

[113] Nicholas J. Higham. *Accuracy and Stability of Numerical Algorithms*. Society for Industrial and Applied Mathematics, second edition, 2002. doi: 10.

1137/1.9780898718027. URL https://epubs.siam.org/doi/abs/10.1137/1.9780898718027. 7, 15, 17, 18, 19, 20, 21, 46, 49, 51, 56, 57, 157

[114] Nicholas J. Higham. How fast is quadruple precision arithmetic? https://nhigham.com/2017/08/31/how-fast-is-quadruple-precision-arithmetic/, June 2016. 8

[115] Nicholas J. Higham. Error analysis for standard and GMRES-based iterative refinement in two and three-precisions. Technical Report 2019.19, November 2019. URL http://eprints.maths.manchester.ac.uk/2735/. 88

[116] Nicholas J. Higham and Theo Mary. A new preconditioner that exploits low-rank approximations to factorization error. *SIAM Journal on Scientific Computing*, 41 (1):A59–A82, 2019. doi: 10.1137/18M1182802. URL https://doi.org/10.1137/18M1182802. 14

[117] Nicholas J. Higham and Theo Mary. Sharper probabilistic backward error analysis for basic linear algebra kernels with random data. *SIAM Journal on Scientific Computing*, 42(5):A3427–A3446, 2020. doi: 10.1137/20M1314355. URL https://doi.org/10.1137/20M1314355. 14

[118] Nicholas J. Higham and Theo Mary. Solving block low-rank linear systems by LU factorization is numerically stable. *IMA Journal of Numerical Analysis*, 42(2):951–980, April 2021. doi: 10.1093/imanum/drab020. URL https://doi.org/10.1093/imanum/drab020. 28, 29, 119

[119] Nicholas J Higham and Théo Mary. Mixed precision algorithms in numerical linear algebra. working paper or preprint, January 2022. URL https://hal.archives-ouvertes.fr/hal-03537373. 12, 53

[120] 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, January 2021. doi: 10.1137/19m1298263. URL https://doi.org/10.1137/19m1298263. 52, 56, 57, 70, 80

[121] Nicholas J. Higham, Srikara Pranesh, and Mawussi Zounon. Squeezing a Matrix into Half Precision, with an Application to Solving Linear Systems. *SIAM Journal on Scientific Computing*, 41(4):A2536–A2551, January 2019. doi: 10.1137/18m1229511. URL https://doi.org/10.1137/18m1229511. 11, 76, 100

[122] HITACHI. HD61810 Digital Signal Processor Users Manual. URL https://archive.org/details/bitsavers_hitachidatlSignalProcessorUsersManual_4735688/page/n1/mode/2up. [Online; accessed 7-July-2022]. 8

[123] 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*, 37(2), April 2010. ISSN 0098-3500. doi: 10.1145/1731022.1731027. URL https://doi.org/10.1145/1731022.1731027. 48, 56, 112, 150, 152

[124] HPL-AI. HPL-AI Mixed-Precision Benchmark. URL https://hpl-ai.org/. 52

[125] Pascal Hénon, Pierre Ramet, and Jean Roman. A mapping and scheduling algorithm for parallel sparse fan-in numerical factorization. pages 1059–1067, 08 1999. doi: 10.1007/3-540-48311-X_148. 46, 120

[126] IBM. IBM advances against x86 with Power9. URL https://www.hpcwire.com/2016/08/30/ibm-unveils-power9-details/. 8

[127] IEEE Computer Society. IEEE Standard for Floating-Point Arithmetic. *IEEE Std 754-2019 (Revision of IEEE 754-2008)*, pages 1–84, 2019. doi: 10.1109/IEEESTD.2019.8766229. 7, 8, 41

[128] Ilse Ipsen, Petros Drineas, and Christos Boutsikas. An attempt at explaining why matrices can be better conditioned in lower precision. Talk given at workshop "Advances in Numerical Linear Algebra: Celebrating the 60th Birthday of Nick Higham", Manchester. UK, July 2022. URL https://www.youtube.com/watch?v=unyuQXi0qlo. 85

[129] Michal Jankowski and Henryk Woźniakowski. Iterative refinement implies numerical stability. *BIT*, 17(3):303–311, September 1977. doi: 10.1007/bf01932150. URL https://doi.org/10.1007/bf01932150. 44, 45, 46, 56, 57, 62

[130] Michal Jankowski and Henryk Woźniakowski. The accurate solution of certain continuous problems using only single precision arithmetic. *BIT*, 25(4):635–651, December 1985. doi: 10.1007/bf01936142. URL https://doi.org/10.1007/bf01936142. 44, 57

[131] William Kahan. Writeups of library tape subroutines LEQU, LEQUN, FLEQU, CLEQU and DLEQU. *Institute of Computer Science, University of Toronto*, 1965. 42, 56

[132] George Karypis. METIS – *A Software Package for Partitioning Unstructured Graphs, Partitioning Meshes, and Computing Fill-Reducing Orderings of Sparse Matrices – Version 5.1.0*. University of Minnesota, March 2013. 121

[133] George Karypis and Vipin Kumar. METIS—A Software Package for Partitioning Unstructured Graphs, Partitioning Meshes and Computing Fill-Reducing Ordering of Sparse Matrices. 01 1997. 23

[134] Andrzej Kiełbasiński. Iterative refinement for linear systems in variable-precision arithmetic. *BIT*, 21(1):97–103, March 1981. doi: 10.1007/bf01934074. URL https://doi.org/10.1007/bf01934074. 46, 53, 56, 61, 80

[135] Jakub Kurzak and Jack Dongarra. Implementation of mixed precision in solving systems of linear equations on the Cell processor. *Concurrency and Computation: Practice and Experience*, 19(10):1371–1385, 2007. doi: 10.1002/cpe.1164. URL https://doi.org/10.1002/cpe.1164. 49, 56

[136] 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 *ACM/IEEE SC 2006 Conference (SC'06)*. IEEE, November 2006. doi: 10.1109/sc.2006.30. URL https://doi.org/10.1109/sc.2006.30. 2, 11, 47, 48, 51, 56, 66

[137] Jun Kyu Lee and Gregory D. Peterson. Iterative Refinement on FPGAs. In *2011 Symposium on Application Accelerators in High-Performance Computing*. IEEE, July 2011. doi: 10.1109/saahpc.2011.19. URL https://doi.org/10.1109/saahpc.2011.19. 49, 56

[138] JunKyu Lee, Hans Vandierendonck, Mahwish Arif, Gregory D. Peterson, and Dimitrios S. Nikolopoulos. Energy-Efficient Iterative Refinement Using Dynamic Precision. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, 8(4):722–735, December 2018. doi: 10.1109/jetcas.2018.2850665. URL https://doi.org/10.1109/jetcas.2018.2850665. 53, 56, 57

[139] JunKyu Lee, Gregory D. Peterson, Dimitrios S. Nikolopoulos, and Hans Vandierendonck. AIR: Iterative refinement acceleration using arbitrary dynamic precision. *Parallel Computing*, 97:102663, September 2020. doi: 10.1016/j.parco.2020.102663. URL https://doi.org/10.1016/j.parco.2020.102663. 53, 56, 80

[140] Jean-Yves L'Excellent. *Multifrontal Methods: Parallelism, Memory Usage and Numerical Aspects*. Habilitation à diriger des recherches, Ecole normale supérieure de lyon - ENS LYON, September 2012. URL https://tel.archives-ouvertes.fr/tel-00737751. 21

[141] Jean-Yves L'Excellent and Wissam M. Sid-Lakhdar. A study of shared-memory parallelism in a multifrontal solver. *Parallel Computing*, 40(3-4):34–46, March 2014. doi: 10.1016/j.parco.2014.02.003. URL https://doi.org/10.1016%2Fj.parco.2014.02.003. 121, 144

[142] Xiaoye S. Li and James W. Demmel. Making Sparse Gaussian Elimination Scalable by Static Pivoting. In *Proceedings of the IEEE/ACM SC98 Conference*. IEEE, 1998. doi: 10.1109/sc.1998.10030. URL https://doi.org/10.1109%2Fsc.1998.10030. 26, 29, 44, 56, 112

[143] Xiaoye S. Li and James W. Demmel. SuperLU_DIST. *ACM Transactions on Mathematical Software*, 29(2):110–140, June 2003. doi: 10.1145/779359.779361. URL https://doi.org/10.1145%2F779359.779361. 26

