Iterative methods for solving linear systems on massively parallel architectures

Olivier Tissot

Sorbonne Université

Inria

Doctoral School **Sciences Mathématiques de Paris Centre**

University Department **Laboratoire Jacques-Louis Lions**

Thesis defended by **Olivier Tissot**

Defended on **21st January, 2019**

In order to become Doctor from Sorbonne Université

Academic Field **Applied Mathematics**

Iterative methods for solving linear systems on massively parallel architectures

Thesis supervised by Laura Grigori

Committee members

Referees	Hassane Sadok	Professor at Université du Littoral Côte d'Opale
	Daniel Szyld	Professor at Temple University
Examiners	Yvon Maday	Professor at Sorbonne Université
	Christian Rey	Senior Researcher at Safran
Supervisor	Laura Grigori	Senior Researcher at Inria

Sorbonne Université

Inria

Doctoral School **Sciences Mathématiques de Paris Centre**

University Department **Laboratoire Jacques-Louis Lions**

Thesis defended by **Olivier Tissot**

Defended on **21st January, 2019**

In order to become Doctor from Sorbonne Université

Academic Field **Applied Mathematics**

Iterative methods for solving linear systems on massively parallel architectures

Thesis supervised by Laura Grigori

Committee members

Referees	Hassane Sadok	Professor at Université du Littoral Côte d'Opale
	Daniel Szyld	Professor at Temple University
Examiners	Yvon Maday	Professor at Sorbonne Université
	Christian Rey	Senior Researcher at Safran
Supervisor	Laura Grigori	Senior Researcher at Inria

Sorbonne Université

Inria

École doctorale **Sciences Mathématiques de Paris Centre**

Unité de recherche **Laboratoire Jacques-Louis Lions**

Thèse présentée par **Olivier Tissot**

Soutenue le **21 janvier 2019**

En vue de l'obtention du grade de docteur de Sorbonne Université

Discipline **Mathématiques Appliquées**

Méthodes itératives de résolution de systèmes linéaires sur des architectures parallèles

Thèse dirigée par Laura Grigori

Composition du jury

Rapporteurs	Hassane Sadok	professeur à l'Université du Littoral Côte d'Opale
	Daniel Szyld	professeur à Temple University
Examinateurs	Yvon Maday	professeur à Sorbonne Université
	Christian Rey	directeur de recherche à Safran
Directeur de thèse	Laura Grigori	directrice de recherche à Inria

Keywords: linear solvers, parallel computing, Krylov methods, recycling techniques

Mots clés : solveurs linéaires, calcul parallèle, méthodes de Krylov, techniques de recyclage

This thesis has been prepared at the following research units.

Laboratoire Jacques-Louis Lions
4 place Jussieu
75005 Paris
France

☎ +33 1 44 27 42 98
Web Site http://ljll.math.upmc.fr/

Inria Paris
2 rue Simone Iff
75012 Paris
France

☎ +33 1 80 49 40 00
Web Site http://inria.fr/

Lorsque j'ai un système linéaire à résoudre c'est très simple : j'utilise backslash de MATLAB™. Mais je sais qu'il y en a dans la salle pour qui c'est compliqué.

C. C.

Acknowledgements/Remerciements

Ça y est, je m'attaque aux remerciements. Le reste est fait, je respire depuis un moment. Et, ça n'est pas simple finalement — dire que j'attendais presque cet instant. Déjà, il ne faut oublier personne. Ensuite, il faut satisfaire la curiosité du lecteur, et garder à l'esprit qu'il ne lira très certainement que cette partie. Idéalement, il faut même l'impressionner en faisant de l'esprit, essayer de caler un jeu de mot — une contrepèterie, c'est un peu trop. Enfin, il faut faire relativement court quand même. Pour l'originalité, c'est compliqué. Apparemment, il y a des règles à respecter[1][2][3]. Et visiblement plusieurs acteurs sont déjà présents sur le marché du doctorant–peu–inspiré–au-moment–des–remerciements [4][5]. Non, ça n'est pas simple finalement...

Je ne dérogerai pas à la règle, et je commencerai par exprimer ma profonde gratitude à Laura. Après ces trois années, je réalise à quel point c'est difficile de mettre des mots sur tout ce que je te dois. Ta confiance et ton soutien tout au long de cette thèse ont été déterminants. Merci de m'avoir proposé ce sujet de thèse. Merci de m'avoir toujours laissé une grande liberté pour aborder les problèmes. Merci pour toutes les opportunités dont j'ai eu la chance de profiter pendant ces trois ans. Merci pour toute l'aide dans la recherche d'un post-doc, et pour avoir eu le soucis de l'après–thèse.

Evidemment, je suis très reconnaissant à Hassane Sadok et Daniel Szyld d'avoir accepté d'être rapporteurs de mon manuscrit de thèse. Merci à Hassane Sadok d'avoir très gentiment accepté d'adapter son emploi du temps afin d'assister à la soutenance. Daniel Szyld, ya que los agradecimientos son en francés y la mayor parte del manuscrito en inglés, he pensado que sería simpático de expresarle en español mis más sinceros agradecimientos por las numerosas correcciones del manuscrito — a pesar de mi (lamentable) nivel de español. Je remercie bien évidemment tout particulièrement Yvon Maday et Christian Rey qui ont accepté de faire partie du jury.

I would like to deeply thank all the members of the NLAFET project, especially Bo Kågström the project leader. I acknowledge HPC2N for providing me the access to Kebnekaise and Abisko.

I deeply thank Jan Papež and Radek Stompor who contributed significantly to the fourth chapter. Thank you Jan to have explained me, more than once, the astrophysics application; thank you, above all, for your help and support. I acknowledge NERSC for providing me the access to Cori and Edison.

Ma mémoire est parfois défaillante mais il y a des choses que je ne peux pas oublier. Je n'oublie pas que Frédéric Bonnans a accepté que je finisse mon contrat un peu plus tôt que prévu pour pouvoir commencer ma thèse dans les meilleures conditions. I do not forget that Jim Demmel hosted me in his group during three months at the

[1] Nicolas Curien, "Arbres et cartes aléatoires", HDR. LPMA, Université Pierre et Marie Curie, Paris, 2013. https://www.math.u-psud.fr/~curien/papers/HDR.pdf

[2] Daniel Pennac, "Merci", 2004.

[3] Quentin Verreycken, "Anatomie des remerciements de thèse", in ParenThèses, 2015, https://parenthese.hypotheses.org/1127.

[4] https://www.scribbr.fr/these-doctorat/remerciements-these/

[5] https://www.corep.fr/guide-de-la-these/rediger-these/remerciements-these/

begining of 2016. Je n'oublie pas que Nicole Spillane m'a invité à faire une présentation à DD24. Je n'oublie pas que Iain Duff et Frédéric Nataf ont accepté d'écrire des lettres de recommandation pour moi. Peut-être que pour vous ce n'était pas grand chose, mais pour moi c'était beaucoup : merci à vous.

Pour beaucoup, une thèse c'est–des–hauts–et–des–bas, et je ne fais pas exception. Dans mon cas, ça a aussi été le théâtre de rigolades inoubliables avec Hussam Al-Daas et Sébastien Cayrols. Merci Husssam pour ton sens de l'humour à toute épreuve. Merci Sébastien pour ta gentillesse sans limite. Je ne vous dois pas un merci. Je ne vous dois pas deux mercis. Non, moi je vous dois mille mercis. Votre impact direct et indirect sur mon travail est énorme. En résumé, grâce à vous deux, ma thèse c'est pas la même.

Il y a trois ans, l'équipe Alpines en était à ses balbutiements. Le centre Inria Paris était encore dans les cartons, littéralement. Je remercie sincèrement tous les membres, présents et passés, de l'équipe Alpines qui font qu'elle est toujours une équipe "particulièrement excellente". Je remercie chaleureusement tous les autochtones du 3A, comprendre 3$^{\text{ème}}$ étage du bâtiment A de l'Inria Paris. C'était un vrai plaisir de partager avec vous tous des repas, des cafés, des goûters, des verres, des footings, des baskets, des (baby)foots, et surtout plein de discussions extrêmement enrichissantes; merci pour tout.

Je tiens à remercier la famille Boittin pour son hospitalité. Merci à Martine et Philippe pour les week–ends à Bayeux et tous les très bons restaurants à Paris. Merci à Christine et Dominique, l'antenne Toulouso–Bordelaise, pour le week–end viticole à Bordeaux et le choix toujours cornélien à l'apéro. Merci à Clément pour toutes les anecdotes judiciaires.

Merci à ma famille qui me supporte depuis un certain temps déjà. Mes frères et sœurs Pierre, Laurence, Nathalie et Isabelle qui ont tous essayé de comprendre ce que je faisais. Mes parents m'ont toujours soutenu et, tout en me laissant suivre mon chemin, m'ont donné le goût de la reflexion; merci à vous deux, je me suis souvent dit que j'avais eu bien de la chance de vous avoir. Pendant ces 3 ans, j'ai vu grandir Éloïne, Tristan et Valentin avec un grand bonheur; merci pour tous vos jolis sourires.

Et pour finir, merci à Léa bien sûr. À vrai dire, de simples mots ne peuvent retranscrire tout ce que je te dois. Je ne suis pas Saint–John Perse alors je me contenterai de te dire merci pour les relectures, les corrections, les encouragements, et le reste surtout.

Et puis merci aussi au pied droit de Benjamin Pavard parce qu'on était mal embarqué quand même.

Iterative methods for solving linear systems on massively parallel architectures

Abstract

Krylov methods are widely used for solving large sparse linear systems of equations. On distributed architectures, their performance is limited by the communication needed at each iteration of the algorithm. In this thesis, we first study the use of so-called Enlarged Krylov subspaces for reducing the number of iterations, and therefore the overall communication, of Krylov methods. We consider a reformulation of the Conjugate Gradient (CG) method using these enlarged Krylov subspaces: the Enlarged Conjugate Gradient (ECG) method. This method is first studied from a theoretical point of view. In particular, we show that its convergence speed is close to that of the so-called Deflated Conjugate Gradient method. In order to mitigate the effect of the extra arithmetic operations induced by the method, we explain how to dynamically reduce the number of search directions during the iterations. We then present the parallel design of two variants of the ECG method as well as their corresponding dynamic versions. Using a block Jacobi preconditioner, we show that our implementation scales up to several thousands of cores, and it can be significantly faster than the PETSc implementation of the CG method. We then focus on the Cosmic Microwave Background (CMB) analysis. We investigate the usage of so–called recycling strategies in this context. As a result of the multiplicity of the smallest eigenvalue, these techniques may not improve the convergence in some cases. Hence, we propose a cheap procedure to adapt the initial guess that permits to reduce the overall number of iterations in such situations.

Keywords: linear solvers, parallel computing, Krylov methods, recycling techniques

---

Méthodes itératives de résolution de systèmes linéaires sur des architectures parallèles

Résumé

Les méthodes de Krylov sont largement utilisées pour résoudre des systèmes linéaires creux de grande taille. Sur une architecture distribuée, leur performance est souvent limitée par les communications requises à chaque itération de l'algorithme. Dans cette thèse, nous commençons par étudier l'utilisation des sous–espaces dits de Krylov élargis pour réduire le nombre d'itérations, et ainsi le nombre de communications, des méthodes de Krylov. Nous nous intéressons à une reformulation de la méthode du Gradient Conjugué (CG) qui utilise ces sous–espaces de Krylov élargis : la méthode du Gradient Conjugué Élargi (ECG). Cette méthode est d'abord étudiée d'un point de vue théorique. En particulier, nous montrons que sa vitesse de convergence est proche de celle de la méthode dite du Gradient Conjugué Déflaté. Afin d'atténuer l'effet des opérations arithmétiques supplémentaires requises par la méthode, nous expliquons comment réduire dynamiquement le nombre de directions de recherche pendant les itérations. Nous présentons ensuite le design parallèle des deux variantes de la méthode ECG ainsi que les versions dynamiques qui correspondent. En utilisant un préconditionneur de type bloc Jacobi, nous montrons que notre implémentation est scalable jusqu'à plusieurs milliers de processeurs, et qu'elle peut être significativement plus rapide que l'implémentation de la méthode CG présente dans la librairie PETSc. Nous nous concentrons ensuite sur l'analyse des observations du fond diffus cosmologique. Nous évaluons l'usage des techniques dites de recyclage dans ce contexte. En raison de la multiplicité de la plus petite valeur propre, ces techniques ne permettent pas d'améliorer la convergence dans certains cas. Par conséquent, nous proposons une procédure peu coûteuse pour adapter la solution initiale qui permet de réduire le nombre total d'itérations dans ces situations.

Mots clés : solveurs linéaires, calcul parallèle, méthodes de Krylov, techniques de recyclage

---

Laboratoire Jacques-Louis Lions
4 place Jussieu – 75005 Paris – France

Contents

List of Figures

List of Tables

Introduction (version française)

Sommaire du présent chapitre

Contexte

De nos jours, quasiment tous les ordinateurs, de votre ordinateur portable personnel jusqu'au plus moderne des supercalculateurs en passant par les smartphones, reposent sur le même paradigme : ajouter de plus en plus de processeurs pour calculer de plus en plus vite. Ce paradigme est relativement nouveau : aux alentours de 2005, la fréquence des processeurs a commencé à stagner autour de 3GHz afin de limiter leur consommation d'énergie (Figure 1). En conséquence, le développement de logiciels sur ces architectures parallèle requiert une attention particulière. En effet, sur ces machines parallèles, il est plus coûteux de déplacer des données que de calculer une opération à virgule flottante [42].

Un énorme effort est actuellement déployé afin de développer la première machine exascale, c'est-à-dire capable de calculer 10^{18} opérations à virgule flottante par seconde. Par exemple, en 2015 Barack Obama a signé un décret présidentiel pour créer une "National Strategic Computing Initiative" qui doit découler sur une accélération du développement d'un système exascale. La Chine est censée se doter d'un ordinateur exascale pendant le 13ème Plan Quinquennal (2016–2020). De son côté, la Commission européenne a adopté en 2012 sa stratégie pour le Calcul Haute Performance (HPC en anglais), dont l'un des buts est le développement d'une machine exascale dans les 10 ans. Toutes ces instances ont reconnu qu'il y a un besoin urgent de nouveaux logiciels, et algorithmes, qui seraient efficaces sur ces machines.

Au cœur de cet effort, NLAFET — Algèbre Linéaire Numérique Parallèle pour Systèmes Exascale, est un projet Horizon 2020 FET-HPC financé par l'Union européenne. Son but est de minimiser le fossé entre les capacités maximales théoriques des machines et les performances réalisées en pratiques par les applications HPC qui reposent sur des logiciels d'algèbre linéaire. En pratique, l'objectif est de produire et rendre accessible une libraire, qui serait utilisable comme une boîte noire, pour les parties qui

prennent le plus de temps lors de l'exécution des applications HPC, comme la solution d'un système linéaire. En fait, de la mécanique des fluides numérique [66] jusqu'aux simulations en Astrophysique [113] en passant par l'analyse de données [78], ou l'imagerie du cerveau [75], beaucoup d'applications requièrent de calculer la solution d'un système linéaire (éventuellement creux).

Les méthodes pour résoudre les systèmes linéaires sont généralement séparées en deux catégories : les méthodes directes [47] et les méthodes itératives [101]. Les méthodes directes sont basées sur la factorisation de la matrice associée au système considéré : en général sous la forme d'un produit de matrices triangulaires (LU), et ensuite la solution est trouvée par substitutions. Ces méthodes calculent la solution de n'importe quel système linéaire non singulier en un nombre fini d'opérations, en supposant que l'on néglige les erreurs d'arrondi. En revanche, la quantité de mémoire requise est prohibitive lorsque le système linéaire est très grand. De l'autre côté, les méthodes itératives n'ont besoin de stocker que quelques vecteurs de la taille du système linéaire — même la matrice du système n'a pas besoin d'être stockée explicitement. Néanmoins, la convergence de la méthode est plus sujette aux erreurs d'arrondi, et peut même échouer à trouver une solution acceptable. Afin de prévenir un tel comportement, généralement on ne résout pas le système original $Ax = b$, mais plutôt le système dit préconditionné $M^{-1}Ax = M^{-1}b$, où la matrice M^{-1} est proche de la matrice A^{-1}.

En pratique, une itération classique d'une méthode itérative requiert les opérations suivantes :

- la somme de deux vecteurs,
- le produit d'une matrice (creuse) fois un vecteur,
- et le produit scalaire de deux vecteurs.

Lorsque le problème devient très grand, il n'est pas possible de stocker un vecteur en entier au sein d'un processeur. Généralement, le vecteur est donc divisé parmi les processeurs, et chacun d'eux n'en stocke qu'une partie. De la même manière, la matrice ne peut pas être stockée sur un seul processeur. Si l'on suppose que la matrice est creuse, alors un choix simple est de la distribuer par paquets de lignes. Avec ces hypothèses la somme de deux vecteurs est une opération BLAS1 [43] sans communication entre les processeurs. Le produit matrice–vecteur est une opération BLAS2 avec des communications entre les processeurs voisins. Le produit scalaire est une opération BLAS1 suivie d'une communication globale entre tous les processeurs (Figure 2).

Du coup, le ratio des communications par rapport aux opérations en virgule flottante est élévé, ce qui engendre une mauvaise efficacité sur des machines massivement parallèles. Ceci est particulièrement bien illustré par le benchmark HPCG [44] : en juin 2018, les meilleures machines atteignent environ 1.5% de leur pic de performance lorsqu'elles résolvent un système linéaire creux avec une méthode itérative[6] !

[6] http://icl.utk.edu/hpcg/custom/index.html?lid=155&slid=295

Résumé et contributions

Comme expliqué précédemment, cette thèse fait partie d'un effort global pour améliorer la convergence des méthodes itératives afin de :

- diminuer le ratio communication–calcul,
- augmenter le nombre d'opérations en virgule flottante pour tirer parti des nouvelles architectures des micro–processeurs,
- fournir un solveur qui peut être utilisé comme une boîte noire par l'utilisateur final.

Pour atteindre ce but, nous étudions en détail la méthode dite du Gradient Conjugué Élargi (ECG), initialement proposée dans [58]. Ensuite, nous nous intéressons à un problème lié à l'analyse des observations du fond diffus cosmologique (CMB) où plusieurs systèmes linéaires proches les uns des autres doivent être résolus successivement. Dans ce contexte, nous évaluons les bénéfices potentiels ainsi que les limitations des techniques dites de recyclage. Il y a quatre chapitres dans le manuscrit.

Dans le Chapitre 1, nous rappelons la méthode du Gradient Conjugué ainsi que d'autres méthodes directement dérivées de celle-ci et particulièrement adaptées au calcul parallèle. Plus précisément, nous commençons par rappeler quelques bases d'algèbre linéaire. Ensuite, nous rappelons la définition des méthodes dites de Krylov pour résoudre des systèmes linéaires en se concentrant sur le cas où la matrice est symétrique définie positive. Cela nous mène naturellement à rappeler la définition de la méthode du Gradient Conjugué, ainsi que sa vitesse de convergence. Finalement, nous rappelons quelques méthodes directement dérivées du Gradient Conjugué et qui sont plus adaptées aux architectures parallèles.

Dans le Chapitre 2, la méthode du Gradient Conjugué Élargi est étudiée d'un point de vue théorique. Premièrement, nous présentons une dérivation simplifiée de la méthode en remarquant qu'elle peut être vue comme un cas particulier des méthodes de Krylov par blocs. Cette analogie nous permet de présenter deux variantes de la méthode du Gradient Conjugué : Orthodir et Orthomin. Nous fournissons une explication rigoureuse du manque de robustesse d'Orthomin comparé à Orthodir qui a été observée numériquement. Nous donnons une preuve de la vitesse de convergence de la méthode du Gradient Conjugué Élargi — basée sur une extension de la preuve de [24, Theorem 3.2] — qui améliore le précédent résultat existant présenté dans [58]. Cela montre que l'élargissement des sous–espaces de Krylov agit comme un deuxième niveau pour le préconditionneur, qui atténue l'effet des petites valeurs propres sur la convergence de la méthode. Afin d'accroître l'efficacité de la méthode, nous expliquons comment réduire dynamiquement les directions de recherches. En effet, nous montrons qu'en mesurant le rang d'une petite matrice, il est possible de réduire la taille du bloc pendant les itérations. Une étude théorique nous donne un critère algébrique pour détecter les directions de recherches qui ont un impact significatif sur la convergence. À partir de cette étude, nous dérivons un choix pratique qui n'induit aucun surcoût et qui vérifie le critère théorique dans tous nos cas tests. Ce choix ne dépend pas de la méthode

et peut donc être appliqué dans le contexte de la méthode du Gradient Conjugué par blocs en général. Ceci nous permet de comparer différentes méthodes directement dérivées de celle-ci, dont la méthode du Gradient Conjugué Élargi. Nous observons que la méthode du Gradient Conjugué Élargi est particulièrement bien adaptée à la réduction des directions de recherche. Elle donne les meilleurs résultats en termes d'efficacité — la taille de l'espace de recherche final est significativement réduite — et robustesse — les erreurs d'arrondi n'ont pas d'impact significatif sur la convergence — dans la grande majorité des cas test numériques que nous avons réalisés.