[144] 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, Brandon J. Thompson, Teresa Tung, and Daniel J. Yoo. Design, Implementation and Testing of Extended and Mixed Precision BLAS. *ACM Transactions on Mathematical Software*, 28(2):152–205, June 2002. ISSN 0098-3500. doi: 10.1145/567806.567808. URL https://doi.org/10.1145/567806.567808. 8, 49

[145] Cedric Lichtenau, Steven Carlough, and Silvia Melitta Mueller. Quad Precision Floating Point on the IBM z13. In *2016 IEEE 23nd Symposium on Computer Arithmetic (ARITH)*. IEEE, July 2016. doi: 10.1109/arith.2016.26. URL https://doi.org/10.1109/arith.2016.26. 8

[146] Neil Lindquist, Piotr Luszczek, and Jack Dongarra. Improving the Performance of the GMRES Method Using Mixed-Precision Techniques. In *Communications in Computer and Information Science*, pages 51–66. Springer International Publishing, 2020. doi: 10.1007/978-3-030-63393-6_4. URL https://doi.org/10.1007/978-3-030-63393-6_4. 53, 57, 150, 152

[147] 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, April 2022. doi: 10.1109/tpds.2021.3090757. URL https://doi.org/10.1109/tpds.2021.3090757. 53, 57, 70, 150, 152

[148] Joseph W. H. Liu. Modification of the minimum-degree algorithm by multiple elimination. *ACM Transactions on Mathematical Software*, 11(2):141–153, June 1985. doi: 10.1145/214392.214398. URL https://doi.org/10.1145%2F214392.214398. 23

[149] Joseph W. H. Liu. The Multifrontal Method for Sparse Matrix Solution: Theory and Practice. *SIAM Review*, 34(1):82–109, March 1992. doi: 10.1137/1034004. URL https://doi.org/10.1137%2F1034004. 21

[150] Jennifer A. Loe, Christian A. Glusa, Ichitaro Yamazaki, Erik G. Boman, and Sivasankaran Rajamanickam. Experimental Evaluation of Multiprecision Strategies for GMRES on GPUs. 2021. doi: 10.48550/ARXIV.2105.07544. URL https://arxiv.org/abs/2105.07544. 53, 57, 150, 151, 152, 153

[151] Jennifer A. Loe, Christian A. Glusa, Ichitaro Yamazaki, Erik G. Boman, and Sivasankaran Rajamanickam. A Study of Mixed Precision Strategies for GMRES on GPUs. 2021. doi: 10.48550/ARXIV.2109.01232. URL https://arxiv.org/abs/2109.01232. 53, 57, 70, 150, 151, 152

[152] Florent Lopez and Théo Mary. Mixed Precision LU Factorization on GPU Tensor Cores: Reducing Data Movement and Memory Footprint. September 2020. URL https://hal.archives-ouvertes.fr/hal-02937325. working paper or preprint. 12, 52

[153] Florent Lopez and Theo Mary. Mixed Precision LU Factorization on GPU Tensor Cores: Reducing Data Movement and Memory Footprint, 2021. URL http://eprints.maths.manchester.ac.uk/2782/. MIMS EPrint 2020.20, Manchester Institute for Mathematical Sciences, The University of Manchester, UK, September 2020. 112

[154] NAG Ltd. NAG Fortran Library Manual, 2005. 50

[155] Magma. Matrix algebra on GPU and multicore architectures (MAGMA). http://icl.cs.utk.edu/magma/. 52, 80

[156] Rawlyn R. M. Mallock. An electrical calculating machine. *Proceedings of the Royal Society of London. Series A, Containing Papers of a Mathematical and Physical Character*, 140(841):457–483, May 1933. doi: 10.1098/rspa.1933.0081. URL https://doi.org/10.1098/rspa.1933.0081. 40

[157] Roger S. Martin, G. Peters, and James H. Wilkinson. Iterative refinement of the solution of a positive definite system of equations. In *Linear Algebra*, pages 31–44. Springer Berlin Heidelberg, 1971. doi: 10.1007/978-3-662-39778-7_2. URL https://doi.org/10.1007/978-3-662-39778-7_2. 42, 56

[158] Théo Mary. *Block low-rank multifrontal solvers : complexity, performance, and scalability*. Theses, Université Paul Sabatier - Toulouse III, November 2017. URL https://tel.archives-ouvertes.fr/tel-01929478. 21, 27

[159] Stephen F. McCormick, Joseph Benzaken, and Rasmus Tamstorf. Algebraic Error Analysis for Mixed-Precision Multigrid Solvers. *SIAM Journal on Scientific Computing*, 43(5):S392–S419, January 2021. doi: 10.1137/20m1348571. URL https://doi.org/10.1137/20m1348571. 53, 57, 149

[160] Cleve B. Moler. SOLVE, Accurate simultaneous linear equation solver with iterative improvement. *SHARE Distribution No. 3194*, 1964. 42, 56

[161] Cleve B. Moler. Iterative refinement in floating point. *Journal of the ACM*, 14(2):316–321, April 1967. doi: 10.1145/321386.321394. URL https://doi.org/10.1145/321386.321394. 42, 56

[162] Netlib. LAPACK. URL https://www.netlib.org/lapack/. 46

[163] Esmond G. Ng and Padma Raghavan. Performance of Greedy Ordering Heuristics for Sparse Cholesky Factorization. *SIAM Journal on Matrix Analysis and Applications*, 20(4):902–914, January 1999. doi: 10.1137/s0895479897319313. URL https://doi.org/10.1137%2Fs0895479897319313. 23

[164] Steven A. Niederer, Eric Kerfoot, Alan P. Benson, Miguel O. Bernabeu, Olivier Bernus, Chris Bradley, Elizabeth M. Cherry, Richard Clayton, Flavio H. Fenton, Alan Garny, Elvio Heidenreich, Sander Land, Mary Maleckar, Pras Pathmanathan, Gernot Plank, José F. Rodríguez, Ishani Roy, Frank B. Sachse, Gunnar Seemann, Ola Skavhaug, and Nic P. Smith. Verification of cardiac tissue electrophysiology simulators using an $N$-version benchmark. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 369(1954):4331–4351, 2011. doi: 10.1098/rsta.2011.0139. URL https://royalsocietypublishing.org/doi/abs/10.1098/rsta.2011.0139. 122

[165] Thomas Norrie, Nishant Patil, Doe Hyun Yoon, George Kurian, Sheng Li, James Laudon, Cliff Young, Norman Jouppi, and David Patterson. The Design Process for

Google's Training Chips: TPUv2 and TPUv3. *IEEE Micro*, 41(2):56–63, March 2021. doi: 10.1109/mm.2021.3058217. URL https://doi.org/10.1109/mm.2021.3058217. 9

[166] NVIDIA. NVIDIA Hopper Architecture, . URL https://developer.nvidia.com/blog/nvidia-hopper-architecture-in-depth/. 9

[167] NVIDIA. tensorfloat-32, . URL https://blogs.nvidia.com/blog/2020/05/14/tensorfloat-32-precision-format/. 9

[168] NVIDIA, 2019. URL https://docs.nvidia.com/cuda/cusolver/. 52, 80

[169] Takeshi Ogita. Accurate matrix factorization: Inverse LU and inverse QR factorizations. 31(5):2477–2497, 2010. doi: 10.1137/090754376. 99

[170] Eda Oktay and Erin Carson. Multistage mixed precision iterative refinement. *Numerical Linear Algebra with Applications*, February 2022. doi: 10.1002/nla.2434. URL https://doi.org/10.1002/nla.2434. 53, 56, 57, 61, 65, 105, 151, 152

[171] Eda Oktay and Erin Carson. Mixed Precision GMRES-based Iterative Refinement with Recycling. 2022. doi: 10.48550/ARXIV.2201.09827. URL https://arxiv.org/abs/2201.09827. 52, 56, 57, 93, 151, 152, 153

[172] Kyaw L. Oo and Andreas Vogel. Accelerating Geometric Multigrid Preconditioning with Half-Precision Arithmetic on GPUs. 2020. doi: 10.48550/ARXIV.2007.07539. URL https://arxiv.org/abs/2007.07539. 53, 57, 149

[173] Christopher C. Paige and Michael A. Saunders. Solution of sparse indefinite systems of linear equations. *SIAM Journal on Numerical Analysis*, 12(4):617–629, September 1975. doi: 10.1137/0712047. URL https://doi.org/10.1137%2F0712047. 37

[174] Christopher C. Paige and Michael A. Saunders. LSQR: An algorithm for sparse linear equations and sparse least squares. *ACM Transactions on Mathematical Software*, 8(1):43–71, March 1982. doi: 10.1145/355984.355989. URL https://doi.org/10.1145/355984.355989. 38

[175] 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, January 2006. doi: 10.1137/050630416. URL https://doi.org/10.1137/050630416. 33, 69, 82, 83, 84, 110, 155, 205, 206, 207, 208, 209, 210, 211, 212

[176] Grégoire Pichon, Eric Darve, Mathieu Faverge, Pierre Ramet, and Jean Roman. Sparse supernodal solver using block low-rank compression: Design, performance and analysis. *Journal of Computational Science*, 27:255–270, July 2018. doi: 10.1016/j.jocs.2018.06.007. URL https://doi.org/10.1016%2Fj.jocs.2018.06.007. 27

[177] Gregorio Quintana-Ortí, Enrique S. Quintana-Ortí, Robert A. Van De Geijn, Field G. Van Zee, and Ernie Chan. Programming matrix algorithms-by-blocks for thread-level parallelism. *ACM Transactions on Mathematical Software*, 36(3):1–26, July

2009. doi: 10.1145/1527286.1527288. URL https://doi.org/10.1145%2F1527286.1527288. 17

[178] J. L. Rigal and J. Gaches. On the compatibility of a given solution with the data of a linear system. *Journal of the ACM*, 14(3):543–548, July 1967. doi: 10.1145/321406.321416. URL https://doi.org/10.1145%2F321406.321416. 13

[179] Edward Rothberg and Stanley C. Eisenstat. Node Selection Strategies for Bottom-Up Sparse Matrix Ordering. *SIAM Journal on Matrix Analysis and Applications*, 19(3):682–695, July 1998. doi: 10.1137/s0895479896302692. URL https://doi.org/10.1137%2Fs0895479896302692. 23

[180] Siegfried M Rump. Inversion of extremely ill-conditioned matrices in floating-point. *Japan Journal of Industrial and Applied Mathematics*, 26(2-3):249–277, 2009. 99

[181] Youcef Saad. Krylov subspace methods for solving large unsymmetric linear systems. *Mathematics of Computation*, 37(155):105–126, 1981. doi: 10.1090/s0025-5718-1981-0616364-6. URL https://doi.org/10.1090/s0025-5718-1981-0616364-6. 37

[182] Youcef Saad. A flexible inner-outer preconditioned GMRES algorithm. *SIAM Journal on Scientific Computing*, 14(2):461–469, March 1993. doi: 10.1137/0914028. URL https://doi.org/10.1137/0914028. 35

[183] Youcef Saad and Martin H. Schultz. GMRES: A Generalized Minimal Residual Algorithm for Solving Nonsymmetric Linear Systems. *SIAM Journal on Scientific and Statistical Computing*, 7(3):856–869, July 1986. doi: 10.1137/0907058. URL https://doi.org/10.1137%2F0907058. 30

[184] Yousef Saad. ILUT: A dual threshold incomplete LU factorization. *Numerical Linear Algebra with Applications*, 1(4):387–402, July 1994. doi: 10.1002/nla.1680010405. URL https://doi.org/10.1002%2Fnla.1680010405. 119

[185] Olaf Schenk, Klaus Gärtner, and Wolfgang Fichtner. Efficient Sparse LU Factorization with Left-Right Looking Strategy on Shared Memory Multiprocessors. *Bit Numerical Mathematics*, 40(1):158–176, 2000. doi: 10.1023/a:1022326604210. URL https://doi.org/10.1023/a:1022326604210. 26, 120

[186] Robert Schreiber. A New Implementation of Sparse Gaussian Elimination. *ACM Transactions on Mathematical Software*, 8(3):256–276, September 1982. doi: 10.1145/356004.356006. URL https://doi.org/10.1145%2F356004.356006. 21, 22

[187] Daniil V. Shantsev, Piyoosh Jaysaval, Sébastien de la Kethulle de Ryhove, Patrick R. Amestoy, Alfredo Buttari, Jean-Yves L'Excellent, and Theo Mary. Large-scale 3-D EM modelling with a Block Low-Rank multifrontal direct solver. *Geophysical Journal International*, 209(3):1558–1571, March 2017. doi: 10.1093/gji/ggx106. URL https://doi.org/10.1093%2Fgji%2Fggx106. 27

[188] Valeria Simoncini and Daniel B. Szyld. Recent computational developments in krylov subspace methods for linear systems. *Numerical Linear Algebra with Applications*, 14 (1):1–59, 2007. doi: https://doi.org/10.1002/nla.499. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/nla.499. 95

[189] Robert D. Skeel. Scaling for Numerical Stability in Gaussian Elimination. *Journal of the ACM*, 26(3):494–526, July 1979. doi: 10.1145/322139.322148. URL https://doi.org/10.1145/322139.322148. 44

[190] Robert D. Skeel. Iterative refinement implies numerical stability for gaussian elimination. *Mathematics of Computation*, 35(151):817–832, 1980. doi: 10.1090/s0025-5718-1980-0572859-4. URL https://doi.org/10.1090/s0025-5718-1980-0572859-4. 44, 45, 56

[191] Alicja Smoktunowicz and Jolanta Sokolnicka. Binary cascades iterative refinement in doubled-mantissa arithmetics. *BIT*, 24(1):123–127, March 1984. doi: 10.1007/bf01934524. URL https://doi.org/10.1007/bf01934524. 46, 53, 57

[192] Alicja Smoktunowicz and Jolanta Sokolnicka. Solving the linear least squares problem with very high relative accuracy. *Computing*, 45(4):345–354, December 1990. doi: 10.1007/bf02238802. URL https://doi.org/10.1007/bf02238802. 46, 57

[193] James N. Snyder. On the improvement of the solutions to a set of simultaneous linear equations using the ILLIAC. *Mathematical Tables and Other Aids to Computation*, 9(52):177, October 1955. doi: 10.2307/2002054. URL https://doi.org/10.2307/2002054. 42, 56

[194] Peter Sonneveld. CGS, a fast lanczos-type solver for nonsymmetric linear systems. *SIAM Journal on Scientific and Statistical Computing*, 10(1):36–52, January 1989. doi: 10.1137/0910004. URL https://doi.org/10.1137%2F0910004. 37

[195] Gilbert W. Stewart. *Introduction to matrix computations*. Computer science and applied mathematics. Academic Press, New York, 1973. ISBN 0126703507. 43, 56, 57

[196] Rita Streich, Christoph Schwarzbach, Michael Becken, and Klaus Spitzer. Controlled-source Electromagnetic Modelling Studies – Utility of Auxiliary Potentials for Low-frequency Stabilization. *Conference Proceedings, 72nd EAGE Conference*, (cp-161-00065), 2010. ISSN 2214-4609. doi: https://doi.org/10.3997/2214-4609.201400657. URL https://www.earthdoc.org/content/papers/10.3997/2214-4609.201400657. 123

[197] Robert Strzodka and Dominik Göddeke. Mixed precision methods for convergent iterative schemes. pages D–59–60, May 2006. URL http://asc.ziti.uni-heidelberg.de/sites/default/files/research/papers/public/StGo06convIter.pdf. 48, 57, 66

[198] Robert Strzodka and Dominik Göddeke. Pipelined Mixed Precision Algorithms on FPGAs for Fast and Accurate PDE Solvers from Low Precision Components. In *2006*

*14th Annual IEEE Symposium on Field-Programmable Custom Computing Machines*. IEEE, April 2006. doi: 10.1109/fccm.2006.57. URL https://doi.org/10.1109/fccm.2006.57. 48, 49, 57, 150, 152

[199] Yuki Sumiyoshi, Akihiro Fujii, Akira Nukada, and Teruo Tanaka. Mixed-Precision AMG Method for Many Core Accelerators. In *Proceedings of the 21st European MPI Users' Group Meeting*, EuroMPI/ASIA '14, page 127–132, New York, NY, USA, 2014. Association for Computing Machinery. ISBN 9781450328753. doi: 10.1145/2642769.2642794. URL https://doi.org/10.1145/2642769.2642794. 48, 49, 57