Dans le Chapitre 3, nous présentons la version parallèle de la méthode du Gradient Conjugué Élargi. Nous considèrons à la fois Orthodir et Orthomin, ainsi que les versions dynamiques de ces deux variantes où les directions de recherche sont réduites dynamiquement. La distribution des données et l'analyse du coût de ce design sont présentées en détail. Ensuite, nous présentons des expériences numériques pour évaluer l'efficacité parallèle de ce design ainsi que de la méthode du Gradient Conjugué Élargi sur des machines parallèles. En pratique, nous observons qu'élargir les sous-espaces de Krylov peut entraîner une réduction remarquable du nombre d'itérations. En effet, dans les expériences numériques la méthode est utilisée avec un préconditionneur de type bloc Jacobi, et agit comme un second niveau qui, en quelque sorte, déflate les valeurs propres les plus petites ; ce qui est en accord avec la théorie présentée dans le Chapitre 2. Cela conduit à un gain significatif en termes de temps de calcul comparé à la méthode du Gradient Conjugué classique. À titre d'exemple, pour un problème d'élasticité linéaire en 3D avec des coefficients hétérogènes de taille $4,5$ millions (et 165 millions de coefficients non nuls), nous observons que la méthode ECG est jusqu'à $5,7$ fois plus rapide que l'implémentation de la méthode du Gradient Conjugué présente dans PETSc [8], les deux utilisant un preconditionneur de type bloc Jacobi. Ce cas test est connu pour être difficile parce que les preconditionneurs classiques à un niveau ne sont généralement pas très efficaces [36]. Puisque la méthode ECG accroit le nombre d'opérations en virgule flottante tout en réduisant les communications, nous montrons que cette méthode est particulièrement bien adaptée aux architectures modernes et futures qui possèdent un parallélisme massif. Pour le problème d'élasticité précédent, nous montrons que la méthode passe à l'échelle jusqu'à $16,384$ processus, chacun étant attaché à un cœur physique. Nous souhaitons insister sur le fait que notre objectif n'est pas de concevoir un solveur spécifique pour les équations aux dérivées partielles (EDP) elliptiques. Pour ces cas tests, il est très probable qu'il existe des solveurs qui sont plus efficaces que la méthode ECG avec un préconditionneur de type bloc Jacobi. Néanmoins, à la différence de ces méthodes, ECG est une méthode algébrique. Elle ne nécessite aucune information sur l'EDP sous-jacente et ne repose sur aucune hypothèse particulière, excepté le fait que la matrice soit symétrique définie positive. Par conséquent elle peut être vue comme un solveur type boîte noire et intégrée très facilement dans un code préexistant. Pour illustrer ceci, nous expliquons en détail comment reproduire les expériences numériques. En particulier, nous montrons comment utiliser notre implémentation et nous fournissons un exemple minimal d'utilisation. Ensuite, nous détaillons la marche à suivre pour reproduire les expériences et com-

ment : 1) générer les matrices, 2) télécharger et installer le code, 3) régler les paramètres correctement et soumettre un calcul sur un cluster. Finalement, nous expliquons comment améliorer le passage à l'echelle de la méthode en fusionnant les communications globales qui ont lieu lors d'une itération. Nous effectuons des expériences numériques qui montrent une réduction du temps d'exécution par un facteur allant jusqu'à deux lorsqu'on utilise les versions des algorithmes où les communications sont fusionnées.

Dans le Chapitre 4, nous étudions les stratégies dites de recyclage afin d'augmenter l'efficacité du solveur linéaire dans le contexte de l'analyse des données d'observation du fond diffus cosmologique (CMB). Afin d'effectuer cette analyse, il est nécessaire de résoudre une suite, éventuellement très longue, de systèmes linéaires. A l'intérieur de cette suite, nous supposons que les systèmes changent "peu" les uns par rapport aux autres. Nous examinons les bénéfices possibles qu'il y aurait à utiliser des techniques de recyclage. Elles consistent à améliorer dynamiquement le préconditionneur en utilisant de l'information obtenues lors des précédentes résolutions. Plus précisément, l'information recherchée correspond aux vecteurs propres associés aux plus petites valeurs propres de la matrice. Cette information est ensuite incorporée au préconditionneur grâce des techniques de type deflation. Nous commencons par rappeler plusieurs méthodes pour calculer les valeurs propres et les vecteurs propres associés d'une matrice à partir d'une base (de Krylov) déjà calculée. Ensuite, nous expliquons comme déflater un sous–espace donné, *i.e.*, supprimer son effet éventuellement néfaste sur la convergence de la méthode de Krylov. Ces deux ingrédient sont la base d'un cadre général pour les méthodes de recyclage que nous présentons ensuite. Des expériences numériques sont menées sur des problèmes provenant de l'analyse des données d'observation du fond diffus cosmologique, et l'efficacité de ces méthodes est évaluée d'un point de vue qualitatif. Nous commencons par étudier une situation simplifiée où la matrice sous–jacente est fixée et il y a plusieurs second–membres donnés les uns après les autres. Dans ce cas, les techniques de recyclage nous permettent de réduire le nombre d'itérations significativement. En particulier, lorsque toutes les directions de recherches précédentes sont gardées afin d'approximer les vecteurs propres associés aux plus petites valeurs propres, le nombre total d'itérations est réduit par un facteur jusqu'à 4. Nous étudions ensuite une situtation plus réaliste où à la fois la matrice et les second–membres changent tout au long de la suite. Dans ce cas, ces techniques ne sont pas très efficaces. Les expériences numériques montrent que cela est dû à la multiplicité de la plus petite valeur propre, et que cela est donc inhérent à notre problème. Afin de surmonter cette difficulté, nous proposons une procédure peu coûteuse pour adapter la solution initial avant la résolution. Nous observe alors que le nombre total d'itération est réduit sensiblement. Cette procédure repose sur la structure tensorisée exposée par la matrice sous–jacente.

Pour finir, nous concluons et nous présentons quelques perspectives à ce travail.

! Le reste du manuscrit est rédigé en anglais, à l'exception des résumés qui précédent chaque chapitre et qui sont traduits en français.

Introduction (English version)

Outline of the current chapter

Context

Nowadays almost all the computers, from your personal laptop to the most modern supercomputer including your smartphone, rely on the same paradigm: adding more and more processors to compute faster and faster. This paradigm is relatively new: around 2005 the frequencies of the chips (like a CPU) started to stagnate around 3GHz in order to limit their energy consumption (Figure 1). As a consequence a special care has to be taken when designing and implementing software for these parallel architectures. Indeed, when addressing such parallel machines it is actually much more costly to move the data than performing floating-point operations [42].

A great effort is currently being put in building the first exascale machine, *i.e.*, a machine that can compute 10^{18} floating-point operations per second. For instance, in 2015 Barack Obama signed an executive order creating a National Strategic Computing Initiative calling for the accelerated development of an exascale system. China is supposed to develop an exascale computer during the 13th Five–Year–Plan period (2016–2020). The European Commission adopted in 2012 its High Performance Computing (HPC) strategy, one of the goals of which is the development of an exascale machine within 10 years. These instances recognized that there is an urgent need for new software and algorithms that would be effective on such machines.

Being at the heart of this effort, NLAFET — Parallel Numerical Linear Algebra for Extreme Scale Systems, is a Horizon 2020 FET-HPC project funded by the European Union. Its goal is to minimize the gap between the peak capabilities of the hardware and the performance realized by HPC applications relying on linear algebra software. In practice, it aims at delivering a library that would be used as a black–box for the most time–consuming parts of the HPC applications software such as computing the

[7] https://github.com/karlrupp/microprocessor-trend-data/blob/master/LICENSE.txt

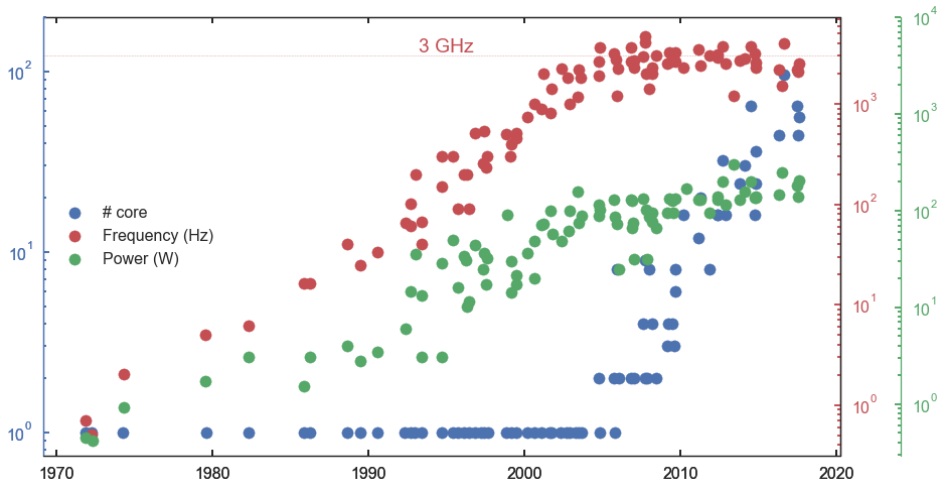


Figure 1 – Microprocessor trend over the last 40 years. Credits: original data collected and provided under Creative Commons License[7]by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond, C. Batten, and K. Rupp.

solution of a linear system. In fact, from computational fluid dynamics [66], to astrophysics simulations [113] including data analysis [78], or brain imaging [75], many applications require computing the solution of a (possibly sparse) linear system.

The methods for solving linear systems are usually split into two categories: direct methods [47] and iterative methods [101]. Direct methods rely on the factorization of the original system matrix in a simpler form: usually a product of triangular matrices (LU), and then the solution is found using backward and forward substitution. They compute the solution of any non singular linear system in a finite number of operations, assuming that round–off errors are neglected. However, their memory requirement is prohibitive when the linear system is very large. On the other hand, iterative methods usually require storing only few vectors of the size of the linear system — even the system matrix does not have to be stored explicitly. However, the convergence of the process is more prone to round–off errors, yielding to delay in the convergence, and even failure to find an acceptable solution. To prevent such undesired behavior, one usually does not solve the original system $Ax = b$ but rather a preconditioned system $M^{-1}Ax = M^{-1}b$ where M^{-1} is close to A^{-1}.

In practice, a typical iteration of an iterative method usually requires the following operations:

- the sum of two vectors,
- the product of a (sparse) matrix times a vector,
- and the dot product of two vectors.

When the problem becomes very large it is not possible to store an entire vector in one processor. Usually the vector is split among the processors, and each one of them

only stores a part of it. Similarly, the matrix cannot be stored in one processor. If we assume that the matrix is sparse, then a simple choice is to distribute the matrix by row panels among the processors. With these assumptions, the sum of two vectors is a BLAS1 [43] operation without any communication. The matrix–vector product is a BLAS2 operation involving communication between neighboring processors. The dot product is a BLAS1 operation followed by a global communication among all the processors (Figure 2).

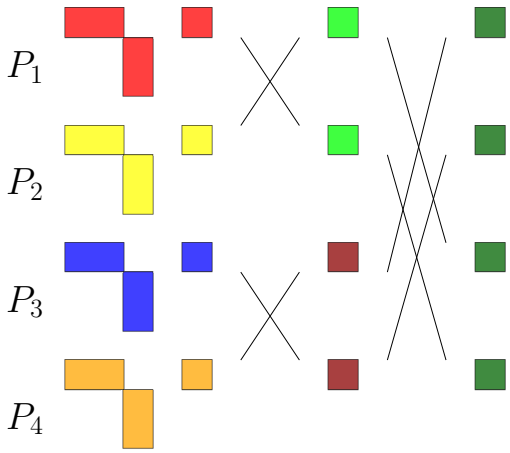


Figure 2 – Schematic workflow of a parallel dot product when the vectors are distributed among 4 processors (P_1, P_2, P_3 and P_4). The straight lines are representing communication through the network.

Thus the communication–to–computation ratio of iterative methods is high, resulting in poor efficiency on massively parallel machines. This is well illustrated by the HPCG benchmark [44]: in June 2018, the best machines reach around 1.5% of their peak performance when solving a sparse linear system with an iterative method[8]!

Summary and contributions

As explained previously, this thesis is part of a global effort for enhancing iterative methods in order to:

- decrease the communication–to–computation ratio,
- increase the arithmetic intensity to take advantage of the new microprocessor architectures,
- provide a solver that can be used as a black–box for the end–user.

[8] http://icl.utk.edu/hpcg/custom/index.html?lid=155&slid=295

To achieve this goal, we study in thorough details the so–called Enlarged Conjugate Gradient method (ECG) originally proposed in [58]. Then, we study a problem related to the data analysis of the cosmic microwave background (CMB) where several linear systems, that are close to each other, have to be solved successively. In this context, we evaluate the potential benefits and limitations of the so–called recycling techniques. There are four chapters in the manuscript.

In Chapter 1, we recall the Conjugate Gradient method as well as other methods directly derived from it, and specially designed for parallel computing. More precisely, we start by recalling some basics about linear algebra. Then, we recall the definition of the Krylov methods for solving linear systems focusing on the case where the matrix is SPD. This naturally leads us to recall the derivation of the Conjugate Gradient algorithm, and the study of its convergence speed. Finally, we recall several methods directly derived from the Conjugate Gradient which are more adapted to parallel architectures.

In Chapter 2, the Enlarged Conjugate Gradient method is studied from a theoretical point of view. First, we present a simplified derivation of the method by noticing that it can be seen as a special case of a block Krylov method. This analogy also allows us to present two variants of the Enlarged Conjugate Gradient method: Orthodir and Orthomin. We provide a rigorous justification of the lack of robustness of Orthomin compared to Orthodir that has been observed experimentally. We give a proof of the speed of convergence of the Enlarged Conjugate Gradient method — based on an extension of the proof of [24, Theorem 3.2] — which greatly improves the previous existing result presented in [58]. This shows that enlarging the Krylov subspaces acts as a second level preconditioner that mitigates the effect of the smallest eigenvalues on the convergence of the iterative method. In order to increase the efficiency of the method, we explain how to dynamically reduce the search directions. Indeed, we show that by monitoring the rank of a small matrix, it is possible to reduce the size of the block during the iterations. A theoretical study gives us an algebraic criterion to detect the search directions that do have a significant impact on the convergence. From this study, we derive a practical choice that induces no extra cost and verifies the theoretical criterion in all our test cases. This choice does not depend on the method and therefore can be applied in the context of the Block Conjugate Gradient method. This allows us to compare several methods derived from it, including the Enlarged Conjugate Gradient method. We observe that the Enlarged Conjugate Gradient method is particularly adapted to the reduction of the search directions, it leads to the best results in terms of effectiveness — the size of the final search space is significantly decreased — and robustness — the round–off errors do not have a significant impact on the convergence — in most of the numerical tests performed.

In Chapter 3, we present the parallel design of the Enlarged Conjugate Gradient method. We consider both Orthodir and Orthomin variants, as well as dynamic versions of these variants that reduce dynamically the number of search directions. The data distribution and the cost analysis of this design are presented in details. Then, we present numerical experiments to asses the efficiency of this design as well as the

Enlarged Conjugate Gradient on parallel machines. In practice, we observe that enlarging the Krylov subspaces can drastically reduce the number of iterations. Indeed in the numerical experiments it is used with a block Jacobi preconditioner and acts as a second-level that, in a way, deflates the smallest eigenvalues; this is in accordance with the theory presented in Chapter 2. This leads to a significant speed-up over the standard Conjugate Gradient method. For instance for a 3D linear elasticity problem with heterogeneous coefficients with 4.5 millions of unknowns and 165 millions of nonzero entries, we observe that ECG is up to 5.7 times faster than the PETSc [8] implementation of the Conjugate Gradient method, both using a block Jacobi preconditioner. This test case is known to be difficult because the classical one-level preconditioners are not expected to be very effective [36]. As it increases the arithmetic intensity and reduces the communication, we show that it is well suited for modern and future architectures that exhibit massive parallelism. For the previous elasticity problem, we show that the method can scale up to 16,384 threads, each one being bound to one physical core. We want to point out that our aim is not to design a specific solver for elliptic partial differential equations (PDE). It is very likely that for these test cases, there exists solvers that are more effective than ECG with a block Jacobi preconditioner. Nevertheless, unlike these methods ECG is an algebraic method. It does not require any information from the underlying PDE and does not rely on any assumption, except that the matrix is symmetric positive definite. Hence it can be seen as a black-box solver and integrated very easily in any existing code. As an illustration, we explain in details how to reproduce the numerical experiments. In particular, we show how to use our implementation and provide a minimal example as an illustration. Then, we detail the workflow for reproducing the experiments and how to: 1) generate the matrices, 2) download and install the code, 3) set the proper parameters when submitting the job to a cluster. Finally, we explain how to increase further the scalability of the method by fusing the global communications that occur during one iteration. We perform numerical experiments that show a reduction of the runtime of a factor up to almost two at large scale when using the fused versions of the algorithms.

In Chapter 4, we study so–called recycling strategies in order to increase the efficiency of the linear solver in the context of the Cosmic Microwave Background (CMB) analysis. The CMB analysis requires solving a possibly very large sequence of symmetric positive definite linear systems. Within this sequence, it is assumed that the systems are changing "slowly" from one to the other. Thus we investigate the possible benefits of using recycling techniques which consist in improving the preconditioner dynamically using informations from the previous solve. More precisely, the information sought is the eigenvectors associated to the smallest eigenvalues of the matrix, and this information is then incorporated in the preconditioner of the next matrix by using deflation. We start by recalling several methods for computing eigenvalues and the associated eigenvectors of a matrix from an already computed (Krylov) basis. Then we explain how to deflate a given subspace, *i.e.*, remove its possibly bad effect on the convergence of the Krylov method. These two ingredients are the basis of the general framework of the recycling methods presented then. Numerical experiments are per-

formed on problems coming from the CMB analysis, and the efficiency of the methods is assessed from a qualitative point of view. We first study a simplified situation where the underlying matrix is fixed, and there are multiple right-hand sides given one by one. In this case, the recycling techniques allow to reduce the number of iterations significantly. In particular, when all the previous search directions are kept in order to approximate the smoothest eigenvectors the overall number of iterations is reduced by a factor up to 4. We then study a more realistic situation where both the matrix and the right-hand sides are changing through the sequence. In this case, these techniques are not very efficient. The numerical experiments show that this is a result of the multiplicity of the smallest eigenvalue, and thus is inherent to our case of interest. In order to overcome this difficulty, we propose a cheap procedure to adapt the initial guess before the solve. It permits to reduce the overall number of iterations noticeably. This procedure relies on the tensorized structure exhibited by the underlying matrix.

Finally, we conclude and present some perspectives of this work.

This thesis has lead to the following publications.

Journal papers in revision

L. Grigori and O. Tissot. *Scalable Linear Solvers based on Enlarged Krylov subspaces with Dynamic Reduction of Search Directions*. Research Report RR-9190. Inria Paris, Laboratoire Jacques-Louis Lions, UPMC, Paris, 2018. Submitted to SIAM SISC.

L. Grigori and O. Tissot. *Reducing the communication and computational costs of Enlarged Krylov subspaces Conjugate Gradient*. Research Report (old version) RR-9023. Inria Paris, Laboratoire Jacques-Louis Lions, UPMC, Paris, 2017. Submitted to NLAA.

Journal paper in preparation

L. Grigori, J. Papez, R. Stompor, and O. Tissot. *Solving sequences of linear systems by recycling deflation subspaces - application to CMB data analysis*. In preparation. 2018.

Deliverables of the H2020 NLAFET project

S. Donfack, L. Grigori, and O. Tissot. *Deliverable 4.5: Integration*. NLAFET deliverable. 2018.

M. Abalenkovs et al. *Deliverable 5.2: Software integration*. NLAFET deliverable. 2018.

S. Donfack, L. Grigori, and O. Tissot. *Deliverable 4.4: Performance evaluation*. NLAFET deliverable. 2018.

S. Donfack, L. Grigori, and O. Tissot. *Deliverable 4.3: Prototype software, phase 2*. NLAFET deliverable. 2017.

S. Donfack, L. Grigori, and O. Tissot. *Deliverable 4.2: Analysis and algorithm design*. NLAFET deliverable. 2017.

1

Preamble

Outline of the current chapter

Abstract

In this chapter, we recall some background about iterative methods for solving linear systems, focusing on Krylov methods and the Conjugate Gradient method in particular. We start by recalling basic facts about linear algebra. Then, we explain how to derive the Conjugate Gradient algorithm, a special case of Krylov methods, as well as a well–knwon result about its speed of convergence. In practice, preconditioning is crucial for the method to converge reasonably, and we review some strategies to construct efficient preconditioners. Finally, we present a brief state of the art of the parallel variants of the Conjugate Gradient method.

Résumé

Dans ce chapitre, on rappelle quelques résultats fondamentaux à propos des methodes itératives de résolution de systèmes linéaires, en se concentrant sur les méthodes de Krylov et la méthode du Gradient Conjugué en particulier. On commence par rappeler quelques éléments basiques d'algèbre linéaire. Ensuite, on explique comment dériver l'algorithme du Gradient Conjugué, un cas particulier de méthode de Krylov, ainsi qu'un résultat connu à propos de sa vitesse de convergence. En pratique, le préconditionnement est crucial pour que la méthode

converge raisonnablement et on passe en revue quelques stratégies pour construire des préconditionneurs efficaces. Finalement, on présente un bref état de l'art des variantes parallèles de la méthode du Gradient Conjugué.

1.1 Background in Linear Algebra

In this section, we recall some basic results of linear algebra and introduce notations that will be used throughout this thesis. We do not cite explicit references for the results stated in this section. They should be found in any good graduate-level course about linear algebra. In particular, further details can be found in [55, 101] for instance[1].

In this thesis, we will manipulate mainly two mathematical objects: vectors and matrices. We call v a *vector* if it is an element of $\mathbb{K}^n$ where $\mathbb{K} = \mathbb{R}$ or $\mathbb{K} = \mathbb{C}$, and $n > 0$ is an integer. For the sake of simplicity, we consider that v is real, *i.e.*, $v \in \mathbb{R}^n$. We call A a *matrix* if it is an element of $\mathbb{K}^{m\times n}$ where $\mathbb{K} = \mathbb{R}$ or $\mathbb{K} = \mathbb{C}$, and $m, n > 0$ are integers. Similarly, we consider that A is real, *i.e.*, $A \in \mathbb{R}^{m\times n}$.

In fact, one should see vectors as columns of real numbers, and matrices as arrays of numbers with m rows and n columns. In spite of the apparent simplicity of these objects, we will see that their study is not that simple. The coefficient at row i and column j of the matrix A is denoted a_{ij}. We denote $A^\top$ the matrix whose coefficient are defined by $c_{ij} = a_{ji}$ — the rows and columns are switched in the coefficients. Similarly, $v^\top$ is used to denote a row vector.

We start by the definitions of inner product and norm for vectors.

Definition 1.1.1: inner product

Let $n \in \mathbb{N}^*$, an inner product on $\mathbb{R}^n$ is any mapping denoted $\langle .,. \rangle$ from $\mathbb{R}^n \times \mathbb{R}^n$ to $\mathbb{R}$ such that,

1. linearity: $\forall v, u_1, u_2 \in \mathbb{R}^n, \forall \lambda_1, \lambda_2 \in \mathbb{R}, \langle \lambda_1 u_1 + \lambda_2 u_2, v\rangle = \langle \lambda_1 u_1, v\rangle + \langle \lambda_2 u_2, v\rangle$,
2. symmetry: $\forall u, v \in \mathbb{R}^n, \langle u, v\rangle = \langle v, u\rangle$,
3. positive definiteness: $\forall u \in \mathbb{R}^n \setminus \{0\}, \langle u, u\rangle > 0$

Definition 1.1.2: norm

Let $n \in \mathbb{N}^*$, a norm on $\mathbb{R}^n$ is any mapping denoted $\|.\|$ from $\mathbb{R}^n$ to $\mathbb{R}^+$ such that,

1. triangle inequality: $\forall u, v \in \mathbb{R}^n, \|u + v\| \leqslant \|u\| + \|v\|$,
2. homogeneity: $\forall u \in \mathbb{R}^n, \forall \lambda \in \mathbb{R} \|\lambda u\| = |\lambda| \|u\|$,
3. positive definiteness: $\forall v \in \mathbb{R}^n, \|v\| \geqslant 0$, and $\|v\| = 0$ iff $v = 0$.

[1] These 2 very popular books cover much more material: [55] focuses on numerical linear algebra, and [101] covers iterative methods for solving linear systems.

Intuitively, the norm can be seen as a *measure* of a vector — although this does not mean that different vectors should have different norms. An important property that links inner product and norm is the Proposition 1.1.1: given an inner product it is always possible to construct a norm — but the opposite is not true.

Proposition 1.1.1

Given an inner product $\langle .,. \rangle$, the quantity $\sqrt{\langle .,. \rangle}$ defines a norm.

We now define the notion of orthogonality which is crucial in all the methods that we will discuss in this thesis.

Definition 1.1.3: orthogonality

Let $u, v \in \mathbb{R}^n$, then u, v are said to be *orthogonal* (denoted $u \perp v$) with respect to $\langle .,. \rangle$ if and only if,

$$\langle u, v \rangle = 0. \tag{1.1}$$

A fundamental theorem, the Riesz representation theorem[2], makes the link between the notion of inner product and a special type of matrices. Indeed, one can represent any scalar product using a matrix.

Theorem 1.1.1: Riesz representation

Given an inner product $\langle .,. \rangle$ then there exists a unique matrix $A \in \mathbb{R}^{n \times n}$, such that,

$$\forall u, v \in \mathbb{R}^n, \langle u, v \rangle = u^\top A v. \tag{1.2}$$

Definition 1.1.4: symmetric positive definite matrix

$A \in \mathbb{R}^{n \times n}$ is symmetric positive definite if and only if,

1. symmetry: $A^\top = A$,
2. positive definiteness: $\forall v \in \mathbb{R}^n \setminus \{0\}, v^\top A v > 0$.

In fact, it is easy to show that any symmetric positive definite (SPD) matrix induces an inner product. Thus, there is an equivalence between SPD matrix and inner product. So given a SPD matrix A, it is straightforward to define an A–norm, denoted $\|.\|_A$ and an A–orthogonality denoted $\perp_A$.

[2]Although its exact statement is much more general, we decided to restrict the presentation to our simplified case where the Hilbert space is simply $\mathbb{R}^n$.

1.2 Krylov Subspace Methods

In this section, we recall the definition of Krylov subspaces methods for solving linear systems. Our presentation is brief and much more details can be found in [101, Chapter 5] and [82, Chapter 2]. We focus on the case where $A \in \mathbb{R}^{n\times n}$ is SPD, and on the induced method: the Conjugate Gradient (CG) method [68].

Let us recall that we are interested in solving the linear system,

$$Ax = b, \tag{1.3}$$

where $b \in \mathbb{R}^n$ is a given right-hand side $x \in \mathbb{R}^n$ is the unknown solution. Krylov subspaces methods are iterative methods that rely on 2 ingredients: first a *projection process*, second the definition of *Krylov subspaces*.

First, we briefly explain what is a projection process. Given an initial guess x_0[3], it searches an *approximate* solution x_k of the form,

$$x_k \in x_0 + \mathcal{S}_k, \tag{1.4}$$

where $\mathcal{S}_k$ is some k-dimensional subspace of $\mathbb{R}^{n\times n}$ called the *search space*. The condition (1.4) is also known as the *subspace condition*. Of course, this condition is not enough to define uniquely x_k and one need to add k *constraints* on this approximate solution. This is done by enforcing the following condition,

$$b - Ax_k \perp \mathcal{C}_k, \tag{1.5}$$

where $\mathcal{C}_k$ is some k-dimensional subspace of $\mathbb{R}^{n\times n}$ called the *constraints space*. The quantity $b - Ax_k$ is also denoted r_k and it is called the *residual*. The condition (1.5) is also known as the *Petrov-Galerkin condition*. It is possible to show that if the matrix A is SPD, then $\mathcal{C}_k$ can be chosen equal to $\mathcal{S}_k$ ([101, Proposition 5.1] for instance).

It becomes clear that the choice of the subspace $\mathcal{S}_k$ is crucial as for the efficiency of the resulting method. Here comes into play the second ingredient: the definition of the Krylov subspaces denoted $\mathcal{K}_k(A, r_0)$. These subspaces are defined such as,

$$\mathcal{K}_k(A, r_0) = \text{span}\left\{r_0, Ar_0, \ldots, A^{k-1}r_0\right\}. \tag{1.6}$$

One major property of these subspaces is given by Theorem 1.2.1 [82, Theorem 2.2.3].

Theorem 1.2.1

If r_0 is of grade d with respect to A, *i.e.*, the degree of the nonzero monic polynomial p of lowest degree such that $p(A)v = 0$ is d, then $r_0 \in \mathcal{K}_1 \subset \mathcal{K}_2 \subset \ldots \mathcal{K}_d = \mathcal{K}_{d+j}$, for all $j \geqslant 0$. Moreover, $r_d = 0$

We want to emphasize on 2 intuitive motivations for searching an approximate so-

[3] It is always possible to choose it equals to 0, if no "good" first guess is known.

lution in these subspaces:

1. the method finds the exact solution x in at most n iterations,
2. the construction of $\mathcal{K}_k(A, r_0)$ involves a sequence of matrix-vector products which is both simple to implement and relatively cheap.

Remark 1.2.1

In this section and in the rest of this thesis, we focus on the case where A is SPD. In this case, the method of choice is the Conjugate Gradient algorithm. However, when the matrix is not SPD, there exist a lot of Krylov methods such as GMRES [99], BiCGSTAB [118], or QMR [49]. We do not cover these methods in our presentation as we are interested in the case where A is SPD.

1.2.1 Derivation of the Conjugate Gradient algorithm

Following the previous discussion, the iterative process characterized by,

$$x_k \in x_0 + \mathcal{K}_k(A, r_0), \tag{1.7}$$
$$r_k \perp \mathcal{K}_k(A, r_0), \tag{1.8}$$

is well defined. In particular, we have seen that x_n is the exact solution of the original linear system.

From these 2 conditions and the Lanczos algorithm [101, Algorithm 6.15], it is possible to show Lemma 1.2.1.

Lemma 1.2.1

Let x_k and r_k ($k \in \{1, \dots, n\}$) satisfying the iterative process (1.7)–(1.8), then,

$$x_k = x_{k-1} + p_k \alpha_k, \tag{1.9}$$
$$r_k = r_{k-1} - A p_k \alpha_k, \tag{1.10}$$

where,

$$\alpha_k \in \mathbb{R} \tag{1.11}$$
$$p_k \in \mathcal{K}_k(A, r_0), \tag{1.12}$$
$$p_i^\top A p_k = 0, i \leqslant k. \tag{1.13}$$

Proof. See [101, Section 6.7.1], or [82, Section 2.5.1]. □

One can easily see that $r_k \in \mathcal{K}_{k+1}(A, r_0)$ and $r_k \perp \mathcal{K}_k(A, r_0)$. In particular, it is easy to see that $r_k \perp A p_i$ if $i \leqslant k-1$, which is equivalent to $p_i^\top A r_k = 0$. Following Lemma 1.2.1,

it seems natural to construct p_{k+1} such that,

$$p_{k+1} = r_k - p_k \beta_k, \tag{1.14}$$

where β_k is a scalar. In order to determine β_k, one has to enforce,

$$p_k^\top A p_{k+1} = 0 \iff p_k^\top A(r_k - p_k\beta_k) = 0 \tag{1.15}$$

$$\iff \beta_k = \frac{p_k^\top A r_k}{p_k^\top A p_k} \tag{1.16}$$

Similarly, a simple computation allows us to determine α_k,

$$p_k^\top r_k = 0 \iff p_k^\top r_{k-1} = \alpha_k, \tag{1.17}$$

$$\iff \alpha_k = \frac{p_k^\top r_{k-1}}{p_k^\top A p_k}. \tag{1.18}$$

We can notice that $p_k^\top A p_k = \|p_k\|_A$ so we can A–normalize the search directions in order to further simplify the expression of α_k and β_k leading to Algorithm 1. This is known as the Conjugate Gradient method [68].

Algorithm 1 Conjugate Gradient (CG)

$r_0 = b - Ax_0$
$p_1 = r_0(r_0^\top A r_0)^{-1/2}$
$k = 1$
while $\|r_{k-1}\|_2 > \varepsilon_{\text{solver}}\|r_0\|_2$ and $k < k_{\max}$ **do**
  $\alpha_k = p_k^\top r_{k-1}$
  $x_k = x_{k-1} + p_k\alpha_k$
  $r_k = r_{k-1} - Ap_k\alpha_k$
  $\beta_k = p_k^\top A r_k$
  $z_{k+1} = r_k - p_k\beta_k$
  $p_{k+1} = z_{k+1}(z_{k+1}^\top A z_{k+1})^{-1/2}$
  $k = k + 1$
end while

Remark 1.2.2

The Conjugate Gradient algorithm is sometimes written in a slightly different form where p_k is not A–normalized, and β_k is expressed in terms of $\|r_k\|_2$ and $\|r_{k-1}\|_2$ [101, Algorithm 6.18]. We decided to show this form because it is closer to the algorithm of the Enlarged Conjugate Gradient method.

1.2.2 Convergence study

As the Conjugate Gradient method is an iterative method it is of primary interest to characterize its speed of convergence. First of all, it is possible to prove that the Conjugate Gradient method finds the approximate solution x_k that minimizes the A–norm of the error over the Krylov subspace $\mathcal{K}_k(A, r_0)$ (Proposition 1.2.2 [101, Proposition 5.2]).

Proposition 1.2.1

Let x_k ($k \in \{1, \dots, n\}$) satisfying the iterative process (1.7)–(1.8), then

$$\|x - x_k\|_A = \min_{y \in x + \mathcal{K}_k(A, r_0)} \|x - y\|_A \tag{1.19}$$

Moreover, one can show [82, p. 265] that the approximate residual r_k can expressed as,

$$r_k = q_k(A) r_0, \tag{1.20}$$

where q_k is a polynomial of degree k, such that $q_k(0) = 1$. We denote $\mathbb{P}_1^k$ the set of such polynomials. Moreover, we have $r_k = A(x - x_k)$ thus,

$$x - x_k = q_k(A)(x - x_0). \tag{1.21}$$

Let $A = \Phi^\top \Lambda \Phi$ be the spectral decomposition of A whose eigenvalues are denoted λ_i ($1 \leqslant i \leqslant n$), then we have,

$$\|x - x_k\|_A = \|q_k(\Lambda)\Phi(x - x_0)\|_A \tag{1.22}$$

This expression and Proposition 1.2.2 imply that,

$$\begin{aligned} \|x - x_k\|_A &= \min_{q_k \in \mathbb{P}_1^k} \|q_k(\Lambda)\Phi(x - x_0)\|_A && (1.23)\\ &\leqslant \|(x - x_0)\|_A \min_{q_k \in \in \mathbb{P}_1^k} \|q_k(\Lambda)\|_A && (1.24)\\ &\leqslant \|(x - x_0)\|_A \min_{q_k \in \in \mathbb{P}_1^k} \max_{1 \leqslant j \leqslant n} \|q_k(\lambda_j)\|_A && (1.25) \end{aligned}$$

Then, it is possible to use Chebyshev polynomials to estimate the min-max quantity [101, Theorem 6.25],

$$\min_{p \in \mathbb{P}_1^k} \max_{1 \leqslant i \leqslant n} |p(\lambda_i)| \leqslant 2 \left(\frac{\sqrt{\kappa} - 1}{\sqrt{\kappa} + 1} \right)^k, \tag{1.26}$$

where κ denotes the condition number of the matrix, $\kappa = \frac{\lambda_{\max}}{\lambda_{\min}}$.

Finally, a very popular result concerning the convergence of the Conjugate Gradient algorithm follows,

$$\|x - x_k\|_A \leqslant 2\|x - x_0\|_A \left(\frac{\sqrt{\kappa} - 1}{\sqrt{\kappa} + 1} \right)^k \tag{1.27}$$

Remark 1.2.3

Although very popular, this bounds has several limitations in practice both theoretical, and practical due to round–off errors (see [82] for a very interesting discussion about these flaws).

1.3 Preconditioners

We have seen that the convergence of the Conjugate Gradient method is related to the condition number κ of the matrix A. To accelerate the convergence of the method, one usually solves a preconditioned system instead of the original one, *i.e.*,

$$M^{-1}Ax = M^{-1}b, \tag{1.28}$$

where $M^{-1}A$ is assumed to have a better condition number than A.

The careful reader may have noticed that this formulation is somehow problematic for the Conjugate Gradient method because $M^{-1}A$ is obviously non–symmetric even if A and M are SPD. However, it is possible to derive a preconditioned version of the Conjugate Gradient method by noticing that $M^{-1}A$ is self–adjoint with respect to the M-inner product,

$$x^\top M(M^{-1}Ay) = (M^{-1}Ax)^\top My = x^\top Ay. \tag{1.29}$$

Thus, by replacing A by $M^{-1}A$ and $^\top$ by $^\top M$ in Algorithm 1, it follows the Preconditioned Conjugate Gradient algorithm (Algorithm 2).

Algorithm 2 Preconditioned Conjugate Gradient (PCG)

$r_0 = M^{-1}(b - Ax_0)$
$p_1 = r_0(r_0^\top Ar_0)^{-1/2}$
$k = 1$
while $\|r_{k-1}\|_2 > \varepsilon_{\text{solver}}\|r_0\|_2$ and $k < k_{\max}$ **do**
  $\alpha_k = p_k^\top r_{k-1}$
  $x_k = x_{k-1} + p_k\alpha_k$
  $r_k = r_{k-1} - Ap_k\alpha_k$
  $z_{k+1} = M^{-1}r_k$
  $\beta_k = p_k^\top Az_{k+1}$
  $z_{k+1} = z_{k+1} - p_k\beta_k$
  $p_{k+1} = z_{k+1}(z_{k+1}^\top Az_{k+1})^{-1/2}$
  $k = k + 1$
end while

Of course, the choice of the preconditioner M has a huge impact on the performance of the Krylov method: 1) computing $M^{-1}v$ should be cheap and 2) M^{-1} has to be as

close as possible of A^{-1} — these 2 requirements being conflicting. We now review some popular families of preconditioners.

1.3.1 Incomplete factorization

In order to understand this type of preconditioners, one has to go back to direct methods [47]. These methods factorize A such that $A = LU$, L is a lower triangular square matrix, and U is an upper triangular matrix. When A is sparse, there are more non–zeros entries in L and U than in A, this phenomenon is known as fill–in.

Incomplete factorizations are decompositions of A such that,

$$R = A - LU, \tag{1.30}$$

where L is sparse lower triangular, U is sparse upper triangular and R verifies some properties in order to limit memory requirement for storing L and U.

For instance, ILU(0) defines L and U such that there is no fill–in, *i.e.*, R has zero entries at the location of non–zero entries of A. It is possible to extend this idea in several ways. First, ILU(p), where p is an integer, aims at improving the quality of ILU(0) by allowing fill–in up to the level p [101, Definition 10.5]. Second, ILU(τ) where τ is a real number, also aims at improving the quality of ILU(0), but it allows fill–in in the entry if the factor is larger than τ. It results from the observation that many values in the L and U factors are usually very small.

In practice, as direct methods, such preconditioners can be used as a black–box but they are lacking of parallelism. However, a Communication–Avoiding ILU(0) preconditioner has been derived [57].

1.3.2 Domain Decomposition

Domain Decomposition methods are used to solve linear systems coming from the discretization of Partial Differential Equations. Originally, they were derived as numerical schemes based on a "divide and conquer" idea, but it was found that they can be expressed as preconditioners for Krylov solvers [36, 117]. These preconditioners naturally express a high degree of parallelism. Even if they ultimately rely on the discretization of some differential operator, some algebraic formulations have been derived [36, 50, 51], and our presentation is intended to be as algebraic as possible, pointing out when the non–algebraic hypothesis are needed.

First, we need to introduce some notations. As previously, A denotes a symmetric positive definite matrix of size $n \times n$. Let N be an integer such that $N \ll n$. We assume that A is partitioned into N subdomains, with or without overlapping. More precisely, we use the following definition and notations inspired by [36, 50, 51, 117]. Let $\mathcal{N} = \{1,\dots,N\}$ and let $\mathcal{N} = \bigcup_{i=0}^{N} \mathcal{N}_{i,0}$ a partition of $\mathcal{N}$ where the $\mathcal{N}_{i,0}$ are nonempty and pairwise disjoint. For each $\mathcal{N}_{i,0}$ we consider a sequence of nested subsets $\mathcal{N}_{i,\delta}$ such that,

$$\mathcal{N}_{i,0} \subseteq \mathcal{N}_{i,1} \subseteq \mathcal{N}_{i,2} \subseteq \cdots \subseteq \mathcal{N}, \tag{1.31}$$

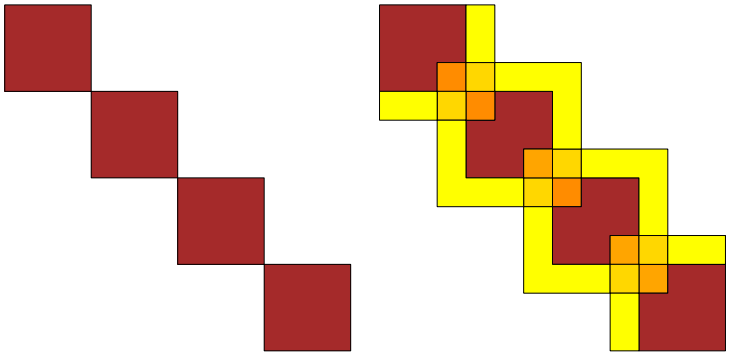


Figure 1.1 – Schematic view of some additive Schwarz preconditioners without overlap (Left) or with overlap (Right).

and $\mathcal{N} = \bigcup_{i=0}^{N} \mathcal{N}_{i,\delta}$ for all δ but when $\delta > 0$ the sets are no longer necessary pairwise disjoint, so δ represents the overlap. Let $n_{i,\delta}$ be the cardinal of $\mathcal{N}_{i,\delta}$. The matrix $R_{i,\delta} \in \mathbb{R}^{n_{i,\delta} \times n}$ whose rows are the rows j of identity for which $j \in \mathcal{N}_{i,\delta}$ are called restriction matrices. Indeed, if we denote $W_{i,\delta} = \text{Range}\left(R_{i,\delta}^{\top} R_{i,\delta}\right)$, and $v \in \mathbb{R}^n$ a vector then $v_{i,\delta} = R_{i,\delta} v$ is the restriction of v on $W_{i,\delta}$. Similarly, $A_{i\delta} = R_{i\delta} A R_{i\delta}^{\top}$ is the restriction of A on $W_{i,\delta}$.

Following [36], it is possible to define a general one-level additive Schwarz preconditioner $M_{ASM,\delta}^{-1}$ (Figure 1.1) as,

$$M_{ASM,\delta}^{-1} = \sum_{i=1}^{N} R_{i,\delta}^{\top} (R_{i,\delta} A R_{i,\delta}^{\top})^{-1} R_{i,\delta}. \tag{1.32}$$

The case where $\delta = 0$ leads to the Block Jacobi preconditioner.