[200] Junqing Sun, Gregory D. Peterson, and Olaf O. Storaasli. High-Performance Mixed-Precision Linear Solver for FPGAs. *IEEE Transactions on Computers*, 57(12):1614–1623, December 2008. doi: 10.1109/tc.2008.89. URL https://doi.org/10.1109%2Ftc.2008.89. 49, 56

[201] Rasmus Tamstorf, Joseph Benzaken, and Stephen F. McCormick. Discretization-Error-Accurate Mixed-Precision Multigrid Solvers. *SIAM Journal on Scientific Computing*, 43(5):S420–S447, January 2021. doi: 10.1137/20m1349230. URL https://doi.org/10.1137/20m1349230. 53, 57

[202] Pascal Theissen, Kirstin Heuler, Rainer Demuth, Johannes Wojciak, Thomas Indinger, and Nikolaus Adams. Experimental Investigation of Unsteady Vehicle Aerodynamics under Time-Dependent Flow Conditions - Part 1. In *SAE 2011 World Congress & Exhibition*. SAE International, April 2011. doi: https://doi.org/10.4271/2011-01-0177. URL https://doi.org/10.4271/2011-01-0177. 123

[203] 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, January 2001. doi: 10.1137/s0895479899359837. URL https://doi.org/10.1137/s0895479899359837. 50, 57

[204] D. Titley-Peloquin, J. Pestana, and A. J. Wathen. GMRES convergence bounds that depend on the right-hand-side vector. *IMA Journal of Numerical Analysis*, 34(2):462–479, July 2013. doi: 10.1093/imanum/drt025. URL https://doi.org/10.1093%2Fimanum%2Fdrt025. 93

[205] Kathryn Turner and Homer F. Walker. Efficient High Accuracy Solutions with GMRES(m). *SIAM Journal on Scientific and Statistical Computing*, 13(3):815–825, May 1992. doi: 10.1137/0913048. URL https://doi.org/10.1137/0913048. 2, 4, 46, 48, 53, 57, 70, 149, 150, 152

[206] J. Uhlig. C. Forsythe and C. B. Moler, Computer Solution of Linear Algebraic Systems. (Series in Automatic Computation) XI + 148 S. Englewood Cliffs, N.J. 1967. Prentice-Hall, Inc. Preis geb. 54 s. net. *ZAMM - Zeitschrift für Angewandte Mathematik und Mechanik*, 52(6):378–379, 1972. doi: 10.1002/zamm.19720520620. URL https://doi.org/10.1002/zamm.19720520620. 56

[207] Henk A. van der Vorst. Bi-CGSTAB: A fast and smoothly converging variant of bi-CG for the solution of nonsymmetric linear systems. *SIAM Journal on Scientific and Statistical Computing*, 13(2):631–644, March 1992. doi: 10.1137/0913035. URL https://doi.org/10.1137%2F0913035. 37

[208] John von Neumann and Herman H. Goldstine. Numerical inverting of matrices of high order. *Bulletin of the American Mathematical Society*, 53(11):1021–1099, 1947. doi: 10.1090/s0002-9904-1947-08909-6. URL https://doi.org/10.1090/s0002-9904-1947-08909-6. 40, 41

[209] Homer F. Walker. Implementation of the GMRES Method Using Householder Transformations. *SIAM Journal on Scientific and Statistical Computing*, 9(1):152–163, January 1988. doi: 10.1137/0909010. URL https://doi.org/10.1137/0909010. 33

[210] Wikipedia. GeForce FX series — Wikipedia, The Free Encyclopedia, . 8

[211] Wikipedia. IBM 700/7000 series — Wikipedia, The Free Encyclopedia, . 40

[212] Wikipedia. English Electric KDF9 — Wikipedia, The Free Encyclopedia, . 42

[213] James H. Wilkinson. Progress report on the Automatic Computing Engine. Report MA/17/1024, Mathematics Division, Department of Scientific and Industrial Research, National Physical Laboratory, April 1948. URL http://www.alanturing.net/turing_archive/archive/l/l10/l10.php. 2, 40, 41, 56

[214] James H. Wilkinson. *Rounding Errors in Algebraic Processes*. Notes on Applied Science No. 32, Her Majesty's Stationery Office, London, 1963. ISBN 0-486-67999-3. Also published by Prentice-Hall, Englewood Cliffs, NJ, USA. Reprinted by Dover, New York, 1994. 41, 42, 56

[215] Ichitaro Yamazaki, Stanimire Tomov, Tingxing Dong, and Jack Dongarra. Mixed-precision orthogonalization scheme and adaptive step size for improving the stability and performance of CA-GMRES on GPUs. In *Lecture Notes in Computer Science*, pages 17–30. Springer International Publishing, 2015. doi: 10.1007/978-3-319-17353-5_2. URL https://doi.org/10.1007/978-3-319-17353-5_2. 150

[216] Mihalis Yannakakis. Computing the Minimum Fill-In is NP-Complete. *SIAM Journal on Algebraic Discrete Methods*, 2(1):77–79, March 1981. doi: 10.1137/0602010. URL https://doi.org/10.1137%2F0602010. 23

[217] Zahari Zlatev. Use of Iterative Refinement in the Solution of Sparse Linear Systems. *SIAM Journal on Numerical Analysis*, 19(2):381–399, April 1982. doi: 10.1137/0719024. URL https://doi.org/10.1137/0719024. 44, 45, 56, 112, 119

[218] Mawussi Zounon, Nicholas J. Higham, Craig Lucas, and Françoise Tisseur. Performance impact of precision reduction in sparse linear systems solvers. *PeerJ Computer Science*, 8:e778, January 2022. doi: 10.7717/peerj-cs.778. URL https://doi.org/10.7717/peerj-cs.778. 52, 56, 121

# Appendix

This annex is devoted to provide the full details for the proof of Theorem 5.2 in chapter 5.

To prove Theorem 5.2, we essentially need to adapt the analysis of Paige et al. [175, sect. 8], the results from the previous sections of this article being almost unchanged. In particular, [175, sect. 3] reviews the MGS algorithm, [175, sect. 4] makes the link between the MGS and the Arnoldi algorithm, [175, sect. 5] analyzes the impact of the loss of orthogonality in MGS which has strong consequences on the analysis, [175, sect. 6] exhibits the existence of a key iteration which allows proving the convergence, and [175, sect. 7] provides stability results for the solution of the LS problem with MGS. Note that the original notation of Paige et al. [175] has been slightly adapted to be consistent with the notation of this manuscript inherited from Carson and Higham [44; 45].

The proof is essentially composed of three steps: the inclusion of the arbitrary matrix–vector product (5.5) in [175, eq. (4.3)] and the following consequences on, first, the stability result of the MGS-GMRES least squares solutions and, second, the stability result of MGS-GMRES for the solution of the linear system (5.4).

*Arbitrary matrix–vector product.* In our version of MGS-GMRES we consider a product with $B$ satisfying (5.5). We now show that considering (5.5) with $\epsilon_p \neq \gamma_n^g$ mainly changes [175, eq. (4.3)]. Let us consider $\widehat{V}_k = [\widehat{v}_1, \ldots, \widehat{v}_k] \in \mathbb{R}^{n\times k}$, the matrix of computed basis vectors, and $\dot{V}_k = [\dot{v}_1, \ldots, \dot{v}_k]$ the same matrix but with its columns correctly normalized; that is, for $j \leq k$,

$$\widehat{v}_j = \dot{v}_j + \Delta v_j^{(1)}, \quad \|\Delta v_j^{(1)}\|_2 \leq \widetilde{\gamma}_n^g, \tag{1a}$$

$$\widehat{V}_k = \dot{V}_k + \Delta V_k^{(1)}, \quad \Delta V_k^{(1)} = [\Delta v_1^{(1)}, \ldots, \Delta v_k^{(1)}], \tag{1b}$$

where $\Delta v_j^{(1)}$ is the error for the normalization of $\widehat{v}_j$ and $\Delta V_k^{(1)}$ is the accumulated error for the normalization of the basis at step $k$. By (5.5) and (1), we obtain

$$\mathrm{fl}(P\,\widehat{v}_j) = P(\dot{v}_j + \Delta v_j^{(1)}) + f_j \tag{2a}$$

$$= P\,\dot{v}_j + \Delta v_j^{(2)}, \tag{2b}$$

where $\Delta v_j^{(2)} = P\Delta v_j^{(1)} + f_j$ satisfies $\|\Delta v_j^{(2)}\|_2 \lesssim (\epsilon_p + \widetilde{\gamma}_n^g)\|P\|_F$ since $\|\dot{v}_j\|_2 = 1$ and $\|f_j\|_2 \lesssim \epsilon_p\|P\|_F\|\dot{v}_j + \Delta v_j^{(1)}\|_2$. We therefore obtain

$$\mathrm{fl}(P\widehat{V}_k) = P\dot{V}_k + \Delta V_k^{(2)}, \quad \|\Delta V_k^{(2)}\|_F \lesssim k^{1/2}(\epsilon_p + \widetilde{\gamma}_n^g)\|P\|_F, \tag{3}$$

where $\Delta V_k^{(2)}$ contains the error for both the product and the normalization at the kth iteration. (3) gives a new [175, 4.3].