For the sake of simplicity, we drop the subscript δ and denote $M_{ASM,\delta}$ as M_{ASM}. This definition is completely algebraic as it only requires $R_{i,\delta}$ and $R_{i,\delta}^{\top}$. Despite its simplicity the following result holds ([36, Lemma 5.9 p.120] or with a slightly different formulation [117, Lemma 2.6 p.42]).

Lemma 1.3.1

Let A be a symmetric positive definite sparse matrix and $\delta \geqslant 0$ an overlap. The largest eigenvalue of $M_{ASM}^{-1} A$ is bounded by a constant independent of N. In other words, for each $u \in \mathbb{R}^n$ not zero,

$$0 < u^{\top} M_{ASM}^{-1} A u \leqslant (k+1) u^{\top} u, \tag{1.33}$$

where $k = \max_{i \in \{1,\dots,N\}} \{\#\mathcal{N}(i)\}$ and $\mathcal{N}(i)$ denotes the set of neighbors of subdomain i.

However, it has some limitations because it is not possible to control the condition number of $M_{ASM}^{-1} A$. Indeed, no information is provided about the smallest eigenvalues

of $M_{ASM}^{-1}A$. In practice, $M_{ASM}^{-1}A$ can have eigenvalues close to 0 leading to very slow convergence of Krylov methods [36].

In order to overcome this limitation of M_{ASM}, it is usual to add a so-called coarse space that contains information about the eigenspace associated with the smallest eigenvalues of $M_{ASM}^{-1}A$ in order to deflate it [36, 48, 62, 115, 117]. For instance, GenEO [107] provides a construction of a coarse space that can bound the condition number of the resulting preconditioned matrix by a given threshold. This construction is based on the solution of local generalized eigenvalue problems leading to a remarkable efficiency at large scale [74]. However, the parallel implementation also requires an access to an existing software for the discretization of the underlying partial differential equation. In that sense, it cannot be used as a black–box solver. To our knowledge, there is no coarse space which is both algebraic, in the sense that it only needs the global matrix A, reliable, it bounds the condition number of the preconditioned matrix, and effective in parallel, its construction is local and does not involve global communication.

1.3.3 Multigrid

As Domain Decomposition methods, Multigrid methods [120] were originally designed for solving partial differential equations (PDE). They rely on a nested sequence of discretizations with different mesh sizes, approximating the solution using a relaxation technique through this sequence. This is also known as Geometric Multigrid. However, the theory is in general known for special cases such as the Poisson equation and strongly relies on the underlying partial differential equation to decompose effectively the error in its so–called "oscillatory" and "smooth" part.

Indeed, this idea has been extended to non–PDE problems by considering the graph of the matrix directly, this is known as Algebraic Multigrid (AMG). Although it is possible to derive algorithms by analogy with the Geometric case, it becomes much more complicated to prove convergence results. As an exception, one can cite [84] where the convergence rate is proven for symmetric M-matrices with non–negative row sum. However, one important feature of multigrid methods is that they can decrease the residual extremely rapidly during the first iterations [120]. Thus, it seems natural to define an Algebraic Multigrid preconditioner such as it performs only one step of the method for instance. Such approach has been very successful [10, 97, 112, 124]. We also want to point out *Jolivet*'s thesis [73] which provides a detailed comparison between the performance of Domain Decomposition and Multigrid methods. In particular, it is shown that Multigrid methods do require to tune some parameters in order to obtain good performances, especially when there is a high heterogeneity in the coefficients of the differential operator, making their usage as black–box sometimes complicated for the non–expert user.

1.4 Parallel design of Krylov Methods

Recently, a lot of effort has been put in enhancing the performance of Krylov methods by avoiding global communication [19, 25, 26, 69], overlapping communication with computation [54], or decreasing the number of iterations by searching in multiple directions at once [58, 108].

1.4.1 Mitigating the effect of communication

As explained previously, the major performance bottleneck of the Conjugate Gradient algorithm comes from the global synchronizations for computing α_k and β_k (line 5 and 8 of Algorithm 1). In what follows, we review some algorithmic reformulations in order mitigate the number of global synchronizations. One common feature of these methods is that they are all theoretically equivalent to the Conjugate Gradient method.

Avoiding communication

The main idea behind s–step methods [25, 26, 27], also known as Communication–Avoiding Krylov methods [19, 22, 69], is to perform s iterations at the same time instead of one, thus reducing the number of global synchronizations by a factor s.

Schematically, these methods construct

$$P_k = \begin{pmatrix} p_{k+1} & p_{k+2} & \dots p_{k+s} \end{pmatrix},$$

from

$$Y_k = \begin{pmatrix} r_k & Ar_k \dots & A^{s-1} r_k \end{pmatrix}.$$

Using so–called "matrix powers kernel" [35], it is possible to construct Y_k at the same asymptotical communication cost than a matrix–vector product. Once Y_k is constructed, the construction of P_k involves a global communication [20, Algorithm 2]. In practice, due to round–off errors the numerical behavior of s–step methods can be much worse than that of the standard methods and the parameter s has to be chosen very carefully [20]. Several improvements have been proposed including residual replacement [17], using deflation [21], or the adaptation of the parameter s dynamically during the iterations [20, 70].

However, the special data distribution induced by the matrix powers kernel lead to a lack of preconditioners adapted to these methods — CA-ILU(0) [57] being an exception. In particular, Multigrid, and Domain Decomposition preconditioners cannot be applied as this.

Pipelining communication with computation

In a similar vein, Pipelined methods [53, 54] aim at overlapping the communication with computation. Starting from the s–step Conjugate Gradient method [26], *Ghysels*

and Vanroose [54] consider the case $s = 1$. They propose to introduce the following variables,

$$s_k \equiv Ap_k, \tag{1.34}$$
$$w_k \equiv Ar_k, \tag{1.35}$$
$$z_k \equiv As_k \equiv A^2 p_k, \tag{1.36}$$

for which they derive recurrence formulas that do not involve any additional communications. In doing so they are able to decouple the application of the matrix–vector product and the dot products within an iteration, and thus to overlap them. It is also possible to derive a Preconditioned Pipelined Conjugate Gradient by introducing 2 more variables and the corresponding recurrences formulas. Recently, a generalization of this method called the Deep Pipelined Conjugate Gradient method has been proposed where l ($l \in \mathbb{N}^*$) iterations can be overlapped [31].

Because they are very similar in their spirits, Pipelined and s–step methods are also suffering from the same flaws. In particular, the Pipelined methods exhibit a high sensibility to round–off errors [18]. Several remedies have been proposed such as residual replacement [54], or introducing some shifts [30]. Also, it seems that the Deep Pipelined Conjugate Gradient method with $l = 1$ leads to a better maximum attainable accuracy than the original Pipelined Conjugate Gradient method [31]. As pointed out in [18], it is not an easy task to analyze thoroughly the finite precision behavior of the Krylov subspaces methods. Hence it is not surprising that there is still a lack of understanding of the numerical behavior of the Pipelined methods despite some recent efforts in this direction [29].

1.4.2 Searching in several directions at once

Block Krylov methods are receiving an increasing attention in the HPC field [3, 15, 28, 75, 79, 108]. They appear to be well suited for modern computers' architectures with a high level of parallelism because they allow to reduce the number of global synchronizations, while also featuring a higher arithmetic intensity at the cost of some extra computations. Unlike Communication–Avoiding or Pipelined methods, they are no longer equivalent to the Conjugate Gradient method because the solution is no longer sought in standard Krylov subspaces.

Block Krylov methods

The seminal paper of *O'Leary* [89] introduced the first block Krylov method: the Block Conjugate Gradient method (Block CG). In summary, the main idea is to replace the vectors in the standard CG method (such as x_k, r_k and p_k) by tall and skinny matrices of size $n \times t$ where $t \ll n$ (usually denoted with uppercase, *i.e.*, X_k, R_k, P_k, etc.), and the scalars (such as α_k) become $t \times t$ matrices. This method was motivated by problems which require to solve a linear system with several right-hand sides and it was

shown theoretically that it can converge significantly faster than the standard Conjugate Gradient method [89]. This idea was then generalized and extended to other standard Krylov methods as GMRES[80, 95], MINRES [104], or BiCGSTAB [64]. Later *Gutknecht* [65] introduced a general framework for defining Block Krylov subspaces.

As the arithmetic cost increases with the number of right-hand sides, Block Krylov methods usually become prohibitive when the number of right-hand sides becomes large. In the early 90s, *Nikishin and Yeremin* [86] propose to reduce the size of the block during the iterations of the Block CG by monitoring the rank of the residual matrix. More recently *Robbé and Sadkane* in [95] improved the idea introduced by *Nikishin and Yeremin* and applied it to the Block GMRES method. That way they aim at obtaining the convergence behavior of the block method while maintaining an acceptable arithmetic extra cost compared to the standard Conjugate Gradient method. In fact, as pointed out by *Gutknecht* [65], so–called *inexact breakdowns* where the residual matrix is almost rank deficient are inherent to Block Krylov methods, and they have to be taken into account in order to increase their efficiency.

Linear systems with single right-hand side

Even if the Block CG method was initially used to solve linear systems with several right-hand sides [89], it is possible to use a block method to solve a linear system with single right-hand side. This is useful because Block CG can converge significantly faster than CG [89, Theorem 5]. Let us recall that A is a SPD matrix of size $n \times n$ and X_0, B are matrices of size $n \times t$. Hence the parameter t is the number of (initial) search directions, or the block size. We denote $\mathbb{1}_t$ a row of size $1 \times t$ and full of ones.

In the early 90s, *Nikishin and Yeremin* [86] introduced a novel method based on the Block Conjugate Gradient to solve a linear system with single right-hand side. We refer to this method as *BRRHS-CG*. In this method, the initial residual matrix R_0 is chosen such that $R_0^{(i)}$ is a random vector, $\forall t \geqslant i > 1$ and $R_0^{(1)} = r_0$. If the first residual is rank deficient a QR factorization is performed in order to get a new R_0 which is full rank [86]. The method is stopped as soon as the first column of the residual matrix has a norm smaller than $\varepsilon_{\text{solver}}\|r_0\|$ and the solution is given by the first column of the approximate solution matrix.

In the late 90s, *Brezinski* introduced *Multi-parameter Descent Methods* [12, 13]. Instead of using one search direction at each iteration he proposed to use several. This variant is a hybrid between a block method and a method for solving systems with single right-hand side: even if the descent directions and the steps are blocks, as in the block methods, the remaining quantities are vectors, as in standard methods.

With a different point of view *Bhaya et al.* [9] introduced the Cooperative Conjugate Gradient (Coop-CG) where several agents cooperate in order to solve a linear system. This method can be seen as Block CG that uses several random initial guesses [58]. More precisely, X_0 is an uniform random matrix and $B = b\mathbb{1}_t$. As soon as one column of the residual matrix has a norm smaller than $\varepsilon_{\text{solver}}\|r_0\|$, the method is stopped. The approximate solution is given by the corresponding column of the approximate solution

matrix.

Finally, we also want to mention the *AMPCG* method recently proposed by *Spillane* [108]. It aims at improving the Multi–Preconditioned CG method [14] by selecting adaptively some search directions during the iterations. Indeed, both methods take root in the Domain Decomposition framework and they assume that a family of preconditioners is given, $M_1,\ldots,M_{\mathcal{N}}$. Then, at iteration k, they construct $\mathcal{N}$ search directions such as

$$P_{k+1} = \begin{pmatrix} M_1^{-1} r_k & \ldots & M_{\mathcal{N}}^{-1} r_k \end{pmatrix} - \sum_{i=1}^{k} P_i (P_i^\top A P_i)^{-1} P_i^\top A \begin{pmatrix} M_1^{-1} r_k & \ldots & M_{\mathcal{N}}^{-1} r_k \end{pmatrix}. \tag{1.37}$$

In particular, the short–recurrence property of the Conjugate Gradient method is lost, and all the previous search directions must be kept. Also, in its original form, the number of search directions constructed at each iteration is typically of the order of the number of subdomains which is prohibitive in practice. In [108], the author proposes to reduce adaptively the number of search directions using a criterion that is algebraic but requires to know the smallest eigenvalue of the matrix A. It has been applied in the context of the FETI method [23] which provides such information, but it is not algebraic in the sense that it requires some knowledge about the underlying partial differential equation. This method is very close to the *MSDO–CG* method introduced in [58] where the residual r_k is split at each iteration, and the short recurrence property is lost. One difference is that the splitting used in MSDO–CG is completely algebraic, but the number of search directions is not reduced during the iterations.

2

Enlarged Conjugate Gradients

Outline of the current chapter

Abstract

In this chapter, we present a novel derivation of the Enlarged Conjugate Gradient (ECG) method making the analogy with the Block Conjugate Gradient method. More precisely, we derive 2 variants of the method: Orthomin and Orthodir. Then, we study the relationship between the 2 variants, and we prove a new speed of convergence result for ECG that greatly improves the previous known result. Finally, we explain how to dynamically reduce the number of search directions during the iterations in order to mitigate the effect of the additional arithmetic operations induced by the method. These theoretical results are illustrated by numerical experi-

ments.

Résumé

Dans ce chapitre, nous présentons une nouvelle dérivation de la méthode du Gradient Conjugué Élargi en faisant l'analogie avec la méthode du Gradient Conjugué par Blocs. Plus précisément, nous dérivons 2 variantes de la méthode : Orthomin et Orthodir. Ensuite, nous étudions le lien entre les 2 variantes, et nous prouvons un nouveau résultat de sur la vitesse de convergence de la méthode. Ce résultat est une nette amélioration du précédent résultat connu. Finalement, nous expliquons comment réduire dynamiquement le nombre de directions de recherche pendant les itérations, ceci afin de diminuer le nombre d'opérations arithmétiques de la méthode. Ces résultats théoriques sont illustrés par des résultats numériques.

2.1 Introduction

Our starting point is the work of *Grigori, Moufawad and Nataf* from [58] where *Enlarged Krylov* subspaces are introduced. They enrich the Krylov subspaces by splitting the initial residual. In particular, we focus on the *Short Recurrence Enlarged CG* (ECG) method introduced in [58] because it keeps the short recurrence property of the standard Conjugate Gradient. In fact, this method can be seen as a *Multi-parameter Descent Method* [12, 13]. The main difference lies in the approach used to construct the descent directions (or search directions) Z_{k+1} at iteration $k+1$. In [13] the authors constructed Z_{k+1} from the residual matrix R_k and the descent directions Z_k from iteration k,

$$Z_{k+1} = R_k - Z_k(Z_k^\top AZ_k)^{-1}Z_k^\top AR_k. \tag{2.1}$$

This is very close to the original formulation of Block Conjugate Gradient [89] and Conjugate Gradient [68]. In [58], the authors proceed a bit differently. First, they do not use Z_k as this for constructing the approximate solution, residual and search directions, but consider A-orthonormalizing its columns first, *e.g.*, they consider $P_k = Z_k(Z_k^\top AZ_k)^{-1/2}$. Second, they construct Z_{k+1} using P_k and P_{k-1},

$$Z_{k+1} = AP_k - P_kP_k^\top AAP_k - P_{k-1}P_{k-1}^\top AAP_k. \tag{2.2}$$

Then P_{k+1} is constructed by A-orthonormalizing Z_{k+1}. This is a slight modification of the original Block Lanczos formula which reads,

$$Z_{k+1} = AP_k - P_kP_k^\top AP_k - P_{k-1}P_{k-1}^\top AP_k, \tag{2.3}$$

in order to obtain a block of search directions Z_{k+1} which is A-orthonormal to P_i $(i \leqslant k)$ instead of being orthonormal.

2.1.1 The Block Conjugate Gradient method

First, we recall the definition of the Block CG algorithm according to *O'Leary* [89], also referred to as Orthomin [6]. Then we recall the so–called Orthodir [6] variant of the method proposed in [58] which is more stable in practice (see results in Section 2.6).

In what follows, we consider the following notations: B is a real matrix of size $n \times t$, $B^{(i)}$ is the i-th column of a matrix B, X_0 is an initial guess for the linear system with multiple right-hand sides $AX = B$, i.e., it is a real matrix of size $n \times t$. We denote the initial residual matrix $R_0 = B - AX_0$. We call t the initial block size. $\|.\|$ denotes the euclidean norm for vectors, and $\|.\|_F$ the Frobenius norm for matrices ($\|A\|_F = \sqrt{\text{trace}(A^\top A)}$).

Following *Gutknecht at al.* [65], block Krylov subspaces are defined as,

$$\mathcal{K}_k^\square(A, R_0) \equiv \text{span}^\square\left\{R_0, AR_0, \ldots, A^{k-1}R_0\right\} \tag{2.4}$$

$$\equiv \left\{\sum_{s=0}^{k-1} A^s R_0 \gamma_s \text{ such that } \forall s \in \{0, \ldots, k-1\},\ \gamma_s \in \mathbb{R}^{t\times t}\right\}. \tag{2.5}$$

When there is no ambiguity we denote $\mathcal{K}_k^\square(A, R_0)$ by $\mathcal{K}_k^\square$. Using this definition block Krylov subspaces projection methods are defined as,

$$X_k \in X_0 + \mathcal{K}_k^\square, \tag{2.6}$$

$$R_k \equiv B - AX_k \perp \mathcal{L}_k^\square, \tag{2.7}$$

where $\mathcal{L}_k^\square$ is a subspace which has the same size as $\mathcal{K}_k^\square$. The first equation (2.6) is called the subspace condition and the second one (2.7) is called the Petrov-Galerkin condition.

The Block Conjugate Gradient method is defined as the block Krylov subspaces projection method, where A is symmetric positive definite and $\mathcal{L}_k^\square = \mathcal{K}_k^\square$. As a result of this projection process,

$$\phi(X_k) = \min_{Y \in X_0 + \mathcal{K}_k^\square} \text{trace}\left(\phi(Y)\right), \tag{2.8}$$

where

$$\phi(Y) = \frac{1}{2} Y^\top A Y - B^\top Y, \tag{2.9}$$

$$\nabla\phi(Y) = AY - B. \tag{2.10}$$

As in gradient methods, the new solution at iteration k is defined as $X_k = X_{k-1} + P_k\alpha_k$. P_k is called the descent directions (or search directions) and α_k is a $t \times t$ matrix generalization of the so–called optimal step which is usually a scalar. One important property of the Block Conjugate Gradient is the A-orthogonality (or conjugacy) of the descent directions, that is $P_i^\top A P_j = 0$ when $i \neq j$. And as in Krylov methods, the search directions form a basis for the Krylov subspace, $\mathcal{K}_k^\square = \text{span}^\square\{P_1, \ldots, P_k\}$.

The standard version of Block CG defined by *O'Leary* [89] is given in Algorithm 3. This method is very similar to the one originally proposed by *Hestenes and Stiefel* [68] because it constructs Z_{k+1} (the unnormalized search directions) using R_k and P_k (equation (2.1)). Then Z_{k+1} itself is A-orthonormalized in order to construct P_{k+1}, *e.g.*, $P_{k+1} = Z_{k+1}(Z_{k+1}^\top A Z_{k+1})^{-1/2}$.

Nevertheless in practice we notice that this variant is less effective when R_k becomes rank deficient. We will discuss this in more details in Section 2.6 where we will present

numerical results illustrating this behavior in Table 2.3. Following *Dubrulle* [46] it is possible to improve this algorithm by performing a QR decomposition of the residual matrix R_k before constructing Z_{k+1} — this turns out to be very similar to Pre-CholQR (Algorithm 6) described in the following section.

Following [58], it is also possible to derive a block version of the Orthodir method defined in [6] (Algorithm 4). Unlike the previous variant, in Algorithm 4 Z_{k+1} is constructed using P_k and P_{k-1}, as in equation (2.2). This corresponds to the Block Lanczos algorithm (equation (2.3)), but with the inner product induced by A.

Algorithm 3 Block CG: orthomin

$R_0 = B - AX_0$
Compute P_1 such that $P_1^\top A P_1 = I$ (using, *e.g.*, algorithms 5, 6 or 7)
$k = 1$
while $\|R_{k-1}\|_F > \varepsilon_{\text{solver}} \|R_0\|_F$ and $k < k_{\max}$ **do**
  $\alpha_k = P_k^\top R_{k-1}$
  $X_k = X_{k-1} + P_k \alpha_k$
  $R_k = R_{k-1} - A P_k \alpha_k$
  $Z_{k+1} = R_k - P_k P_k^\top A R_k$
  Compute P_{k+1} such that $P_{k+1}^\top A P_{k+1} = I$ (using, *e.g.*, algorithms 5, 6 or 7)
  $k = k + 1$
end while

Algorithm 4 Block CG: orthodir

$R_0 = B - AX_0$
$P_0 = 0$
Compute P_1 such that $P_1^\top A P_1 = I$ (using, *e.g.*, algorithms 5, 6 or 7)
$k = 1$
while $\|R_{k-1}\|_F > \varepsilon_{\text{solver}} \|R_0\|_F$ and $k < k_{\max}$ **do**
  $\alpha_k = P_k^\top R_{k-1}$
  $X_k = X_{k-1} + P_k \alpha_k$
  $R_k = R_{k-1} - A P_k \alpha_k$
  $Z_{k+1} = A P_k - P_k P_k^\top A A P_k - P_{k-1} P_{k-1}^\top A A P_k$
  Compute P_{k+1} such that $P_{k+1}^\top A P_{k+1} = I$ (using, *e.g.*, algorithms 5, 6 or 7)
  $k = k + 1$
end while

As long as $Z_{k+1}^\top A Z_{k+1}$ is nonsingular, P_{k+1} is well defined and both Orthomin and Orthodir produce P_{k+1} that are A-orthonormal and belong to the same space. Hence, they are mathematically equivalent in exact arithmetic. Lemma 2.1.1 summarizes the main properties of the Block CG iterates.

Lemma 2.1.1

Let $k, i \in \{1, \ldots, n\}$ such that $k > i$, the Block CG iterates (Algorithms 3 and 4) verifies,

$$R_k \in \mathcal{K}_{k+1}^{\square}(A, R_0), \tag{2.11}$$
$$R_k \perp \mathcal{K}_k^{\square}(A, R_0), \tag{2.12}$$
$$P_k \in \mathcal{K}_k^{\square}(A, R_0), \tag{2.13}$$
$$AP_k \in \mathcal{K}_{k+1}^{\square}(A, R_0), \tag{2.14}$$
$$P_k \in \operatorname{span}^{\square}\{Z_k\}, \tag{2.15}$$
$$P_k^\top A P_i = 0, \tag{2.16}$$
$$P_k^\top A P_k = I, \tag{2.17}$$
$$\mathcal{K}_k^{\square}(A, R_0) = \operatorname{span}^{\square}\{P_1, \ldots, P_k\}, \tag{2.18}$$
$$P_k^\top A A P_i = 0 \text{ if } i \leqslant k-2, \tag{2.19}$$

Proof. The conditions (2.11)–(2.14) directly follow from (2.6)–(2.7) and the expression of X_k and R_k. The conditions (2.15)–(2.18) directly follows from the construction of the search directions. It follows from (2.16) and (2.18) that $\forall\ V \in \mathcal{K}_{k-1}^{\square}(A, R_0), P_k^\top A V = 0$, and in particular $P_k^\top A A P_i = 0$ if $i \leqslant k-2$. □

The situation where $Z_{k+1}^\top A Z_{k+1}$ is singular is called a breakdown. Constructing Z_{k+1} using Orthodir is twice as expensive as doing so using Orthomin, but Orthodir has the nice property to produce P_{k+1} which is more likely to be well defined (see Section 2.6).

2.1.2 A-orthonormalization algorithms

We now briefly describe different algorithms in order to A-orthonormalize a tall and skinny matrix P.

A-CholQR [81] is a generalization of CholQR to the case of an oblique inner product induced by A. Its implementation is very easy but it can break down if $P^\top AP$ is singular.

Algorithm 5 A-CholQR

Input: Z full rank, AZ
Output: $P^\top AP = I$, $Z = PR$
1: $C = Z^\top AZ$
2: Compute a Cholesky factorization of C, i.e., $C = R^\top R$ where R is upper triangular.
3: $P = ZR^{-1}$

Pre-CholQR [81] is a more robust version of A-CholQR that adds an additional QR factorization at the beginning. The difference with A-CholQR is that $\operatorname{Range}(Z) \subset \operatorname{Range}(P)$.

Algorithm 6 Pre-CholQR

Input: Z, AZ
Output: $P^\top AP = I, Z = PR$
1: Compute a QR factorization of Z, store the Q-factor as P.
2: $C = P^\top AP$
3: Compute a Cholesky factorization of C, i.e., $C = R^\top R$ where R is upper triangular.
4: $P = PR^{-1}$

In order to preserve the property $\text{Range}(Z) = \text{Range}(P)$ while avoiding break downs, it is possible to replace the QR factorization of Pre-CholQR by a rank-revealing QR factorization [71]. In that case, there might be less vectors in P than in Z but $\text{Range}(Z) = \text{Range}(P)$. We call this algorithm Breakdown-free.

Algorithm 7 Breakdown-free

Input: Z, AZ
Output: $P^\top AP = I, Z = PR$
1: Compute a rank-revealing QR factorization of Z, store the first t columns of the Q-factor as P, where t is the rank of Z.
2: $C = Q^\top AQ$
3: Compute a Cholesky factorization of C, i.e., $C = R^\top R$ where R is upper triangular.
4: $P = PR^{-1}$

2.2 The Enlarged Conjugate Gradient method

In this section, we derive two variants of the so–called Enlarged Conjugate Gradient method. Their difference lies in the way to construct the search directions.

2.2.1 Enlarged Krylov Subspaces

In [58], the authors define so–called enlarged Krylov subspaces. First, the matrix A is reordered by partitioning its graph into $\mathcal{N}$ subdomains (using METIS [76] for example). Then, the initial residual r_0 is split into t vectors denoted $R_0^{e(i)}$, $1 \leqslant i \leqslant t$. In the original paper the authors use $t = \mathcal{N}$. It is important to note that the case $t < \mathcal{N}$ can be dealt with many ways as long as $r_0 = \sum_{i=1}^{t} R_0^{e(i)}$ (Fig. 2.1). This is of particular interest in practice because typically $\mathcal{N}$ will correspond to the number of MPI processes. The parameter t is called the enlarging factor. In practice for a given t the splitting of r_0 does not have a high impact on the convergence of the method. In the numerical experiments we construct the initial enlarged residual $R_0^e = [R_0^{e(1)}, \ldots, R_0^{e(t)}]$ as the leftmost example in Fig. 2.1.

Then, the enlarged Krylov subspace of order k denoted $\mathcal{K}_{k,t}(A, r_0)$ is defined as the block Krylov subspace of order k associated to A and the enlarged residual R_0^e. More

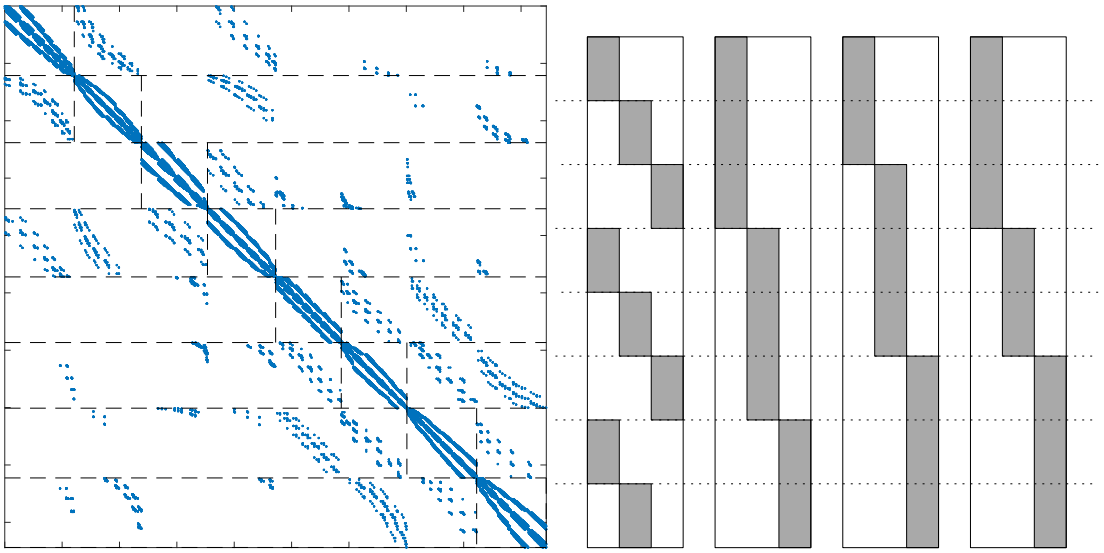


Figure 2.1 – Illustration of the ordering of A into 8 subdomains obtained with METIS [76] and several admissible splittings of r_0 into 3 vectors.

precisely, and, once again, following the notation introduced in [65],

$$\mathcal{K}_{k,t}(A, r_0) := \mathcal{K}_k^{\square}(A, R_0^e) \tag{2.20}$$

$$= \text{span}^{\square}\left\{R_0^e, AR_0^e, \ldots, A^{k-1}R_0^e\right\}. \tag{2.21}$$

2.2.2 Derivation of the method

Using these Enlarged Krylov subspaces, it is possible to derive two variants (Orthomin and Orthodir) of the enlarged Conjugate Gradient (ECG) algorithm (Algorithm 8). More precisely, the enlarged approximate solution is a matrix of size $n \times t$ denoted X_k, and the sum of its columns gives the approximate solution of the original system. We denote R_k the enlarged approximate residual, and similarly we obtain the approximate residual of the original system by summing its columns. P_k is a matrix of size $n \times t$ called search directions, it corresponds to the A-orthonormalization of Z_k. We denote α_k the optimal step, unlike in CG algorithm it is not a scalar but a matrix of size $t \times t$. Depending on the method for constructing Z_{k+1}, it is possible to derive two variants of ECG: Orthomin and Orthodir.

Orthomin (Omin) corresponds to Block CG [89]:

$$\beta_k = (AP_k)^\top R_k, \tag{2.22}$$

$$Z_{k+1} = R_k - P_k\beta_k. \tag{2.23}$$

This method is very similar to the one originally proposed by *Hestenes and Stiefel* [68]

because it constructs the new descent directions Z_{k+1} using R_k and P_k.

Orthodir (Odir) corresponds to the Block Lanczos algorithm but with the inner product induced by A:

$$\gamma_k = (AP_k)^\top (AP_k), \tag{2.24}$$
$$\rho_k = (AP_{k-1})^\top (AP_k), \tag{2.25}$$
$$Z_{k+1} = AP_k - P_k\gamma_k - P_{k-1}\rho_k. \tag{2.26}$$

It is the block equivalent of the homonym method defined in [6]. Unlike the previous variant, Z_{k+1} is constructed using P_k and P_{k-1}.

Both Orthodir and Orthomin produce Z_{k+1} that is A-orthogonal to P_i for $i \leqslant k$. Then the search directions P_{k+1} are defined as

$$P_{k+1} = Z_{k+1}(Z_{k+1}^\top AZ_{k+1})^{-1/2}. \tag{2.27}$$

Unlike CG algorithm a breakdown would occur if $Z_{k+1}^\top AZ_{k+1}$ is singular, i.e., Z_{k+1} is not full rank. Although rare this situation can happen in practice and several variants have been developed in order to handle this case [46, 71, 89]. Overall, both Orthomin and Orthodir generate P_{k+1} such that

$$P_{k+1}^\top AP_i = 0, \forall i \leqslant k \tag{2.28}$$
$$P_{k+1}^\top AP_{k+1} = I. \tag{2.29}$$

Consequently, the ECG method can be summarized in Algorithm 8. Another difference with the original block CG algorithm is that the search directions are A-orthonormalized at each iteration: P_k is used as search directions instead of Z_k. It has be shown numerically that using this variant can increase the numerical stability of the method [46].

Given a preconditioner M^{-1}, the idea for applying left preconditioning to the (block) Conjugate Gradient method is to remark that $M^{-1}A$ is self-adjoint with respect to the M-inner product [101]. Then by replacing A by $M^{-1}A$, and the transpose by ${}^\top M$ in the algorithm (Algorithm 8), it follows the preconditioned enlarged Conjugate Gradient method. In fact, some simplifications occur and the algorithm remains exactly the same except the definition of Z_k that is slightly different. More precisely, it follows that the preconditioned Orthomin method corresponds to,

$$Z_{k+1} = (I - P_kP_k^\top A)M^{-1}R_k, \tag{2.30}$$

and the preconditioned Orthodir method corresponds to,

$$Z_{k+1} = (I - P_kP_k^\top A - P_{k-1}P_{k-1}^\top A)M^{-1}AP_k. \tag{2.31}$$

In both cases, the initialization also slightly differs because $Z_1 = M^{-1}R_0^e$. Overall, the preconditioner is applied once per iteration, as in the standard CG method.

Algorithm 8 ECG algorithm.

$P_0 = 0$
$Z_1 = R_0^e$
$k = 1$
for $k = 1, \ldots, k_{\max}$ **do**
  $P_k = Z_k (Z_k^\top A Z_k)^{-1/2}$
  $\alpha_k = P_k^\top R_{k-1}$
  $X_k = X_{k-1} + P_k \alpha_k$
  $R_k = R_{k-1} - A P_k \alpha_k$
  if $\| \sum_{i=1}^{t} R_k^{(i)} \|_2 < \varepsilon$ **then**
    stop
  end if
  construct Z_{k+1} using (2.22)-(2.23) (Orthomin) or (2.24)-(2.26) (Orthodir)
end for
$x_k = \sum_{i=1}^{t} X_k^{(i)}$

2.3 Relationship between Orthodir and Orthomin

In what follows, we assume exact arithmetic and we study the connection between these two methods with the aim to derive formulas that links the approximate quantities of both variants. Indeed, by construction the approximate solutions computed by Orthodir and Orthomin are equal. Hence, the approximate residuals are also equal. But this does not imply that the search directions generated are equal even if they belong to the same space. We denote with a tilde the variables related to Orthomin and with a hat the variables related to Orthodir (Algorithm 9 and Algorithm 10). In order to simplify the presentation we consider that no breakdowns have occurred, i.e., $\widehat{Z}_k$, $\widehat{P}_k$ and $\widetilde{Z}_k$, $\widetilde{P}_k$ are all well-defined.

By construction, we know that the approximate solutions computed by Orthodir (Algorithm 10) and Orthomin (Algorithm 9) result from the same projection process. Hence, the approximate solutions are equal $\widehat{X}_k = \widetilde{X}_k$ and so are the approximate residuals $\widehat{R}_k = \widetilde{R}_k$. However this does not imply that $\widehat{Z}_{k+1} = \widetilde{Z}_{k+1}$ (and in fact we show that it is not the case).

Due to the equivalence of Orthodir and Orthomin the following lemma holds.

Lemma 2.3.1

There exists $\delta_k \in \mathbb{R}^{t \times t}$ orthogonal and such that $\widetilde{P}_k = \widehat{P}_k \delta_k$.

Proof. Orthodir and Orthomin correspond to the same projection process so $\text{span}^{\square}\left\{\widehat{P}_k\right\} = \text{span}^{\square}\left\{\widetilde{P}_k\right\}$. Hence there exists $\delta_k \in \mathbb{R}^{t \times t}$ such that $\widetilde{P}_k = \widehat{P}_k \delta_k$. Furthermore, $\widehat{X}_k = \widetilde{X}_k$ and $\widehat{R}_k = \widetilde{R}_k$ so we have

$$\widetilde{P}_k \widetilde{\alpha}_k = \widehat{P}_k \widehat{\alpha}_k. \tag{2.32}$$

Algorithm 9 Preconditioned ECG-Omin

$\widetilde{P}_0 = 0$
$\widetilde{Z}_1 = R_0^e$
$k = 1$
for $k = 1, \ldots, k_{\max}$ **do**
  $\widetilde{P}_k = \widetilde{Z}_k (\widetilde{Z}_k^\top A \widetilde{Z}_k)^{-1/2}$
  $\widetilde{\alpha}_k = \widetilde{P}_k^\top \widetilde{R}_{k-1}$
  $\widetilde{X}_k = \widetilde{X}_{k-1} + \widetilde{P}_k \alpha_k$
  $\widetilde{R}_k = \widetilde{R}_{k-1} - A\widetilde{P}_k \alpha_k$
  if $\| \sum_{i=1}^t \widetilde{R}_k^{(i)} \|_2 < \varepsilon$ **then**
    stop
  end if
  $\widetilde{Z}_{k+1} = M^{-1} \widetilde{R}_k$
  $\widetilde{\beta}_k = (A\widetilde{P}_k)^\top \widetilde{Z}_{k+1}$
  $\widetilde{Z}_{k+1} = \widetilde{Z}_{k+1} - \widetilde{P}_k \widetilde{\beta}_k$
end for
$\widetilde{x}_k = \sum_{i=1}^t \widetilde{X}_k^{(i)}$

Algorithm 10 Preconditioned ECG-Odir

$\widehat{P}_0 = 0$
$\widehat{Z}_1 = R_0^e$
$k = 1$
for $k = 1, \ldots, k_{\max}$ **do**
  $\widehat{P}_k = \widehat{Z}_k (\widehat{Z}_k^\top A \widehat{Z}_k)^{-1/2}$
  $\widehat{\alpha}_k = \widehat{P}_k^\top \widehat{R}_{k-1}$
  $\widehat{X}_k = \widehat{X}_{k-1} + \widehat{P}_k \widehat{\alpha}_k$
  $\widehat{R}_k = \widehat{R}_{k-1} - A\widehat{P}_k \widehat{\alpha}_k$
  if $\| \sum_{i=1}^t \widehat{R}_k^{(i)} \|_2 < \varepsilon$ **then**
    stop
  end if
  $\widehat{Z}_{k+1} = M^{-1} A \widehat{P}_k$
  $\widehat{\gamma}_k = (A\widehat{P}_k)^\top \widehat{Z}_{k+1}$
  $\widehat{\rho}_k = (A\widehat{P}_{k-1})^\top \widehat{Z}_{k+1}$
  $\widehat{Z}_{k+1} = \widehat{Z}_{k+1} - \widehat{P}_k \widehat{\gamma}_k - \widehat{P}_{k-1} \widehat{\rho}_k$
end for
$\widehat{x}_k = \sum_{i=1}^t \widehat{X}_k^{(i)}$

And thus,

$$\widetilde{P}_k\widetilde{P}_k^\top = \widehat{P}_k\widehat{P}_k^\top \tag{2.33}$$
$$= \widetilde{P}_k\delta_k\delta_k^\top\widetilde{P}_k^\top. \tag{2.34}$$

This implies $\delta_k\delta_k^\top = I$. Similarly $\widetilde{P}_k^\top\widetilde{P}_k = \widehat{P}_k\widehat{P}_k^\top$ and the conclusion follows. □

Using this lemma we can prove the following proposition.

Proposition 2.3.1

There exists $\delta_k \in \mathbb{R}^{t\times t}$ orthogonal and such that

$$\widetilde{Z}_{k+1} = -\widehat{Z}_{k+1}\delta_k\widetilde{\alpha}_k. \tag{2.35}$$

Proof. A simple computation using the previous relationships gives,

$$-\widehat{Z}_{k+1}\delta_k\widetilde{\alpha}_k = A\widetilde{P}_k\widetilde{\alpha}_k - \widehat{P}_k\widehat{P}_k^\top AA\widetilde{P}_k\widetilde{\alpha}_k - \widehat{P}_{k-1}\widehat{P}_{k-1}^\top AA\widetilde{P}_k\widetilde{\alpha}_k, \tag{2.36}$$
$$= A\widetilde{P}_k\widetilde{\alpha}_k - \widetilde{P}_k\widetilde{P}_k^\top AA\widetilde{P}_k\widetilde{\alpha}_k - \widetilde{P}_{k-1}\widetilde{P}_{k-1}^\top AA\widetilde{P}_k\widetilde{\alpha}_k. \tag{2.37}$$
$$\tag{2.38}$$

On the other hand, by definition of ECG we have

$$R_k - R_{k-1} = -A\widetilde{P}_k\widetilde{\alpha}_k, \tag{2.39}$$

and,

$$\widetilde{P}_{k-1}^\top AR_k = 0. \tag{2.40}$$

Hence, it follows

$$-\widehat{Z}_{k+1}\delta_k\widetilde{\alpha}_k = R_k - \widetilde{P}_k\widetilde{P}_k^\top AR_k - R_{k-1} + \widetilde{P}_{k-1}\widetilde{P}_{k-1}^\top AR_{k-1} + \widetilde{P}_k\widetilde{P}_k^\top AR_{k-1}, \tag{2.41}$$
$$= R_k - \widetilde{P}_k\widetilde{\beta}_k - \widetilde{Z}_k + \widetilde{P}_k\widetilde{P}_k^\top AR_{k-1}. \tag{2.42}$$

and furthermore,

$$\widetilde{P}_k\widetilde{P}_k^\top AR_{k-1} = \widetilde{Z}_k(\widetilde{Z}_k^\top A\widetilde{Z}_k)^{-1}\widetilde{Z}_k^\top AR_{k-1}, \tag{2.43}$$
$$= \widetilde{Z}_k(\widetilde{Z}_k^\top A\widetilde{Z}_k)^{-1}\widetilde{Z}_k^\top A\widetilde{Z}_k, \tag{2.44}$$
$$= \widetilde{Z}_k. \tag{2.45}$$

Thus,

$$\widetilde{Z}_{k+1} = -\widehat{Z}_{k+1}\delta_k\widetilde{\alpha}_k. \tag{2.46}$$

□

This result is a generalization of a previous result presented by *Ashby, Manteuf-*

fel and Saylor [6, p. 1550] for the standard CG. In fact, the authors show that $\widetilde{z}_k = \Pi_{i=0}^{k}(-\widetilde{\alpha}_k)\widehat{z}_k$ but they never consider explicitly the A-orthonormalized search directions. In particular, they define z_{k+1} using z_k (for Omin) and z_{k-1} (for Odir). This explains the slight difference between our generalization and their result.

When k becomes large, $\widetilde{\alpha}_k = \widetilde{P}_k^\top R_{k-1}$ and $\|\widetilde{\alpha}_k\|_2$ is more likely to be low because R_{k-1} is supposed to converge to 0 and $\widetilde{P}_k$ is A-orthonormalized – the same reasoning applies for $\widehat{\alpha}_k$. This result is very interesting because it shows that, since δ_k is an orthogonal matrix, when k becomes large $\|\widetilde{Z}_{k+1}\|_2$ can be significantly lower than $\|\widehat{Z}_{k+1}\|_2$. Hence, the conditioning of $\widetilde{Z}_{k+1}^\top A\widetilde{Z}_{k+1}$ could be much worse than that of $\widehat{Z}_{k+1}^\top A\widehat{Z}_{k+1}$, possibly leading to a breakdown when computing its Cholesky factorization (line 5 in Algorithms 9 and 10).