*Useful results from the original analysis.* We now provide few results from Paige et al. [175] that we need for the rest of the proof. We first recall the theorem of [175, sect. 2].

**Theorem 8.1** ([175, Thm. 2.4])**.** *Let $B \in \mathbb{R}^{n\times k}$ have rank $s$ and singular values $\sigma_1 \geq \cdots \geq \sigma_s > 0$. For $c \in \mathbb{R}^n$ and scalar $\phi \geq 0$ define $\widehat{y} \equiv B^\dagger c$, $\widehat{r} \equiv c - B\widehat{y}$, $\sigma(\phi) \equiv \sigma_{s+1}([c\,\phi, B])$ and $\delta(\phi) \equiv \sigma(\phi)/\sigma_s$. If $\widehat{r}\phi \neq 0$ then $\sigma(\phi) > 0$, and if $\phi_0 \equiv \sigma_s/\|c\|$ then $\delta(\phi) < 1\ \forall \phi \in [0, \phi_0)$,*

$$\sigma^2(\phi)(\phi^{-2} + \|\widehat{y}\|_2^2) \leq \|\widehat{r}\|_2^2 \leq \sigma^2(\phi)(\phi^{-2} + \|\widehat{y}\|_2^2/[1-\sigma^2(\phi)]), \quad \forall \phi > 0 \text{ s.t. } \delta(\phi) < 1. \tag{4}$$

We summarize the results [175, Eqs. (6.1)–(6.3)] in the following. We note $\widehat{B}_m = [q, \mathrm{fl}(P\widehat{V}_m)]$, Paige et al. [175] proved that if

$$\bar{m} \text{ is the first integer such that } \kappa_2(\dot{V}_{\bar{m}}) > 4/3 \tag{5}$$

we obtain the following relations

$$\bar{m}\widetilde{\gamma}_n\kappa_F(\widehat{B}_{\bar{m}}) > 1.8, \text{ so } \sigma_{\min}(\widehat{B}_{\bar{m}}D) < 8\bar{m}\widetilde{\gamma}_n\|\widehat{B}_{\bar{m}}D\|_F\ \forall \text{ diagonal } D > 0, \tag{6}$$

and

$$\kappa_2(\dot{V}_j),\ \sigma_{\min}^{-1}(\dot{V}_j),\ \sigma_{\max}(\dot{V}_j) \leq 4/3, \quad j = 1, \ldots, \bar{m}-1. \tag{7}$$

*MGS-GMRES least squares solution.* In [175, sect. 7], MGS was already proven backward stable for solving the problem

$$\arg\min_y \|q - Cy\|_2, \quad r(y) \equiv q - Cy, \quad C \in R^{n\times(m-1)}. \tag{8}$$

We need to extend this result for $C = \mathrm{fl}(P\widehat{V}_k) = P\dot{V}_k + \Delta V_k^{(2)}$. Let $k$ be the kth step of MGS-GMRES such that $k < \bar{m}$ with $\bar{m}$ satisfying (5). Let $\widehat{B}_{k+1} \equiv [q, P\dot{V}_k + \Delta V_k^{(2)}]$ and $\Delta C_k(y)$ and $\Delta q_k(y)$ be the errors on the MGS least squares solution [175, Eq. (7.13)]. Then [175, eq. (8.1)] becomes

$$\widehat{y}_k = \arg\min_y \|r_k(y)\|_2, \quad r_k(y) \equiv q + \Delta q_k(y) - [P\dot{V}_k + \Delta V_k^{(2)} + \Delta C_k(y)]y, \tag{9}$$

where

$$\|[\Delta q_k(y), \Delta C_k(y)]e_j\|_2 \leq \widetilde{\gamma}_{kn}^g\|\hat{B}_{k+1}e_j\|_2, \quad j = 1, \ldots, k+1; \tag{10a}$$

$$\|\Delta V_k^{(2)}\|_F \lesssim k^{1/2}(\epsilon_p + \tilde{\gamma}_n^g)\|P\|_F, \quad \|\Delta q_k(y)\|_2 \le \tilde{\gamma}_{kn}^g \|q\|_2, \tag{10b}$$

$$\|\Delta V_k^{(2)} + \Delta C_k(y)\|_F \lesssim (k^{1/2}\epsilon_p + \tilde{\gamma}_{kn}^g)\|P\|_F. \tag{10c}$$

The last bound (10c) is computed as follow, as the first bound on $\|[\Delta q_k(y), \Delta C_k(y)]e_j\|_2$ can be extended to the Frobenius norm considering $\|\Delta C_k(y)\|_F^2 = \sum_j \|\Delta C_k(y)e_j\|_2^2$, then we can write

$$\|\Delta C_k(y)\|_F \le \tilde{\gamma}_{kn}^g \|P\dot{V}_k + \Delta V_k^{(2)}\|_F \le \tilde{\gamma}_{kn}^g (\|P\dot{V}_k\|_F + \|\Delta V_k^{(2)}\|_F) \tag{11a}$$

$$\approx \tilde{\gamma}_{kn}^g \|P\dot{V}_k\|_F \le \tilde{\gamma}_{kn}^g \|P\|_F \|\dot{V}_k\|_2 \quad (1) \tag{11b}$$

$$\le 4/3 \times \tilde{\gamma}_{kn}^g \|P\|_F. \quad (2) \tag{11c}$$

(1) since $\|AB\|_F \le \|A\|_2\|B\|_F$, $\|H\|_F = \|H^T\|_F$ and $\|H\|_2 = \|H^T\|_2$.

(2) since $\|\dot{V}_k\|_2 = \sigma_{\max}(\dot{V}_k) \le 4/3$ from (7).

Thus,

$$\|\Delta V_k^{(2)} + \Delta C_k(y)\|_F \lesssim (k^{1/2}\epsilon_p + \tilde{\gamma}_{k^{1/2}n}^g)\|P\|_F + \tilde{\gamma}_{kn}^g \|P\|_F \tag{12a}$$

$$\lesssim (k^{1/2}\epsilon_p + \tilde{\gamma}_{kn}^g)\|P\|_F. \tag{12b}$$

It can then be concluded from (9) that the least squares solution $\hat{y}_k$ is backward stable for

$$\min_y \|q - P\dot{V}_k y\|_2. \tag{13}$$

*MGS-GMRES for the solution of the linear system.* Our goal is now to prove that at a certain step $k$ the computed $\widehat{y}_k$ also provides a backward stable solution $\widehat{x}_k = \mathrm{fl}(\widehat{V}_k \widehat{y}_k)$ to the system $Px = q$. This part is challenging because of the loss of orthogonality of the basis $V_k$. To do so, we show that, at the $(\bar{m}-1)$th iteration, MGS-GMRES has computed a backward stable solution of the system. From now, we set $k$ such that $k \equiv \bar{m}-1 \le n$, we rewrite [175, eq. (8.2)] as

$$r_k(\widehat{y}_k) = q_k - P_k\widehat{y}_k, \quad q_k \equiv q + \Delta q_k(\widehat{y}_k), \quad P_k \equiv P\dot{V}_k + \Delta V_k^{(3)}(\widehat{y}_k), \tag{14a}$$

$$\|\Delta q_k(\widehat{y}_k)\|_2 \le \tilde{\gamma}_{kn}^g \|q\|_2, \quad \Delta V_k^{(3)}(y) \equiv \Delta V_k^{(2)} + \Delta C_k(y), \tag{14b}$$

$$\|\Delta V_k^{(3)}(y)\|_F \lesssim (k^{1/2}\epsilon_p + \tilde{\gamma}_{kn}^g)\|P\|_F, \tag{14c}$$

where $\Delta V_k^{(3)}(y)$ contains the errors on the product, on the normalization, and on the computation of the least square solution via MGS. The main idea of what follows is to use the scaling invariance of MGS to scale the right-hand side $q$ by some scalar $\phi$ and use Theorem 8.1 to bound the residual. We write $D \equiv \mathrm{diag}(\phi, I_k)$ for any scalar $\phi > 0$. Since $\widehat{B}_{k+1} = [q, P\dot{V}_k + \Delta V_k^{(2)}]$, from (14) we have

$$[q_k\phi, P_k] = \widehat{B}_{k+1}D + \Delta B_k D, \quad \Delta B_k \equiv [\Delta q_k(\widehat{y}_k), \Delta C_k(\widehat{y}_k)]. \tag{15}$$