Proposition 2.3 characterizes this difference in terms of the image and rank of $\widetilde{Z}_{k+1}$ and $\widehat{Z}_{k+1}$.

Proposition 2.3.2

Let $k \geqslant 1$, and let us assume that P_{k-1} and P_k are well defined ($Z_{k-1}^\top AZ_{k-1}$ and $Z_k^\top AZ_k$ are non-singular), using either Algorithm 3 or Algorithm 4. Let $\widehat{Z}_{k+1}$ defined as,

$$\widehat{Z}_{k+1} = AP_k - P_kP_k^\top AAP_k - P_{k-1}P_{k-1}^\top AAP_k, \tag{2.47}$$

and $\widetilde{Z}_{k+1}$ defined as,

$$\widetilde{Z}_{k+1} = R_k - P_kP_k^\top AR_k, \tag{2.48}$$

then we have,

$$\operatorname{rank}\left(\widehat{Z}_{k+1}\right) = t - \dim\left(\operatorname{Range}(AP_k) \cap \operatorname{Range}\begin{pmatrix} P_{k-1} & P_k \end{pmatrix}\right), \tag{2.49}$$

$$\operatorname{rank}\left(\widetilde{Z}_{k+1}\right) = \operatorname{rank}(R_k). \tag{2.50}$$

Proof. Using Lemma 2.1.1, it is possible to rewrite $\widehat{Z}_{k+1}$ as,

$$\widehat{Z}_{k+1} = (I - W(W^\top AW)^{-1}W^\top A)AP_k, \tag{2.51}$$

where $W \equiv \begin{pmatrix} P_k & P_{k-1} \end{pmatrix}$. Furthermore, $\mathcal{P} = I - W(W^\top AW)^{-1}W^\top A$ is the matrix of the A-orthogonal projection onto $\operatorname{Range}(W)^{\perp_A}$ (see [100, p. 1915]). In particular, $\mathcal{P}$ satisfies $\mathcal{P}^\top A = A\mathcal{P} = \mathcal{P}^\top A\mathcal{P}$. Thus, we have,

$$\operatorname{Ker}\left(\widehat{Z}_{k+1}\right) = (\operatorname{Range}(AP_k) \cap \operatorname{Ker}(\mathcal{P})) \cup \operatorname{Ker}(AP_k) = \operatorname{Range}(AP_k) \cap \operatorname{Ker}(\mathcal{P}). \tag{2.52}$$

Furthermore, it is easy to show that $\operatorname{Ker}(\mathcal{P}) = \operatorname{Range}(W)$. Using the rank theorem it follows,

$$\operatorname{rank}\left(\widehat{Z}_{k+1}\right) = t - \dim(\operatorname{Range}(W) \cap \operatorname{Range}(AP_k)). \tag{2.53}$$

Similarly, $\widetilde{Z}_{k+1}$ can be rewritten as

$$\widetilde{Z}_{k+1} = \mathcal{Q}R_k, \tag{2.54}$$

where $\mathcal{Q} = I - P_k(P_k^\top AP_k)^{-1}P_k^\top A$ is the matrix of the A-orthogonal projection onto $\text{Range}\,(P_k)^{\perp_A}$. Thus, we have,

$$\text{Ker}\left(\widetilde{Z}_{k+1}\right) = (\text{Range}\,(R_k) \cap \text{Ker}\,(\mathcal{Q})) \cup \text{Ker}\,(R_k). \tag{2.55}$$

Furthermore, we have $\text{Ker}\,(\mathcal{Q}) = \text{Range}\,(P_k)$ and it follows,

$$\dim\left(\text{Ker}\left(\widetilde{Z}_{k+1}\right)\right) = \dim\left(\text{Ker}\,(R_k)\right), \tag{2.56}$$

because R_k is orthogonal to P_k. The conclusion follows using the rank theorem once again. □

Proposition 2.3 ensures that $\widehat{Z}_{k+1}$ is full rank if $\dim\left(\text{Range}\,(AP_k) \cap \text{Range}\begin{pmatrix} P_k & P_{k-1}\end{pmatrix}\right) = 0$. This assumption is needed because if it is not fulfilled it means that a part of the exact solution is already known. Indeed, if there exists a non-zero vector v such that $AP_kv \in \text{Range}\begin{pmatrix} P_k & P_{k-1}\end{pmatrix}$, this implies that $R_kv \in \text{Range}\begin{pmatrix} P_k & P_{k-1}\end{pmatrix}$ but by construction R_k is orthogonal to P_i $(1 \leqslant i \leqslant k)$ so $R_kv = 0$. However, there is not an explicit link between $\dim\left(\text{Range}\,(AP_k) \cap \text{Range}\begin{pmatrix} P_k & P_{k-1}\end{pmatrix}\right)$ and the current residual matrix R_k. Thus, once again, Orthodir seems less prone to the inexact breakdowns problem, i.e., R_k becomes numerically rank deficient [95], although this situation could happen in theory. On the other hand, it has already been proven [71, 86] that the rank of $\widetilde{Z}_{k+1}$ is equal to the rank of R_k. Thus, it is more likely that $\widetilde{Z}_{k+1}$ becomes numerically rank deficient when the method starts to converge. That is why in practice, when the A-orthonormalization of Z_{k+1} is done using A-CholQR [81] (Algorithm 5), Orthomin can break down while Orthodir does not (Table 2.3).

In order to overcome this difficulty, it is possible to use variants of A-CholQR that handle the case where Z_{k+1} is not full rank as, for example, Pre-CholQR [81] (Algorithm 6) or Breakdown-free [71] (Algorithm 7). Following *Dubrulle* [46] it is possible to improve Orthomin by performing a QR decomposition of the residual matrix R_k before constructing Z_{k+1} but we do not consider this method in this paper as it is very similar to using Pre-CholQR.

It is remarkable to notice that even for standard CG method, this has already been noticed by *Ashby, Manteuffel and Saylor* in [6, p. 1551-1552]: "If BCA is indefinite, Omin may still be used, but the previous direction vector [...] should be stored. Then, if $\widehat{\alpha_i} = 0$, control can switch to the 3-term recursion of Odir to get p_{i+1}". In practice, this phenomenon is indeed observed: there are cases where Orthomin breaks down while Orthodir does not (Table 2.3). In conclusion, Orthodir is expected to be more reliable than Orthomin. However, Orthodir is also more costly than Orthomin: the construction of Z_{k+1} requires twice as many flops and memory as for Orthomin.

2.4 Convergence study

As previously mentioned, *O'Leary* proved[89] that BCG can converge significantly faster than the standard CG. In [58] it is proved that ECG converges at least as fast as CG but there is no further informations on the speed of convergence of ECG.

This section is dedicated to the proof of the following theorem.

Theorem 2.4.1

Let x_k the approximate solution given by the Enlarged Conjugate Gradient with an enlarging factor t at step k then we have:

$$\|x_k - x_*\|_A^2 \leqslant C\left(\frac{\sqrt{\kappa_t}-1}{\sqrt{\kappa_t}+1}\right)^{2k} \tag{2.57}$$

where $\kappa_t = \dfrac{\lambda_n}{\lambda_t}$ and C is a constant independent of k whose exact expression will be given in the course of the proof.

It is a big improvement of the theorem stated in [58] because it explains that ECG's convergence is closer to that of Deflated-CG[45] rather than that of standard CG.

The key idea of the proof is to remark the close link between ECG and block CG. Following the proof of Theorem 3.2 by *Jie Chen* in [24] for block CG we adapt it to our case of interest: ECG. The technique of the proof is similar to that of *O'Leary* but the final result is a bit different. For completeness we also added another proof of a similar theorem but obtained from *O'Leary*'s technique in Annex A.1.

Proof. First, we write the error at iteration k as a polynomial of A evaluated in the initial error. Indeed,

$$x_k = x_0 + \sum_{j=1}^{t} q_{kj}(A)Ad_0^{(j)}, \tag{2.58}$$

where q_{kj} is a polynomial of degree not exceeding $k-1$.

Hence,

$$\begin{aligned} e_k &= x_k - x^* && (2.59)\\ &= x_0 + \sum_{j=1}^{t} q_{kj}(A)Ad_0^{(j)} - x^* && (2.60)\\ &= \sum_{j=1}^{t} (1 + q_{kj}(A)A)d_0^{(j)} && (2.61)\\ &= \sum_{j=1}^{t} p_{kj}(A)d_0^{(j)}, && (2.62) \end{aligned}$$

where $p_{kj}(X) = 1 + q_{kj}(X)X$ is a polynomial of degree not exceeding k and such that $p_{kj}(0) = 1$, $\forall k, j$.

Let $A = \Phi\Lambda\Phi^\top$ be the spectral decomposition of A. Let us rewrite e_k according to this decomposition,

$$e_k = \sum_{j=1}^{t} p_{kj}(A)d_0^{(j)} = \sum_{j=1}^{t} \Phi p_{kj}(\Lambda)\Phi^\top d_0^{(j)} \tag{2.63}$$

Thus,

$$\|e_k\|_A^2 = e_k^\top A e_k \tag{2.64}$$

$$= \left(\sum_{j=1}^{t} \Phi p_{kj}(\Lambda)\Phi^\top d_0^{(j)}\right)^\top A \left(\sum_{j=1}^{t} \Phi p_{kj}(\Lambda)\Phi^\top d_0^{(j)}\right) \tag{2.65}$$

$$= \left(\sum_{j=1}^{t} d_0^{(j)\top} \Phi p_{kj}(\Lambda)\right) \Lambda \left(\sum_{j=1}^{t} p_{kj}(\Lambda)\Phi^\top d_0^{(j)}\right) \tag{2.66}$$

This final expression is a generalization of the expression that occurs in the proof of convergence of the CG algorithm in [101].

Let $\widehat{j} \in \{1, \ldots, t\}$. We denote $p \equiv p_{k\hat{j}}$ in order to simplify the notations. Following Chen [24], let us define

$$-p(\Lambda)^{-1}(2I - p(\Lambda))\Phi^\top d_0^{(j)} = \begin{pmatrix} v_j \\ w_j \end{pmatrix}, \forall j \neq \widehat{j}, \tag{2.67}$$

$$\Phi^\top d_0^{(\widehat{j})} = \begin{pmatrix} f_1 \\ f_2 \end{pmatrix}, \tag{2.68}$$

where the vectors v_j, w_j, f_1, and f_2 have sizes $(t-1)\times 1$, $(n-t+1)\times 1$, $(t-1)\times 1$, $(n-t+1)\times 1$, respectively. As in [24], we denote F_1, and respectively F_2, the matrix whose columns are the v_j, and respectively the w_j. Since $v_{\widehat{j}}$ and $w_{\widehat{j}}$ are not defined, F_1 and F_2 have $t-1$ columns. Thus F_1 is a square matrix of size $(t-1)\times(t-1)$, and F_2 is of size $(n-t+1)\times(t-1)$. The $t-1$ other polynomials p_{kj} are chosen such that $p_{kj} = \tau_j(2-p)$ where the τ_j are defined as the components of the solution of $F_1\tau = f_1$.

From these definitions, it follows that for $j \neq \widehat{j}$

$$p_{kj}(\Lambda)\Phi^\top d_0^{(j)} = (2I - p(\Lambda))\Phi^\top d_0^{(j)}\tau_j \tag{2.69}$$

$$= -p(\Lambda)(-p(\Lambda)^{-1}(2I - p(\Lambda))\Phi^\top d_0^{(j)})\tau_j \tag{2.70}$$

$$= -p(\Lambda)\begin{pmatrix} F_1 \\ F_2 \end{pmatrix}\tau_j \tag{2.71}$$

Hence, using (2.67) and summing up these terms lead to

$$\sum_{j\neq\widehat{j}} p_{kj}(\Lambda)\Phi^\top d_0^{(j)} = -p(\Lambda)\begin{pmatrix} f_1 \\ F_2F_1^{-1}f_1 \end{pmatrix}. \tag{2.72}$$

Let us denote $\Lambda = \begin{pmatrix} \Lambda_1 & \\ & \Lambda_2 \end{pmatrix}$ where Λ_1 is a diagonal matrix whose coefficients are the $t-1$ smallest eigenvalues of A. We now have

$$\sum_{j=1}^{t} p_{kj}(\Lambda)\Phi^\top d_0^{(j)} = p(\Lambda)\begin{pmatrix} f_1 \\ f_2 \end{pmatrix} - p(\Lambda)\begin{pmatrix} f_1 \\ F_2F_1^{-1}f_1 \end{pmatrix} \tag{2.73}$$

$$= \begin{pmatrix} 0 & 0 \\ -p(\Lambda_2)F_2F_1^{-1} & p(\Lambda_2) \end{pmatrix}\Phi^\top d_0^{(\widehat{j})} \tag{2.74}$$

$$\equiv \begin{pmatrix} 0 & 0 \\ E & P \end{pmatrix}\Phi^\top d_0^{(\widehat{j})}. \tag{2.75}$$

Finally,

$$\|e_k\|_A^2 = {d_0^{(\widehat{j})}}^\top \Phi \begin{pmatrix} 0 & E^\top \\ 0 & P \end{pmatrix}\Lambda\begin{pmatrix} 0 & 0 \\ E & P \end{pmatrix}\Phi^\top d_0^{(\widehat{j})} \tag{2.76}$$

$$= {d_0^{(\widehat{j})}}^\top \Phi\Lambda \begin{pmatrix} 0 & E^\top \\ 0 & P \end{pmatrix}\Lambda^{1/2}\Lambda^{1/2}\begin{pmatrix} 0 & 0 \\ E & P \end{pmatrix}\Phi^\top d_0^{(\widehat{j})} \tag{2.77}$$

$$= {d_0^{(\widehat{j})}}^\top \Phi \left(\Lambda^{1/2}\begin{pmatrix} 0 & 0 \\ E & P \end{pmatrix}\right)^\top \left(\Lambda^{1/2}\begin{pmatrix} 0 & 0 \\ E & P \end{pmatrix}\right)\Phi^\top d_0^{(\widehat{j})} \tag{2.78}$$

$$\leqslant \|d_0^{(\widehat{j})}\|_A^2 \left\|\begin{pmatrix} E^\top E & E^\top P \\ PE & P^2 \end{pmatrix}\right\|. \tag{2.79}$$

Let $C \equiv \begin{pmatrix} E^\top E & E^\top P \\ PE & P^2 \end{pmatrix}$,

$$\|C\| = \left\|\begin{pmatrix} E^\top E & E^\top P \\ PE & P^2 \end{pmatrix}\right\| \tag{2.80}$$

$$= \left\|\begin{pmatrix} 0 & 0 \\ 0 & P \end{pmatrix}\begin{pmatrix} 0 & F_2F_1^{-\top} \\ 0 & I \end{pmatrix}\begin{pmatrix} 0 & 0 \\ F_2F_1^{-1} & I \end{pmatrix}\begin{pmatrix} 0 & 0 \\ 0 & P \end{pmatrix}\right\| \tag{2.81}$$

$$\leqslant \|P\|^2 \left\|\begin{pmatrix} 0 & 0 \\ F_2F_1^{-1} & I \end{pmatrix}\right\|^2 \tag{2.82}$$

$$\leqslant \|P\|^2\left(a^2+1\right) \tag{2.83}$$

where a is the largest singular value of $F_2F_1^{-1}$.

And eventually, we have

$$\|e_k\|_A^2 \leqslant \|d_0^{(\widehat{j})}\|_A^2 \left(a^2+1\right)\|P\|^2. \tag{2.84}$$

If we replace p (and therefore P) by the optimal choice we can rewrite the bound as the following min-max problem,

$$\|e_k\|_A \leqslant \|d_0^{(\widehat{j})}\|_A \sqrt{(a^2+1)} \min_{p\in\mathbb{P}_1^k} \max_{t\leqslant i\leqslant n} |p(\lambda_i)|. \tag{2.85}$$

Finally, it is possible to use Chebyshev polynomials to estimate the min-max quantity [89, 101],

$$\min_{p\in\mathbb{P}_1^k} \max_{t\leqslant i\leqslant n} |p(\lambda_i)| \leqslant 2\left(\frac{\sqrt{\kappa_t}-1}{\sqrt{\kappa_t}+1}\right)^k. \tag{2.86}$$

□

2.5 Dynamic reduction of the search directions

In this section, we introduce an approach for reducing the block size during the iterations of Orthodir and Orthomin. This is a technique also known as deflation in block Krylov methods [65, 86, 95] — however the term deflation is also used with other meanings.

2.5.1 Selection of the search directions

As explained in the survey [65] the key idea to reduce the block size is to monitor the rank of R_{k-1}. Once R_{k-1} becomes rank deficient, it means that there exists a non-zero vector v of dimensions $t\times 1$ such that

$$R_{k-1}v = 0, \tag{2.87}$$

and,

$$\begin{aligned} R_k v &= R_{k-1}v - AP_k\alpha_k v && (2.88)\\ &= 0 + AP_kP_k^\top R_{k-1}v && (2.89)\\ &= 0. && (2.90) \end{aligned}$$

It follows that $R_i v = 0$ for $i \geqslant k-1$. In other words, $X_{k-1}v$ has already converged at iteration $k-1$ because $X_{k-1}v = A^{-1}Bv$. For $i \geqslant k-1$, there exists a linear combination (independent of i) of columns of X_i denoted $X_i v$ such that $X_i v$ remains constant. As a consequence, it is possible to follow [86] and reduce effectively the sizes of X_k, R_k and P_k. But as *Langou* showed in [80], it can lead to instabilities. It is easy to see, using exactly the same reasoning, that if $\operatorname{rank}(R_{k-2})-\operatorname{rank}(R_{k-1}) = l$ then a part of the solution

of dimension l has converged. As R_{k-1} is an $n \times t$ matrix with n large, it is preferable to avoid computing the rank of R_{k-1} directly. Our approach is based on computing the rank of $\alpha_k = P_k^\top R_{k-1}$. This is similar to the idea developed by *Robbé and Sadkane* in [95]: it is preferable to work with the residual projected onto the Krylov subspace because it is smaller ($t \times t$ instead of $n \times t$) and $\text{rank}(\alpha_k) = \text{rank}(R_{k-1})$.

Proposition 2.5.1

For $k \in \{1, \ldots, n\}$, if P_k is constructed either using Algorithm 3 or Algorithm 4 and is well defined ($Z_k^\top A Z_k$ is non-singular), then $\text{rank}(\alpha_k) = \text{rank}(R_{k-1})$.

Proof. Let $v \in \text{Ker}(\alpha_k)$, then $v \in \text{Ker}(\alpha_k) \cap \text{Ker}(R_{k-1})$ or $v \in \text{Ker}(\alpha_k) \cap \text{Range}(R_{k-1})$. However, $R_k = R_{k-1} + A P_k \alpha_k$, thus $v \in \text{Ker}(\alpha_k) \cap \text{Range}(R_{k-1})$ implies that $v \in \text{Range}(R_k)$. This is not possible, unless $v = 0$, because by construction $R_k^\top R_{k-1} = 0$. Thus, $v \in \text{Ker}(\alpha_k) \cap \text{Ker}(R_{k-1}) \subset \text{Ker}(R_{k-1})$ and $\text{rank}(\alpha_k) \leqslant \text{rank}(R_{k-1})$. Reciprocally, if $v \in \text{Ker}(R_{k-1})$ then $v \in \text{Ker}(\alpha_k)$. Thus, $\text{rank}(\alpha_k) \geqslant \text{rank}(R_{k-1})$, and the conclusion follows. □

Thus $\text{rank}(\alpha_k) = \text{rank}(R_{k-1})$ and $R_i v = 0$ for $i \geqslant k-1$. It follows that $\alpha_i v = 0$ for $i \geqslant k-1$ which means that some search directions are not taken into account anymore. However, in practice this case, where α_k becomes exactly rank deficient (also denoted *exact breakdown* or *lucky breakdown*), is very rare and it is preferable to detect when α_k becomes nearly rank deficient (also denoted *inexact breakdown*) [65, 86, 95].

More precisely, we compute the singular value decomposition (SVD) of α_k,

$$\alpha_k = U_k \Sigma_k V_k^\top, \tag{2.91}$$

where U_k and V_k are orthonormal $t \times t$ matrices and $\Sigma = \text{diag}(\sigma_t, \ldots, \sigma_1)$ ($\sigma_1 \leqslant \cdots \leqslant \sigma_t$ are the singular values of α_k). If α_k is nearly rank deficient then this decomposition can be rewritten as

$$U_k \Sigma_k V_k^\top = \begin{pmatrix} U_{k,1} & U_{k,2} \end{pmatrix} \begin{pmatrix} \Sigma_{k,1} & 0 \\ 0 & \Sigma_{k,2} \end{pmatrix} \begin{pmatrix} {V_{k,1}}^\top \\ {V_{k,2}}^\top \end{pmatrix}, \tag{2.92}$$

where $\|U_{k,2} \Sigma_{k,2} {V_{k,2}}^\top\|_2 < \varepsilon_{\text{def}}$, and ε_{def} is a given tolerance. In this case $\alpha_k \approx U_{k,1} \Sigma_{k,1} {V_{k,1}}^\top$ and the idea is to replace α_k by $U_{k,1} \Sigma_{k,1} {V_{k,1}}^\top$. Hence $P_k \alpha_k \approx (P_k U_{k,1})(\Sigma_{k,1} {V_{k,1}}^\top)$.

Theorem 2.5.1

For $k \in \{1, \ldots, n\}$, let us assume that X_k, and R_k are constructed either using Algorithm 3 or Algorithm 4. We further assume that α_k has a SVD such that $\alpha_k = \begin{pmatrix} U_{k,1} & U_{k,2} \end{pmatrix} \begin{pmatrix} \Sigma_{k,1} & 0 \\ 0 & \Sigma_{k,2} \end{pmatrix} \begin{pmatrix} {V_{k,1}}^\top \\ {V_{k,2}}^\top \end{pmatrix}$ and $\|U_{k,2} \Sigma_{k,2} {V_{k,2}}^\top\|_2 < \varepsilon_{\text{def}}$. Then we have,

$$\|E_k v - E_{k-1} v\|_A \leqslant \varepsilon_{\text{def}}, \tag{2.93}$$

$$\|R_k v - R_{k-1} v\|_{A^{-1}} \leqslant \varepsilon_{\text{def}}, \tag{2.94}$$

where v is a column of $V_{k,2}$, and $E_k = X_* - X_k$ denotes the enlarged error, i.e., X_* is such that $AX_* = T(b)$.

Proof. Let us assume that v is the j^{th} column of $V_{k,2}$. We denote σ_j the corresponding singular value, it corresponds to the j^{th} diagonal value in $\Sigma_{k,2}$. We have,

$$\begin{aligned}
\|E_k\, v - E_{k-1}\, v\|_A &= \|X_{k-1}\, v + P_k \alpha_k\, v - X_{k-1}\, v\|_A && (2.95)\\
&= \|P_k \alpha_k\, v\|_A && (2.96)\\
&= \sqrt{v^\top \alpha_k^\top {P_k}^\top A P_k \alpha_k v} && (2.97)\\
&= \sqrt{v^\top \alpha_k^\top \alpha_k v} && (2.98)\\
&= \sqrt{v^\top V_k \Sigma_k^2 V_k^\top v} && (2.99)\\
&= \sigma_j && (2.100)\\
&\leqslant \varepsilon_{\text{def}}. && (2.101)
\end{aligned}$$

Similarly,

$$\begin{aligned}
\|R_k\, v - R_{k-1}\, v\|_{A^{-1}} &= \|R_{k-1}\, v + A P_k \alpha_k\, v - R_{k-1}\, v\|_{A^{-1}} && (2.102)\\
&= \|A P_k \alpha_k v\|_{A^{-1}} && (2.103)\\
&= \|P_k \alpha_k v\|_A && (2.104)\\
&\leqslant \varepsilon_{\text{def}}. && (2.105)
\end{aligned}$$

□

Now let us define,

$$\begin{aligned}
P_{k,1} &= P_k U_{k,1}, && (2.106)\\
P_{k,2} &= P_k U_{k,2}, && (2.107)\\
& && (2.108)
\end{aligned}$$

Since the part of the solution corresponding to $X_k V_{k,2}$ has almost converged (in the A-norm), the search directions $P_{k,2}$ are not needed anymore. Hence we define the new search directions denoted $Z_{k+1,1}$ as,

$$\begin{aligned}
Z_{k+1,1} \equiv Z_{k+1} U_{k,1} \quad &= \quad AP_{k,1} - \begin{pmatrix} P_{k,1} & P_{k,2} \end{pmatrix} \begin{pmatrix} {P_{k,1}}^\top \\ {P_{k,2}}^\top \end{pmatrix} AAP_{k,1} && (2.109)\\
&\quad -P_{k-1} P_{k-1}^\top AAP_{k,1} \\
&= \quad AP_{k,1} - P_{k,1} {P_{k,1}}^\top AAP_{k,1} && (2.110)\\
&\quad -P_{k,2} {P_{k,2}}^\top AAP_{k,1} - P_{k-1} P_{k-1}^\top AAP_{k,1}
\end{aligned}$$

and $Z_{k+1,1}$ have a smaller size than Z_{k+1}. Furthermore, by construction $Z_{k+1,1}$ belongs to $\text{span}^{\square}\{P_1,\ldots,P_{k-1},P_k,AP_{k,1}\}$ and is A-orthogonal to $P_1,\ldots,P_{k-1},P_{k,1},P_{k,2}$. Indeed, Z_{k+1} is constructed by A-orthogonalizing AP_k which belongs to $\text{span}^{\square}\{P_{k-1},P_k,AP_k\}$ against $P_1,\ldots,P_{k-1},P_{k,1},P_{k,2}$. As $U_{k,1}$ is an orthogonal matrix, it follows that $Z_{k+1}U_{k,1}$ belongs to $\text{span}^{\square}\{P_{k-1},P_{k,1},P_{k,2},AP_{k,1}\}$ and is also A-orthogonal to $P_1,\ldots,P_{k-1},P_{k,1},P_{k,2}$. Finally, we define P_{k+1} by A-orthonormalizing $Z_{k+1,1}$.

This allows us to define the new search directions P_{k+1} the first time we reduce the size of the block but we need to generalize this idea when the size is reduced several times (possibly until it is equal to one).

We denote P_i the search directions at the beginning of the iteration i (now P_i can possibly have less than t columns), and $P_{i,1} = P_iU_{i,1}$ denotes the search directions that are used at iteration i if α_i is nearly rank deficient (and $P_{i,2} = P_iU_{i,2}$). We denote H the matrix whose columns are the removed search directions $P_{i,2} = P_iU_{i,2}$ $(1 \leqslant i \leqslant k)$. It is important to note that if α_k is full rank, then $P_{k,2}$ is empty. Thus the number of columns of H is at most $t-1$ where t is the initial block size. Indeed, the final block size is at least 1 which means that the columns of H are the $t-1$ search directions removed.

We want to construct $Z_{k+1,1}$ such that:

$$Z_{k+1,1}^{\top}AP_{i,d} = 0,\ i \leqslant k,\ d \in \{1,2\}, \tag{2.111}$$

and $Z_{k+1,1} \in \text{span}^{\square}\{P_{1,1},\ldots,P_{k-1,1},P_{k,1},H,AP_{k,1}\}$ where H is the matrix whose columns are the removed search directions $P_i,2$ $(1 \leqslant i \leqslant k)$. One way to construct $Z_{k+1,1}$ such that (2.111) is fulfilled is to use Algorithm 11. Indeed, Proposition 2.5.1 ensures that this algorithm verifies (2.111).

Proposition 2.5.2

Let $P_{k-1,1}$, $P_{k,1}$, H and $Z_{k+1,1}$ defined as in Algorithm 11, i.e.,

$$\begin{aligned} Z_{k+1,1} \quad = \quad & AP_{k,1} - P_{k,1}P_{k,1}^{\top}AAP_{k,1} \\ & -P_{k-1,1}P_{k-1,1}^{\top}AAP_{k,1} - HH^{\top}AAP_{k,1}. \end{aligned} \tag{2.112}$$

Then $Z_{k+1,1}$ is such that,

$$Z_{k+1,1}^{\top}AP_{i,d} = 0,\ i \leqslant k,\ d \in \{1,2\}. \tag{2.113}$$

Proof. The proof is based on induction on k. In order to ease the computations we consider $(Z_{k+1,1}^{\top}AP_{i,d})^{\top} = P_{i,d}^{\top}AZ_{k+1,1}$, i.e., we show that $P_{i,d}^{\top}AZ_{k+1,1} = 0$, $i \leqslant k$ and $d \in \{1,2\}$.

- If $k \in \{0,1\}$ the descent directions $Z_{k,1}$ are the same as in the usual Block Conjugate Gradient and (2.113) is true.
- If $k \in \{2,\ldots,n\}$ we assume that (2.113) is true for k. In particular, it means that one can replace $Z_{k,1}$ by P_k in (2.113) because P_k is constructed by A-orthonormalizing

$Z_{k,1}$. Also, given $l \in \{1,\dots,k\}$ we have defined $P_{l,1} = P_l U_{l,1}$ and $P_{l,2} = P_l U_{l,2}$, thus if $P_l^\top AV = 0$ with V a matrix with appropriate dimensions, then $P_{l,1}^\top AV = 0$, and $P_{l,2}^\top AV = 0$. We want to point out that the case where the block size is reduced for the first time at iteration k, i.e., $P_{k-1} = P_{k-1,1}$ and $H = P_{k,2}$ has been discussed previously in details (see (2.110) and the discussion thereafter). We want to prove that (2.113) remains true for $Z_{k+1,1}$ whose definition is more general. For all $i \leqslant k$, $d \in \{1,2\}$ we have:

$$\begin{aligned} P_{i,d}^\top AZ_{k+1,1} &= P_{i,d}^\top AAP_{k,1} && (2.114)\\ &- P_{i,d}^\top AP_{k,1}P_{k,1}^\top AAP_{k,1} && (2.115)\\ &- P_{i,d}^\top AP_{k-1,1}P_{k-1,1}^\top AAP_{k,1} && (2.116)\\ &- P_{i,d}^\top AHH^\top AAP_{k,1} && (2.117) \end{aligned}$$

- If $d = 1$ then (2.117) is vanishing because given $l \in \{1,\dots,k\}$ $P_{i,1}^\top AP_{l,2} = 0$: whether because $i \neq l$ and it directly follows from the induction hypothesis, or they are equal and $P_{i,1}^\top AP_{i,2} = U_{k,1}^\top U_{k,2} = 0$. If $i = k$ then (2.116) is equal to zero and (2.115) can be rewritten as:

$$P_{k,1}^\top AP_{k,1}P_{k,1}^\top AAP_{k,1} = P_{k,1}^\top AAP_{k,1}. \tag{2.118}$$

  So (2.114) = (2.115) and it follows that (2.113) is true. The same reasoning holds if $i = k-1$. If $i \leqslant k-2$ then (2.115) and (2.116) are both equal to zero. It remains to show that (2.114) is equal to zero. Using the hypothesis, we have that P_k is A-orthogonal to span$^\square\{P_1,\dots,P_{k-1}\}$ so $P_i^\top AAP_{k,1} = 0$, leading to $P_{i,d}^\top AAP_{k,1} = 0$. Hence (2.113) is true.

- If $d = 2$, (2.25) is vanishing because given $l \in \{1,\dots,k\}$ $P_{i,2}^\top AP_{k,1} = 0$: whether because $i \neq k$ and it directly follows from the induction hypothesis, or they are equal and $P_{k,2}^\top AP_{k,1} = U_{k,2}^\top U_{k,1} = 0$. The same reasoning holds in order to show that (2.116) vanishes. Finally, (2.117) simplifies:

$$\begin{aligned} P_{i,2}^\top AHH^\top AAP_{k,1} &= P_{i,2}^\top AP_{i,2}P_{i,2}^\top AAP_{k,1}, && (2.119)\\ &= P_{i,2}^\top AAP_{k,1}. && (2.120) \end{aligned}$$

  Indeed, for any $l \in \{1,\dots,k\}$, $P_{i,2}^\top AP_{l,2} = 0$ if $i \neq l$ because $P_i^\top AP_l = 0$ using the hypothesis, and similarly $P_{i,2}^\top AP_{i,2} = I$. So (2.114) = (2.117) and it follows that (2.113) is true.

□

Using Proposition 2.5.1 leads to a variant of Orthodir where the number of search directions can be reduced dynamically during the iterations (Algorithm 11).

Following *Ji and Li* [71], it is possible to derive a similar algorithm when the search directions are constructed using Orthomin (Algorithm 12). If $\alpha_k = P_k^\top R_{k-1}$ is nearly

Algorithm 11 Orthodir with block size reduction

Input: A, B, X_0, $k_{\max}$, $\varepsilon_{\text{solver}}$, ε_{def}
Output: $\|R_{k-1}\|_F < \varepsilon_{\text{solver}}\|R_0\|_F$ or $k = k_{\max}$

$R_0 = B - AX_0$
$P_0 = 0$
Compute P_1, S_1 such that $P_1^\top AP_1 = I$ and $R_0 = P_1 S_1$ (using, *e.g.*, algorithms 5, 6 or 7)
s_0 = number of columns of P_1
$k = 1$
$H = []$ (empty matrix)
while $\|R_{k-1}\|_F < \varepsilon_{\text{solver}}\|R_0\|_F$ and $k < k_{\max}$ **do**
  $\alpha_k = P_k^\top R_{k-1}$
  $[U_k, \Sigma_k, V_k] = \text{svd}(\alpha_k)$
  s_k = number of singular values of α_k larger than ε_{def}
  if $s_k < s_{k-1}$ **then**
    $U_{k,1} = U_k(:, 1 : s_k)$
    $U_{k,2} = U_k(:, s_k + 1 : end)$
    $\Sigma_{k,1} = \Sigma_k(1 : s_k, 1 : s_k)$
    $V_{k,1} = V_k(:, 1 : s_k)$
    $P_{k,2} = P_k U_{k,2}$
    $H = [H, P_{k,2}]$
    $\alpha_k = \Sigma_{k,1} {V_{k,1}}^\top$
    $P_{k,1} = P_k U_{k,1}$
  end if
  $X_k = X_{k-1} + P_{k,1}\alpha_k$
  $R_k = R_{k-1} - AP_{k,1}\alpha_k$
  $Z_{k+1,1} = AP_{k,1} - P_{k,1}P_{k,1}^\top AAP_{k,1} - P_{k-1,1}P_{k-1,1}^\top AAP_{k,1} - HH^\top AAP_{k,1}$
  Compute P_{k+1}, S_{k+1} such that $P_{k+1}^\top AP_{k+1} = I$ and $Z_{k+1,1} = P_{k+1}S_{k+1}$ (using, *e.g.*, algorithms 5, 6 or 7)
  $k = k + 1$
end while

rank deficient then we replace it by its low rank approximation, i.e., $\alpha_k = U_{k,1}\Sigma_{k,1}V_{k,1}^\top$. Similarly we define $P_{k,1} = P_k U_{k,1}$ and $P_{k,2} = P_k U_{k,2}$. If the size is not reduced then $P_{k,2}$ is empty. Finally, $Z_{k+1,1}$ is constructed as,

$$Z_{k+1,1} = R_k - P_{k,1}^\top P_{k,1} A R_k, \tag{2.121}$$

and then $Z_{k+1,1}$ is A-orthonormalized in order to construct P_{k+1}. This ensures that the so–called search spaces (the search space $\mathcal{P}_i$ is defined as $\text{Range}((\,)P_{i,1})$) are conjugate [71, Theorem 3.2]. This is a bit weaker than (2.111) because it means that $Z_{k+1,1}$ verifies,

$$Z_{k+1,1}^\top A P_{i,1} = 0, 1 \leqslant i \leqslant k, \tag{2.122}$$

and $Z_{k+1,1} \in \text{span}^\square\{P_{1,1}, \ldots, P_{k,1}, R_{k,1}\}$.

In particular, this does not ensure that $Z_{k+1,1}^\top A P_{i,2} = 0$ for $i \leqslant k$. Nevertheless, we perform numerical experiments using Algorithm 12 in order to compare it with Algorithm 11. These tests suggest that ensuring the conjugacy of the search spaces, as in [71], is less robust in practice than ensuring (2.111) (see Section 2.6). However, ensuring (2.111) would mean to keep the removed search directions in H, and then A-orthogonalize $Z_{k+1,1}$ against H at each iteration. Thus the overall cost would be higher than Orthomin without reduction of the search directions which makes such approach not practical. Indeed, the main drawback is that $Z_{k+1,1}$ always has t columns because it is constructed from R_k, and reducing the size of R_k can lead to more numerical instabilities [71, 80].

2.5.2 Choice of the tolerance

In this section we study the choice of the deflation tolerance in Algorithms 11 and 12. Following [101], we denote $\rho(A)$ the spectral radius of a square matrix A, i.e., the maximum modulus of its eigenvalues. In what follows, we need the following definition.

Definition 2.5.1

Let V be any real matrix of size $n \times t$ and A be any real symmetric positive definite matrix of size $n \times n$, we denote

$$\|V\|_F = \sqrt{\text{trace}(V^\top V)}, \tag{2.123}$$

$$\|V\|_2 = \sqrt{\rho(V^\top V)}, \tag{2.124}$$

$$\|V\|_A = \sqrt{\rho(V^\top A V)}. \tag{2.125}$$

Both $\|.\|_F$ and $\|.\|_2$ are well-known matrix norms [101]. It is easy to check that $\|V\|_A$ is nothing but the subordinate norm $\|.\|_{A,2}$,

$$\|V\|_A = \sup_{x \neq 0} \frac{\|Vx\|_A}{\|x\|_2}. \tag{2.126}$$

Algorithm 12 Orthomin with block size reduction

Input: A, B, X_0, $k_{\max}$, $\varepsilon_{\text{solver}}$, ε_{def}
Output: $\|R_{k-1}\|_F < \varepsilon_{\text{solver}}\|R_0\|_F$ or $k = k_{\max}$

$R_0 = B - AX_0$
Compute P_1, S_1 such that $P_1^\top AP_1 = I$ and $R_0 = P_1 S_1$ (using, *e.g.*, algorithms 5, 6 or 7)
$s_0 =$ number of columns of P_1
$H = []$ (empty matrix)
$k = 1$
while $\|R_{k-1}\|_F < \varepsilon_{\text{solver}}\|R_0\|_F$ and $k < k_{\max}$ **do**
  $\alpha_k = P_k^\top R_{k-1}$
  $[U_k, \Sigma_k, V_k] = \text{svd}(\alpha_k)$
  $s_k =$ number of singular values of α_k larger than ε_{def}
  if $s_k < s_{k-1}$ **then**
    $U_{k,1} = U_k(:, 1 : s_k)$
    $\Sigma_{k,1} = \Sigma_k(1 : s_k, 1 : s_k)$
    $V_{k,1} = V_k(:, 1 : s_k)$
    $\alpha_k = \Sigma_{k,1} {V_{k,1}}^\top$
    $P_{k,1} = P_k U_{k,1}$
  end if
  $X_k = X_{k-1} + P_{k,1}\alpha_k$
  $R_k = R_{k-1} - AP_{k,1}\alpha_k$
  $Z_{k+1,1} = R_k - P_{k,1}^\top P_{k,1} AR_k$
  Compute P_{k+1}, S_{k+1} such that $P_{k+1}^\top AP_{k+1} = I$ and $Z_{k+1,1} = P_{k+1}S_{k+1}$ (using, *e.g.*, algorithms 5, 6 or 7)
  $k = k + 1$
end while

Let us recall that in [96] the authors show that if P is A-orthonormal then $\|P\|_2 \leqslant \|A^{-1}\|_2^{1/2}$. It is interesting to note that

$$\begin{aligned} \|\alpha_{k+1}\|_2 &= \|P_{k+1}^\top R_k\|_2 && (2.127)\\ &\leqslant \|P_{k+1}\|_2 \|R_k\|_2 && (2.128)\\ &\leqslant \|A^{-1}\|_2^{1/2} \|R_k\|_2 && (2.129)\\ &\leqslant \|A^{-1}\|_2^{1/2} \|R_k\|_F. && (2.130) \end{aligned}$$

Hence, the following proposition holds.

Proposition 2.5.3

Let us denote s_k the number of columns of $P_{k,1}$. If we choose ε_{def} such that

$$\varepsilon_{\text{def}} \leqslant \|A^{-1}\|_2^{1/2} \varepsilon_{\text{solver}} \|R_0\|_F, \tag{2.131}$$

Then, let k be in $\{1,\dots,n\}$, if $s_{k+1} = 0$ then the current approximate solution satisfies,

$$\|E_k - E_{k-1}\|_2 \leqslant \varepsilon_{\text{solver}} \|R_0\|_F, \tag{2.132}$$

or in the A-norm,

$$\|E_k - E_{k-1}\|_A \leqslant \|A^{-1}\|_2^{1/2} \varepsilon_{\text{solver}} \|R_0\|_F, \tag{2.133}$$

where E_k denotes the error at iteration k. In other words, if $s_k = 0$ the (full) error has almost converged, and the method could be stopped.

Proof. Let $k \in \{1,\dots,n\}$,

$$\begin{aligned} \|E_k - E_{k-1}\|_A &= \|P_k \alpha_k\|_A, && (2.134)\\ &= \|\alpha_k\|_2. && (2.135) \end{aligned}$$

If $s_k = 0$ it means that

$$\begin{aligned} \|E_k - E_{k-1}\|_A &= \|\alpha_k\|_2, && (2.136)\\ &\leqslant \|A^{-1}\|_2^{1/2} \varepsilon_{\text{solver}} \|R_0\|_F. && (2.137) \end{aligned}$$

On the other hand,

$$\|E_k - E_{k-1}\|_A \geqslant \|A^{-1}\|_2^{1/2} \|E_k - E_{k-1}\|_2. \tag{2.138}$$

Hence,

$$\|E_k - E_{k-1}\|_2 \leqslant \varepsilon_{\text{solver}} \|R_0\|_F \tag{2.139}$$

□

Proposition 2.5.4

In Algorithms 11 and 12, if (2.131) is verified, the number of search directions used at iteration k, denoted s_k, is decreasing, i.e., $0 \leqslant s_{k+1} \leqslant s_k \leqslant t \ \forall k \in \{1,\dots,n\}$.

Proof. Using Theorem 2.5.1, $X_k v$ has already converged when removing the search directions related to v for all the following iterations. The size s_k corresponds to t minus the dimension of the space generated by such vectors v. This space will necessarily grow during the iterations. When $s_k = 0$, it is possible to stop the algorithm according to Proposition 2.5.2. □

Remark 2.5.1

The space $\mathcal{K}_k^{\triangle} = \text{span}^{\square}\{P_{1,1},\dots,P_{k,1},H,AP_{k,1}\}$ which corresponds to the search space for the solution when reducing the number of search directions, is not exactly the same as $\mathcal{K}_k^{\square}$, corresponding to the original enlarged Krylov subspace. In fact, it is smaller because some of the search directions have been removed. Due to round-off errors during orthonormalization, one can expect that the dynamic methods will need (hopefully not many) more iterations in order to converge.