We now bound the quantities $\|\Delta B_k D\|_F$, $|\widehat{B}_{k+1} D\|_F$ and $\|q_k\|_2$. We begin with $\|\Delta B_k D\|_F$,

$$\|\Delta B_k D\|_F \leq \tilde{\gamma}^g_{kn} \|\widehat{B}_{k+1} D\|_F = \tilde{\gamma}^g_{kn} \|[q_k \phi, P_k] - \Delta B_k D\|_F \quad (1) \tag{16a}$$

$$\leq \tilde{\gamma}^g_{kn} (\|[q_k \phi, P_k]\|_F + \|\Delta B_k D\|_F) \tag{16b}$$

$$\leq \tilde{\gamma}^g_{kn} (\|[q_k \phi, P_k]\|_F + \tilde{\gamma}^g_{kn} \|\widehat{B}_{k+1} D\|_F). \tag{16c}$$

(1) coming from (10a) $\|\Delta B_k e_j\|_2 \leq \tilde{\gamma}^g_{kn} \|\widehat{B}_{k+1} e_j\|_2$.
So we have $\tilde{\gamma}^g_{kn}(1-\tilde{\gamma}^g_{kn}) \|\widehat{B}_{k+1} D\|_F \leq \tilde{\gamma}^g_{kn} \|[q_k \phi, P_k]\|_F$, giving

$$\|\Delta B_k D\|_F \leq \tilde{\gamma}^g_{kn} \|\widehat{B}_{k+1} D\|_F \leq \tilde{\gamma}^{g\prime}_{kn} \|[q_k \phi, P_k]\|_F, \quad \tilde{\gamma}^{g\prime}_{kn} = \frac{\tilde{\gamma}^g_{kn}}{1-\tilde{\gamma}^g_{kn}}, \tag{17}$$

assuming $\tilde{\gamma}^g_{kn} \ll 1$. In the same fashion we can also write

$$\|\widehat{B}_{k+1} D\|_F = \|[q_k \phi, P_k] - \Delta B_k D\|_F \leq \|[q_k \phi, P_k]\|_F + \tilde{\gamma}^g_{kn} \|\widehat{B}_{k+1} D\|_F, \tag{18}$$

it follows

$$\|\widehat{B}_{k+1} D\|_F \leq \frac{1}{1-\tilde{\gamma}^g_{kn}} \|[q_k \phi, P_k]\|_F. \tag{19}$$

In addition, as $q_k = q + \Delta q_k(\widehat{y}_k)$ and $\|\Delta q_k(\widehat{y}_k)\|_2 \leq \tilde{\gamma}^g_{kn} \|q\|_2$ we can state

$$\|q_k\|_2 \leq (1 + \tilde{\gamma}^g_{kn}) \|q\|_2. \tag{20}$$

(15), (17), (19), and (20) combined corresponds to [175, eq. (8.3)]. In addition, according to (5), $k+1 = \bar{m} \leq n+1$ is the first integer such that $\kappa_2(\dot{V}_{k+1}) > 4/3$. We obtain from (6)

$$\sigma_{\min}(\widehat{B}_{k+1} D) < 8(k+1) \tilde{\gamma}^g_n \|\widehat{B}_{k+1} D\|_F \leq \tilde{\gamma}^g_{kn} \|[q_k \phi, P_k]\|_F, \tag{21a}$$

$$\|P_k\|_F \lesssim \|P \dot{V}_k\|_F + (k^{1/2} \epsilon_p + \tilde{\gamma}^g_{kn}) \|P\|_F \leq (4/3 + k^{1/2} \epsilon_p + \tilde{\gamma}^g_{kn}) \|P\|_F. \quad (1) \tag{21b}$$

(1) $\|P \dot{V}_k\|_F$ can be bounded by same strategy as in (11).
The bound (21) is true starting from a certain iteration $k$ which depends on the problem and the constants; in particular, it carries the loss of orthogonality analysed in [175, sect. 5 and 6]. It generalizes [175, eq. (8.4)]. We now bound $\sigma_{\min}(P_k)$ and $\sigma_{\min}([q_k \phi, P_k])$. We know by the two-norm definition that $\min_{\|x\|_2=1} \|P x\|_2 = \sigma_{\min}(P)$. Therefore, we can write

$$\sigma_{\min}(P_k) = \min_{\|x\|_2=1} \|P_k x\|_2 \geq \min_{\|x\|_2=1} \|P \dot{V}_k x\|_2 - \min_{\|x\|_2=1} \|\Delta V^{(3)}_k(\widehat{y}_k) x\|_2 \tag{22a}$$

$$\geq \sigma_{\min}(P \dot{V}_k) - \|\Delta V^{(3)}_k(\widehat{y}_k)\|_2 \gtrsim 3\sigma_{\min}(P)/4 - (k^{1/2} \epsilon_p + \tilde{\gamma}^g_{kn}) \|P\|_F, \quad (1) \tag{22b}$$

(1) since $\sigma_{\min}(AB) \geq \sigma_{\min}(A) \sigma_{\min}(B)$ and $\|\cdot\|_2 \leq \|\cdot\|_F$.
and

$$\sigma_{\min}([q_k \phi, P_k]) = \min_{\|x\|_2=1} \|[q_k \phi, P_k] x\|_2 \leq \sigma_{\min}(\widehat{B}_{k+1} D) + \|\Delta B_k D\|_2 \quad (1) \tag{23a}$$

$$\leq \tilde{\gamma}^g_{kn} \|[q_k \phi, P_k]\|_F. \quad (2) \tag{23b}$$

(1) from (15) and using the fact that $\min(f(x)+g(x)) \leq \min f(x) + \max g(x)$

(2) from (21) and (17)

The bounds (22) and (23) can be used to analyze an important scalar of Theorem 8.1,

$$\delta_k(\phi) \equiv \frac{\sigma_{\min}([q_k\phi, P_k])}{\sigma_{\min}(P_k)}. \tag{24}$$

In particular, Theorem 8.1 provides an upper bound on the residual norm $\|r_k(\widehat{y}_k)\|_2^2$ expressed with $\phi$ and $\delta_k(\phi)$. A natural way of using it would be to minimize this upper bound according to $\phi$. However, Paige et al. [175] argued it would be unnecessarily complicated, they postulated it is sufficient to show that it exists a value $\phi'$ of $\phi$ satisfying (25) below.

$$\phi' > 0, \quad \sigma_{\min}^2(A_k) - \sigma_{\min}^2([b_k\phi', A_k]) = \sigma_{\min}^2(A_k)\|\hat{y}_k\phi'\|_2^2. \tag{25}$$

The existence of such a $\phi'$ is obtain as follows. Writing LHS $\equiv \sigma_{\min}^2(P_k) - \sigma_{\min}^2([q_k\phi, P_k])$ and RHS $\equiv \sigma_{\min}^2(P_k)\|\widehat{y}_k\phi\|_2^2$ we want to find $\phi$ so that LHS = RHS. For $\phi = 0 \Rightarrow$ LHS > RHS, and $\phi = \|\widehat{y}_k\|_2^{-1} \Rightarrow$ LHS < RHS, so from continuity, necessarily $\exists\phi' \in (0, \|\widehat{y}_k\|_2^{-1})$ satisfying (25). Consequently, using the definitions (24) and (25), we have

$$\delta_k(\phi') < 1, \quad (\phi')^{-2} = \frac{\|\widehat{y}_k\|_2^2}{1-\delta_k(\phi')^2}, \quad 0 < \phi' < \|\widehat{y}_k\|_2^{-1}, \tag{26}$$

and from Theorem 8.1 we can bound the residual by

$$\begin{aligned}
\|r_k(\widehat{y}_k)\|_2^2 &\leq \sigma_{\min}^2([q_k\phi', P_k])((\phi')^{-2} + \|\widehat{y}_k\|_2^2/[1-\delta_k(\phi')^2]) && (27a)\\
&= \sigma_{\min}^2([q_k\phi', P_k])(2(\phi')^{-2}) \quad (1) && (27b)\\
&\leq (\widetilde{\gamma}_{kn}^g)^2(\|q_k\phi'\|_2^2 + \|P_k\|_F^2)2(\phi')^{-2}. \quad (2) && (27c)
\end{aligned}$$

(1) from (26).

(2) by using (23) and since $\|[q_k\phi', P_k]\|_F^2 = \|q_k\phi'\|_2^2 + \|P_k\|_F^2$.

The previous bound (27) corresponds to [175, eq. (8.9)]. Now that we have a bound on the residual, we are looking for expressing it in terms of $\|P\|_F$, $\|q\|_2$ and $\|\widehat{y}_k\|_2$. From (21) we already have a bound on $\|P_k\|_F$, but we still need to bound $\phi'$ and $\|q_k\phi'\|_2$. Let us consider first $\|q_k\phi'\|_2$, we have

$$\begin{aligned}
\|q_k\phi'\|_2^2 &\leq \|r_k(\widehat{y}_k)\phi'\|_2^2 + \|P_k\widehat{y}_k\phi'\|_2^2 \quad (1) && (28a)\\
&= (\phi')^2\|r_k(\widehat{y}_k)\|_2^2 + (\phi')^2\|P_k\widehat{y}_k\|_2^2 && (28b)\\
&\leq 2(\widetilde{\gamma}_{kn}^g)^2(\|b_k\phi'\|_2^2 + \|P_k\|_F^2) + \|P_k\|_F^2(1-\delta_k(\phi')^2) \quad (2) && (28c)\\
&\leq 2(\widetilde{\gamma}_{kn}^g)^2\|b_k\phi'\|_2^2 + (1+2(\widetilde{\gamma}_{kn}^g)^2)\|P_k\|_F^2. \quad (3) && (28d)
\end{aligned}$$

(1) by simply writing $r_k(\widehat{y}_k) = q_k - P_k\widehat{y}_k$.

(2) from (27) for the bound on $\|r_k(\widehat{y}_k)\|_2^2$ and by using (26) knowing $\|P_k\widehat{y}_k\|_2 \leq \|P_k\|_F\|\widehat{y}_k\|_2$.

(3) by removing the negative term.

Then, it gives

$$\|q_k\phi'\|_2^2 \leq \frac{1+2(\tilde{\gamma}_{kn}^g)^2}{1-2(\tilde{\gamma}_{kn}^g)^2}\|P_k\|_F^2, \tag{29}$$

which corresponds to [175, eq. (8.10)]. Let us now consider $\phi'$, we can bound $\delta_k(\phi')$ to obtain a bound on $\phi'$. We have

$$\begin{aligned}
\delta_k(\phi') &\lesssim \frac{\tilde{\gamma}_{kn}^{g\prime}\|[q_k\phi', P_k]\|_F}{\sigma_{\min}(P)-(\frac{4}{3}k^{1/2}\epsilon_p+\tilde{\gamma}_{kn}^g)\|P\|_F} \quad (1) && \text{(30a)}\\
&= \frac{\tilde{\gamma}_{kn}^{g\prime}(\|q_k\phi'\|_2^2+\|P_k\|_F^2)^{1/2}}{\sigma_{\min}(P)-(\frac{4}{3}k^{1/2}\epsilon_p+\tilde{\gamma}_{kn}^g)\|P\|_F} \leq \frac{\tilde{\gamma}_{kn}^{g\prime}(\frac{1+2(\tilde{\gamma}_{kn}^g)^2}{1-2(\tilde{\gamma}_{kn}^g)^2}\|P_k\|_F^2+\|P_k\|_F^2)^{1/2}}{\sigma_{\min}(P)-(\frac{4}{3}k^{1/2}\epsilon_p+\tilde{\gamma}_{kn}^g)\|P\|_F} \quad (2) && \text{(30b)}\\
&\approx \frac{\tilde{\gamma}_{kn}^{g\prime\prime}\|P_k\|_F}{\sigma_{\min}(P)-(\frac{4}{3}k^{1/2}\epsilon_p+\tilde{\gamma}_{kn}^g)\|P\|_F} \lesssim \frac{\tilde{\gamma}_{kn}^{g\prime\prime}(4/3+k^{1/2}\epsilon_p+\tilde{\gamma}_{kn}^g)\|P\|_F}{\sigma_{\min}(P)-(\frac{4}{3}k^{1/2}\epsilon_p+\tilde{\gamma}_{kn}^g)\|P\|_F} \quad (3) && \text{(30c)}\\
&\approx \frac{\tilde{\gamma}_{kn}^{g\prime\prime\prime}\|P\|_F}{\sigma_{\min}(P)-(\frac{4}{3}k^{1/2}\epsilon_p+\tilde{\gamma}_{kn}^g)\|P\|_F}. && \text{(30d)}
\end{aligned}$$

(1) by using the definition of $\delta_k(\phi')$, the bounds (22) and (23), and noting that the $\tilde{\gamma}$ absorbed the term 4/3.

(2) from (29)

(3) from (21)

Considering the nonsingularity condition (5.6) of our theorem stating $\sigma_{\min}(P) \gg n^2\max(u_g,\epsilon_p)\|P\|_F$, we can surely say that $\sigma_{\min}(P) \gg (2\tilde{\gamma}_{kn}^{g\prime\prime\prime}+\frac{4}{3}k^{1/2}\epsilon_p+\tilde{\gamma}_{kn}^g)\|P\|_F$, which gives

$$\delta_k(\phi') \ll \frac{1}{2}. \tag{31}$$

This combined with (26) gives

$$(\phi')^{-2} \leq 4\|\widehat{y}_k\|_2^2/3. \tag{32}$$

We can now bound the residual (27) using (21), (29) and (32). We have

$$\begin{aligned}
\|r_k(\widehat{y}_k)\|_2^2 &\leq (\tilde{\gamma}_{kn}^g)^2(\|q_k\phi'\|_2^2+\|P_k\|_F^2)2(\phi')^{-2} && \text{(33a)}\\
&\leq (\tilde{\gamma}_{kn}^g)^2(1+\frac{1+2(\tilde{\gamma}_{kn}^g)^2}{1-2(\tilde{\gamma}_{kn}^g)^2})\|P_k\|_F^2 8\|\widehat{y}_k\|_2^2/3 \quad (1) && \text{(33b)}\\
&\lesssim (\tilde{\gamma}_{kn}^g)^2(1+\frac{1+2(\tilde{\gamma}_{kn}^g)^2}{1-2(\tilde{\gamma}_{kn}^g)^2})(4/3+k^{1/2}\epsilon_p+\tilde{\gamma}_{kn}^g)^2\|P\|_F^2 8\|\widehat{y}_k\|_2^2/3 \quad (2) && \text{(33c)}\\
&\lesssim (\tilde{\gamma}_{kn}^{g\prime})^2\|P\|_F^2\|\widehat{y}_k\|_2^2. \quad (3) && \text{(33d)}
\end{aligned}$$

(1) from (29) and (32).

(2) from (21).

(3) removing second order terms.

Thus, it gives

$$\|r_k(\widehat{y}_k)\|_2 \lesssim \tilde{\gamma}_{kn}^g\|P\|_F\|\widehat{y}_k\|_2 \leq \tilde{\gamma}_{kn}^g(\|P\|_F\|\widehat{y}_k\|_2+\|q\|_2), \tag{34}$$

which corresponds to [175, eq. (8.12)]. However the computed solution of the linear system (5.4) is not $\widehat{y}_k$ but is $\widehat{x}_k = \text{fl}(\widehat{V}_k\widehat{y}_k)$, and to complete our proof we have to show that $\widehat{x}_k$ is a backward stable solution of the system (5.4). If $\widehat{V}_k\widehat{y}_k$ is a standard matrix vector product in precision $u_g$ the computed solution verifies $\widehat{x}_k = \text{fl}(\widehat{V}_k\widehat{y}_k) = (\widehat{V}_k + \Delta V_k^y)\widehat{y}_k$, with $|\Delta V_k^y| \leq \gamma_k^g |\widehat{V}_k|$ carrying the error on this product. Then, we can define

$$\Delta P_k \equiv [\Delta V_k^{(3)}(\widehat{y}_k) - P(\Delta V_k^y + \widehat{V}_k - \dot{V}_k)]\widehat{y}_k \frac{\widehat{x}_k^T}{\|\widehat{x}_k\|_2^2}, \tag{35}$$

satisfying $(P + \Delta P_k)\widehat{x}_k = (P\dot{V}_k + \Delta V_k^{(3)}(\widehat{y}_k))\widehat{y}_k$. It can be verified by writing

$$\begin{aligned}(P + \Delta P_k)\widehat{x}_k = P\widehat{x}_k + (P\dot{V}_k + \Delta V_k^{(3)}(\widehat{y}_k))\widehat{y}_k \frac{\widehat{x}_k^T\widehat{x}_k}{\|\widehat{x}_k\|_2^2} \qquad (36)\\ - P(\widehat{V}_k + \Delta V_k^y)\widehat{y}_k \frac{\widehat{x}_k^T\widehat{x}_k}{\|\widehat{x}_k\|_2^2} = (P\dot{V}_k + \Delta V_k^{(3)}(\widehat{y}_k))\widehat{y}_k. \quad (1)\end{aligned}$$

(1) since $\frac{\widehat{x}_k^T\widehat{x}_k}{\|\widehat{x}_k\|_2^2} = 1$ and $(\widehat{V}_k + \Delta V_k^y)\widehat{y}_k = \widehat{x}_k$

We want to bound $\Delta P_k$, we start from the definition (35)

$$\begin{aligned}\|\Delta P_k\|_F &\leq [\|\Delta V_k^{(3)}(\widehat{y}_k)\|_F + \|P(\Delta V_k^y + \widehat{V}_k - \dot{V}_k)\|_F] \frac{\|\widehat{y}_k\|_2\|\widehat{x}_k^T\|_2}{\|\widehat{x}_k\|_2^2} \qquad (37a)\\ &= [\|\Delta V_k^{(3)}(\widehat{y}_k)\|_F + \|P(\Delta V_k^y + \Delta V_k^{(1)})\|_F] \frac{\|\widehat{y}_k\|_2}{\|\widehat{x}_k\|_2}. \quad (1) \qquad (37b)\end{aligned}$$

(1) from (1).

We know that $\|\Delta V_k^y\|_F \leq \gamma_k^g \|\widehat{V}_k\|_F \leq \gamma_k^g[\|\dot{V}_k\|_F + \|\Delta V_k^{(1)}\|_F] \approx \gamma_k^g \|\dot{V}_k\|_F = k^{1/2}\gamma_k^g$ and $\|\Delta V_k^{(1)}\|_F^2 = \sum_j^k \|\Delta v_j^{(1)}\|_2^2 \leq k(\widetilde{\gamma}_n^g)^2$; in addition, we have

$$\begin{aligned}\|\widehat{x}_k\|_2 = \|(\widehat{V}_k + \Delta V_k^y)\widehat{y}_k\|_2 \geq \|\widehat{V}_k\widehat{y}_k\|_2 - \|\Delta V_k^y\|_F\|\widehat{y}_k\|_2 \quad (1) \qquad (38a)\\ \gtrsim (3/4 - k^{1/2}\widetilde{\gamma}_n^g - k^{1/2}\gamma_k^g)\|\widehat{y}_k\|_2 = (3/4 - \widetilde{\gamma}_{k^{1/2}n}^g)\|\widehat{y}_k\|_2. \quad (2) \qquad (38b)\end{aligned}$$

(1) since $\|x + y\|_2 \geq \|x\|_2 - \|y\|_2$.

(2) since $\|\widehat{V}_k\widehat{y}_k\|_2 \geq \|\dot{V}_k\widehat{y}_k\|_2 - \|\Delta V_k^n\widehat{y}_k\|_2 \geq (\min_y \frac{\|\dot{V}_k y\|_2}{\|y\|_2} - \|\Delta V_k^n\|_F)\|\widehat{y}_k\|_2 \geq (\sigma_{\min}(\dot{V}_k) - k^{1/2}\widetilde{\gamma}_n^g)\|\widehat{y}_k\|_2$ and from (7).

These previous bounds on $\|\Delta V_k^y\|_F$, $\|\Delta V_k^{(1)}\|_F$, and $\|\widehat{x}_k\|_2$ with the bound (14) on $\|\Delta V_k^{(3)}(\widehat{y}_k)\|_F$ finally give

$$\|\Delta P_k\|_F \lesssim (k^{1/2}\epsilon_p + \widetilde{\gamma}_{kn}^g + \gamma_{k^{3/2}}^g + \widetilde{\gamma}_{k^{1/2}n}^g)(3/4 - \widetilde{\gamma}_{k^{1/2}n}^g)^{-1}\|P\|_F \approx (k^{1/2}\epsilon_p + \widetilde{\gamma}_{kn}^g)\|P\|_F. \tag{39}$$

As we have $\|\widehat{x}_k\|_2 \leq \|\widehat{V}_k\widehat{y}_k\|_2 + \|\Delta V_k^y\widehat{y}_k\|_2) \lesssim 4/3\|\widehat{y}_k\|_2$ we can write $\|r_k(\widehat{y}_k)\|_2 \lesssim \widetilde{\gamma}_{kn}^g(\|P\|_F\|\widehat{x}_k\|_2 + \|q\|_2)$. Summarizing and using (34) gives

$$\begin{aligned}r_k(\widehat{y}_k) &= q + \Delta q_k(\widehat{y}_k) - (P + \Delta P_k)\widehat{x}_k, \qquad (40a)\\ \|r_k(\widehat{y}_k)\|_2 &\lesssim \widetilde{\gamma}_{kn}^g(\|P\|_F\|\widehat{x}_k\|_2 + \|q\|_2), \qquad (40b)\\ \|\Delta q_k(\widehat{y}_k)\|_2 &\leq \widetilde{\gamma}_{kn}^g\|q\|_2, \qquad (40c)\end{aligned}$$

$$\|\Delta P_k\|_F \lesssim (k^{1/2}\epsilon_p + \tilde{\gamma}^g_{kn})\|P\|_F, \tag{40d}$$

which generalizes [175, eq. (8.15)]. Then the backward stability follows by defining

$$\Delta q'_k \equiv -\frac{\|q\|_2}{\|q\|_2 + \|P\|_F\|\widehat{x}_k\|_2} r_k(\widehat{y}_k), \quad \Delta P'_k \equiv \frac{\|P\|_F\|\widehat{x}_k\|_2}{\|q\|_2 + \|P\|_F\|\widehat{x}_k\|_2} \frac{r_k(\widehat{y}_k)\widehat{x}_k^T}{\|\widehat{x}_k\|_2^2}, \tag{41}$$

satisfying $(P + \Delta P_k + \Delta P'_k)\widehat{x}_k = q + \Delta q_k(\widehat{y}_k) + \Delta q'_k$. We have

$$\|\Delta q'_k\|_2 = \frac{\|q\|_2}{\|q\|_2 + \|P\|_F\|\widehat{x}_k\|_2}\|r_k(\widehat{y}_k)\|_2 \lesssim \tilde{\gamma}^g_{kn}\|q\|_2, \tag{42}$$

and

$$\|\Delta P'_k\|_F \leq \frac{\|P\|_F\|\widehat{x}_k\|_2}{\|q\|_2 + \|P\|_F\|\widehat{x}_k\|_2} \frac{\|r_k(\widehat{y}_k)\|_2\|\widehat{x}_k^T\|_2}{\|\widehat{x}_k\|_2^2} \lesssim \tilde{\gamma}^g_{kn}\|P\|_F, \tag{43}$$

giving finally

$$\|\Delta P_k + \Delta P'_k\|_F \lesssim (k^{1/2}\epsilon_p + \tilde{\gamma}^g_{kn})\|P\|_F, \quad \|\Delta q_k(\widehat{y}_k) + \Delta q'_k\|_2 \lesssim \tilde{\gamma}^g_{kn}\|q\|_2. \tag{44}$$

Consequently, the solution $\widehat{x}_k$ is backward stable for (5.4) relative to $\epsilon_p$ and $u_g$, and (44) is providing bounds on its accuracy.