Remark 2.5.2

When using the Enlarged Conjugate Gradient to solve a linear system with a single right-hand side, we are interested in,

$$\|e_k - e_{k-1}\|_A = \|(E_k - E_{k-1})\mathbb{1}_t^\top\|_A \tag{2.140}$$
$$= \|\alpha_k \mathbb{1}_t^\top\|_2 \tag{2.141}$$
$$\leqslant \sqrt{t}\|\alpha_k\|_2. \tag{2.142}$$

Hence if,

$$\|\alpha_k\| \leqslant \frac{1}{\sqrt{t}}\|A^{-1}\|_2^{1/2}\varepsilon_{\text{solver}}\|r_0\|, \tag{2.143}$$

it is possible to use Proposition 2.5.2 and conclude that if $s_k = 0$,

$$\|e_k - e_{k-1}\|_2 \leqslant \varepsilon_{\text{solver}}\|r_0\|. \tag{2.144}$$

Remark 2.5.3

In practice, it is too costly to compute the optimal criterion in Proposition 2.5.2. But we make the assumption that

$$\|A^{-1}\|_2^{1/2} \geqslant 1, \tag{2.145}$$

in order to neglect this term.

In all the numerical experiments, the criterion for removing the number of search directions is $\varepsilon_{\text{def}} = \frac{1}{\sqrt{t}}\varepsilon_{\text{solver}}\|r_0\|_2$ (and if $s_k = 0$ the method is stopped).

2.6 Numerical experiments

All the results are obtained with Matlab R2015b. Matlab Preconditioned Conjugate Gradient, denoted PCG, is used as a reference method. We always use a split block Jacobi preconditioner and we refer to nj as the number of diagonal blocks. The right-hand side is chosen uniformly random and normalized, and the initial guess is set to 0.

2.6.1 Test cases

We compare the performance of the different methods on a set of matrices that are also used in [2, 58, 87] where they are described in more details. These matrices are displayed in Table 2.1 where we present their size, the number of non-zeros, their smallest and largest eigenvalues. The matrices NH2D, SKY2D, SKY3D and ANI3D arise from boundary value problem of the diffusion equations:

$$-\text{div}(\kappa(x)\nabla u) = f \qquad \text{on } \Omega \tag{2.146}$$

$$u = 0 \qquad \text{on } \partial\Omega_D \tag{2.147}$$

$$\frac{\partial u}{\partial n} = 0 \qquad \text{on } \partial\Omega_N \tag{2.148}$$

where Ω is the unit square (2D) or cube (3D). The Dirichlet boundary is denoted $\partial\Omega_D$, and $\partial\Omega_N$ is the Neumann boundary. The right-hand side f represents some body force, and u is the unknown displacement field. The tensor κ is a given coefficient of the partial differential operator. In the 2D case, $\partial\Omega_D = [0,1]\times\{0,1\}$ and in the 3D case, $\partial\Omega_D = [0,1]\times\{0,1\}\times[0,1]$. In both cases, $\partial\Omega_N$ is chosen as $\partial\Omega_N = \partial\Omega \setminus \partial\Omega_D$.

The matrix NH2D is obtained by considering a non-homogeneous problem with large jumps in the coefficients of κ. The tensor κ is isotropic and discontinuous, it jumps from the constant value 10^3 in the ring $\frac{1}{2\sqrt{2}} \leqslant |x-c| \leqslant \frac{1}{2}$ with $c = (\frac{1}{2},\frac{1}{2})^\top$, to 0 outside.

The matrices SKY2D and SKY3D are obtained by considering skyscraper problems where the domain Ω contains many zones of high permeability which are isolated from each other. More precisely κ is taken as:

$$\kappa(x) = 10^3 \times ([10 \times x_2] + 1) \qquad \text{if } [10x_i] \text{ is odd, } i = \{1,2\} \tag{2.149}$$

$$\kappa(x) = 1 \qquad \text{otherwise,} \tag{2.150}$$

where $[x]$ is the integer value of x.

Table 2.1 – The test matrices, their order, the number of non-zeros, their smallest ($\lambda_{\min}$) and largest ($\lambda_{\min}$) eigenvalues; if they are coming from 2D or 3D discretization and the type of problem they are coming from.

	Order	Nonzeros	$\lambda_{\min}$	$\lambda_{\max}$	2D/3D	Problem
NH2D	10 000	49 600	1.9e-3	8.0	2D	Boundary Value
SKY2D	10 000	49 600	3.5e-3	7.0e4	2D	Skyscraper
SKY3D	8 000	53 600	5.3e-3	3.0e3	3D	Skyscraper
ANI3D	8 000	53 600	6.7e-7	1.4	3D	Anisotropic Layers
Ela25	9 438	312 372	2.7e-5	3.4	3D	Linear Elasticity P1 FE
Ela50	18 153	618 747	1.9e-6	3.4	3D	Linear Elasticity P1 FE
Ela100	36 663	1 231 497	2.6e-7	2.4	3D	Linear Elasticity P1 FE
Ela2D200	80 802	964 800	2.8e-8	3.7	2D	Linear Elasticity P1 FE

The matrix ANI3D is obtained by considering anisotropic layers: the domain Ω is made of 10 anisotropic layers with jumps of up to four orders of magnitude and an anisotropy ratio of 10^3 in each layer. These layers are parallel to $z = 0$, of size 0.1, and inside them the coefficients are constant: $\kappa_y = 10\kappa_x$, $\kappa_z = 100\kappa_x$.

All these problems are discretized on cartesian grids, of size 100×100 for the 2D problems and of size $20 \times 20 \times 20$ for the 3D problems.

The Ela matrices arise from the linear elasticity problem with Dirichlet and Neumann boundary conditions defined as follows

$$-\mathrm{div}(\sigma(u)) = f \qquad \text{on } \Omega \tag{2.151}$$

$$u = 0 \qquad \text{on } \partial\Omega_D \tag{2.152}$$

$$\sigma(u) \cdot n = 0 \qquad \text{on } \partial\Omega_N \tag{2.153}$$

where Ω is a unit square (2D) or cube (3D). The matrices Ela$N$ correspond to this equation discretized using a triangular mesh with $N \times 10 \times 10$ points on the corresponding vertices. The matrices Ela2DN correspond to this equation discretized using a triangular mesh with $N \times N$ points on the corresponding vertices. Once again, the Dirichlet boundary is denoted $\partial\Omega_D$, and $\partial\Omega_N$ is the Neumann boundary; the right-hand side f represents some body force, and u is the unknown displacement field. The Cauchy stress tensor is denoted $\sigma(.)$, it is given by Hooke's law: it can be expressed in terms of Young's Modulus E and Poisson's ratio ν. For a more detailed description of the problem see [62, 74]. We consider discontinuous E and ν in 3D: $(E_1, \nu_1) = (2 \times 10^{11}, 0.25)$ and $(E_2, \nu_2) = (10^7, 0.45)$; and discontinuous E in 2D: $(E_1, \nu_1) = (10^{12}, 0.45)$ and $(E_2, \nu_2) = (2 \times 10^6, 0.45)$. All these matrices are scaled in order to reduce the effect of possibly very high values on the diagonal.

Table 2.2 – Numerical parameters we consider in our tests, the size of the initial block (which is also the number of right-hand sides), the number of diagonal blocks nj in the block diagonal preconditioner, the variant we use between Orthomin and Orthodir, the definition of the error and the number of iterations..

t	block size
nj	number of block Jacobi
Odir	Orthodir
Omin	Orthomin
er	$\frac{\|x^*-x_k\|_2}{\|x^*\|_2}$ with x^* the exact solution (computed with Matlab LU)
iter	number of iterations until convergence

2.6.2 Influence of the parameters and algorithmic variants

We now study the influence of the parameters and algorithmic variants, such as the construction of Z_{k+1}, the A-orthogonalization scheme for constructing Z_{k+1}, the choice of the enlarging factor t, the construction of the initial block residual R_0; on the overall convergence of the method.

In Table 2.3, we summarize the results obtained when running Orthomin (Algorithm 3) and Orthodir (Algorithm 4) on the same matrix for BRRHS-CG [86], Coop-CG [9], ECG [58] and the standard PCG method. The initial block size is set to 16, and we use a block Jacobi preconditioner with 64 blocks. As these methods have different stopping criteria, we decided to use the norm of the normalized error, computed with respect to the solution computed by the LU factorization. Although this stopping criterion is usually not available in real-life problems, it allows us to compare fairly the iteration counts of the different methods. We compare different algorithms for A-orthonormalizing Z_{k+1}. ACHQR stands for A-CholQR (Algorithm 5), PreCHQR stands for Pre-CholQR (Algorithm 6) and BF stands for Breakdown-free (Algorithm 7). We observe that Orthomin and Orthodir are equivalent on the simplest test cases (NH2D, ANI3D), i.e., where PCG is converging in a relatively low number of iterations. However, we observe that breakdowns occur with Omin(ACHQR) when the matrix becomes more ill-conditioned (SKY2D, Ela50 and Ela100). This does not happen when considering Odir(ACHQR). The matrix SKY3D is an intermediate case where no breakdown occurs with Omin(ACHQR) but the number of iterations can significantly increase depending on the method. Using Pre-CholQR cures the breakdowns with Orthomin for all the matrices we considered, and the resulting number of iterations is the same, or around the same, as with Odir(ACHQR). On the other hand Breakdown-free is less stable, and we observe that the number of iterations can significantly increase (SKY2D and Ela100 with Coop-CG). We conclude that Orthodir is more stable than Orthomin. In particular, it never breaks down for the test cases we considered. On the other hand,

Table 2.3 – Number of iterations to get the solution ($\varepsilon_{\text{solver}} = 10^{-8}$), the stopping criterion is the error, i.e., $\frac{\|x^*-x_k\|_2}{\|x^*\|_2}$ where x^* is the exact solution computed with LU (- means that a breakdown occurred). The number of blocks in the Block Jacobi preconditioner (nj) is set to 64, and the block size t is set to 16.

			NH2D	SKY2D	SKY3D	ANI3D	Ela50	Ela100
PCG			97	385	313	77	485	486
BRRHS-CG	Odir	ACHQR	38	49	51	64	87	66
	Omin	ACHQR	38	-	52	64	-	-
		PreCHQR	38	39	49	64	87	66
		BF	38	39	49	64	87	66
Coop-CG	Odir	ACHQR	41	55	55	64	101	81
	Omin	ACHQR	41	-	130	64	-	-
		PreCHQR	41	80	55	64	101	132
		BF	41	137	55	64	101	176
ECG	Odir	ACHQR	34	40	50	61	90	65
	Omin	ACHQR	34	-	50	61	-	-
		PreCHQR	34	40	50	61	90	65
		BF	34	40	50	61	90	65

when using Orthomin it is crucial to consider the case where a breakdown occurs when A-orthonormalizing Z_{k+1}. This is why in the following experiments we always use Odir(ACHQR) and Omin(PreCHQR).

The results presented in Table 2.4 compare the standard PCG, BRRHS-CG, and ECG on several matrices from our test set. Indeed, as observed in the previous experiment, the number of iterations with Coop-CG is always higher than with BRRHS-CG and ECG. As in the previous experiment, we use the norm of the error as stopping criterion for all the methods and we report the number of iterations needed to achieve an accuracy of 10^{-8}. In the three leftmost columns we present the results obtained when we increase both the number of block Jacobi nj and the block size t while keeping their ratio constant. As the number of block Jacobi increases, the preconditioner becomes less effective. Our goal is to compensate this by an increase of the block size so that the number of iterations does not increase or even decreases. The corresponding results are summarized in the first columns of Table 2.4. In all the cases considered, BRRHS-CG and ECG behave very similarly: the number of iterations is almost the same for both methods. When the block size increases, and even if the preconditioner is less effective, the number of iterations is significantly reduced with ECG and BRRHS-CG (up to a factor 5.5 for SKY2D), except for ANI3D where it remains almost constant, even slightly increasing. Furthermore, the block methods are very effective in comparison with PCG. This is especially true when the block size is large (up to more than 10 for SKY matrices for instance). Even for small block sizes we can observe a gain with respect to PCG,

around 35% of decrease for SKY3D for example.

The three rightmost columns in Table 2.4 present results obtained when the block size is increased while the number of blocks nj, of block Jacobi preconditioner, is kept constant. In this case we expect that the number of iterations will decrease because the preconditioner is the same but the block size increases, so the search space grows faster with larger block size. And this is what we obtain. For almost all matrices (except ANI3D) we observe a gain of a factor between 3 and 10 in terms of iteration count. However, for several matrices up to a certain block size the gain is not significant anymore (NH2D and SKY2D for example). As in the previous experiment, BRRHS-CG and ECG behave similarly in terms of iteration count.

2.6.3 Dynamic reduction of the search directions

First we study Orthodir variant, the results in Table 2.5 show a comparison between BRRHS-CG, Coop-CG and ECG with Algorithm 11 and without reduction of the block size (Algorithm 4). In these experiments, we set the number of blocks in the block Jacobi to 1024, and the block size to 32. Thus the efficiency of the preconditioner is decreased but it is highly parallel, typically each block would be associated with one processor, and the block size is chosen relatively high in order to compensate this. We observe that our method to reduce the block size is stable in practice: the number of iterations with or without the block size reduction is of the same order, and still far lower than with the usual PCG method. More precisely the gain is between around 20% less iterations for ANI3D test case, and up to a factor of 17 for the quasi-incompressible elasticity test case. For most of the test cases, the block methods perform 5 to 15 less iterations than the usual PCG even when the block size is reduced. Reducing the block size does not delay significantly the convergence. In most of the cases there is only 1 or 2 more iterations when reducing the block size. The dimension of the final search space (denoted $\mathcal{K}_k^{\Delta}$) is more or less decreased depending on the method and the matrix. For instance, the matrix Ela100 allows to reduce significantly the search space: 15% for BRRHS-CG, 30% for Coop-CG, and 25% for ECG. We observe that Coop-CG and ECG allows better reduction than BRRHS-CG. In particular, ANI3D is a good illustration: there is nearly no reduction of the final search space with BRRHS-CG whereas ECG allows a 10% decrease, and Coop-CG a 5% decrease. On the other hand, the dimension of the final search space of Coop-CG is larger than that of ECG (for ANI3D it is around 10% larger), and Coop-CG requires more iterations to converge (around 20% more for ANI3D). We conclude that for these test cases, ECG is the best method, both in terms of iteration count, and decrease of the final search space.

In Fig. 2.2, we plot the error (left), as well as the size of the block (right), as a function of the number of iterations for SKY2D and Ela100. We consider that the initial block size is 32 and the number of blocks nj in the block Jacobi preconditioner is 1024. For both matrices, we observe that the convergence of the error is far better for the block methods compared to PCG even with the block size reduction (a factor of 7 for SKY2D and 10 for Ela100). Still there is a small plateau before convergence for all the methods which is a well-known phenomenon with Krylov methods [65, 108]. The

Table 2.4 – Number of iterations to get the solution ($\varepsilon_{\text{solver}} = 10^{-8}$) with Orthodir (Algorithm 4), and PCG which is the usual Preconditioned Conjugate Gradient, the stopping criterion is the error, i.e., er = $\frac{\|x^*-x_k\|_2}{\|x^*\|_2}$ where x^* is the exact solution computed with LU.

	t	nj	PCG	BRRHS-CG	ECG	nj	PCG	BRRHS-CG	ECG
NH2D	2	8	66	51	51	256	127	102	103
	4	16	76	47	48	256	127	78	78
	8	32	85	43	43	256	127	63	59
	16	64	97	38	34	256	127	48	45
	32	128	112	33	30	256	127	38	34
	64	256	127	28	26	256	127	28	26
SKY2D	2	8	163	153	104	256	487	259	260
	4	16	205	109	70	256	487	132	129
	8	32	326	79	49	256	487	98	74
	16	64	385	49	40	256	487	48	46
	32	128	423	32	31	256	487	37	35
	64	256	487	28	27	256	487	28	27
SKY3D	2	8	166	106	102	256	416	352	338
	4	16	218	80	81	256	416	275	251
	8	32	321	91	90	256	416	157	156
	16	64	313	51	50	256	416	85	83
	32	128	385	40	41	256	416	48	48
	64	256	416	30	31	256	416	30	31
ANI3D	2	8	54	53	52	256	99	93	94
	4	16	65	59	58	256	99	92	88
	8	32	72	62	60	256	99	88	85
	16	64	77	64	61	256	99	83	78
	32	128	89	63	62	256	99	72	69
	64	256	99	58	56	256	99	58	56
Ela25	2	8	190	122	118	256	445	263	295
	4	16	220	96	93	256	445	192	185
	8	32	263	86	85	256	445	139	140
	16	64	313	75	77	256	445	101	104
	32	128	360	63	65	256	445	77	78
	64	256	445	58	59	256	445	58	59
Ela50	2	8	296	169	180	256	647	369	392
	4	16	364	136	131	256	647	235	251
	8	32	411	103	102	256	647	158	159
	16	64	485	87	90	256	647	116	115
	32	128	573	75	77	256	647	83	87
	64	256	647	64	66	256	647	64	66

Table 2.5 – Number of iterations to get the solution with Orthodir with dynamic reduction of the search directions (Algorithm 11, $\varepsilon_{\text{solver}} = 10^{-6}$ and $\varepsilon_{\text{def}} = \frac{\varepsilon_{\text{solver}}\|r_0\|_2}{\sqrt{t}}$), the stopping criterion is the normalized residual. The initial the block size is 32 and the number of block Jacobi is 1024. The symbol ✓ means that we reduced the size of the block and × means that we used the usual algorithm, and er = $\frac{\|x^*-x_k\|_2}{\|x^*\|_2}$ where x^* is the exact solution computed with LU.

		PCG		BRRHS-CG			Coop-CG			ECG		
	red. size	iter	er	iter	er	$\dim(\mathcal{K}_k^{\triangle})$	iter	er	$\dim(\mathcal{K}_k^{\triangle})$	iter	er	$\dim(\mathcal{K}_k^{\triangle})$
NH2D	×	175	7.2e-08	51	3.2e-09	1632	57	2.4e-09	1824	46	4.0e-09	1472
	✓	175	7.2e-08	51	3.4e-09	1571	57	3.5e-09	1599	47	3.4e-09	1346
SKY2D	×	653	9.0e-08	60	7.6e-12	1920	74	3.2e-10	2368	56	7.3e-12	1792
	✓	653	9.0e-08	60	8.6e-12	1719	76	8.5e-10	1706	57	9.5e-12	1536
SKY3D	×	393	1.2e-06	70	6.5e-09	2240	75	3.0e-09	2400	68	3.5e-09	2176
	✓	393	1.2e-06	71	5.3e-09	2187	78	1.0e-08	2159	69	5.4e-09	2054
ANI3D	×	108	2.4e-07	84	1.3e-07	2688	85	1.7e-07	2720	83	1.4e-07	2656
	✓	108	2.4e-07	84	1.3e-07	2680	85	1.8e-07	2607	84	1.6e-07	2327
Ela100	×	954	2.1e-10	102	8.1e-12	3264	120	2.8e-11	3840	101	8.0e-12	3232
	✓	954	2.1e-10	103	8.7e-12	2772	128	2.1e-10	2644	105	8.9e-12	2393
Ela2D200	×	4526	2.4e-09	254	2.2e-09	8128	294	6.6e-10	9408	252	1.8e-09	8064
	✓	4526	2.4e-09	257	2.9e-09	7466	320	2.3e-09	6949	262	3.0e-09	6730

block size reduction and the error behave similarly for both matrices, the block size is reduced when the system is starting to converge (around iteration 40 for SKY2D and iteration 60 for Ela100). For the two test cases, ECG performs better than Coop-CG and BRRHS-CG, it reduces its search directions faster than Coop-CG and better than BRRHS-CG. For these tests, BRRHS-CG has always the biggest block size during the iterations, and Coop-CG has the slowest convergence. On the other hand, ECG keeps a convergence speed similar as that of BRRHS-CG while having the lowest block size.

In short, the results in Table 2.5 and Fig. 2.2 show that ECG finds the solution of the linear system in a smaller subspace than the other block methods considered (BRRHS-CG and Coop-CG) when reducing the search directions during Orthodir iterations (Algorithm 11).

Next we study Orthomin variant, the results in Table 2.6 show a comparison between BRRHS-CG, Coop-CG and ECG with (Algorithm 12) and without reduction of the block size (Algorithm 3). We use the same preconditioner and initial block as the experiments with Orthodir. In that case we observe that our method to reduce the block size is not always stable in practice. Although the number of iterations for NH2D, SKY3D, and ANI3D is almost the same when reducing or not the number of search directions, it is not the case for SKY2D and elasticity matrices where the reduction can lead to very slow convergence. For the matrices NH2D, SKY3D and ANI3D we observe the similar behavior as with Algorithm 11. As when comparing the A-orthonormalization method, it is exactly the same results for NH2D and ANI3D, but they are slightly different for SKY3D. Thus for all these matrices (NH2D, SKY2D, SKY3D and ANI3D) ECG achieves both a fast convergence in terms of iterations and

Figure 2.2 – Block size reduction and error decreasing as a function of the number of iterations when using Orthodir with dynamic reduction of the search directions (Algorithm 11). The left figures represent the plot of the error as a function of the number of iterations. We use a log_{10} scale for the error. The right figures represent the plot of the block size as a function of the number of iterations. We only plot the block size for the block methods and not for PCG.

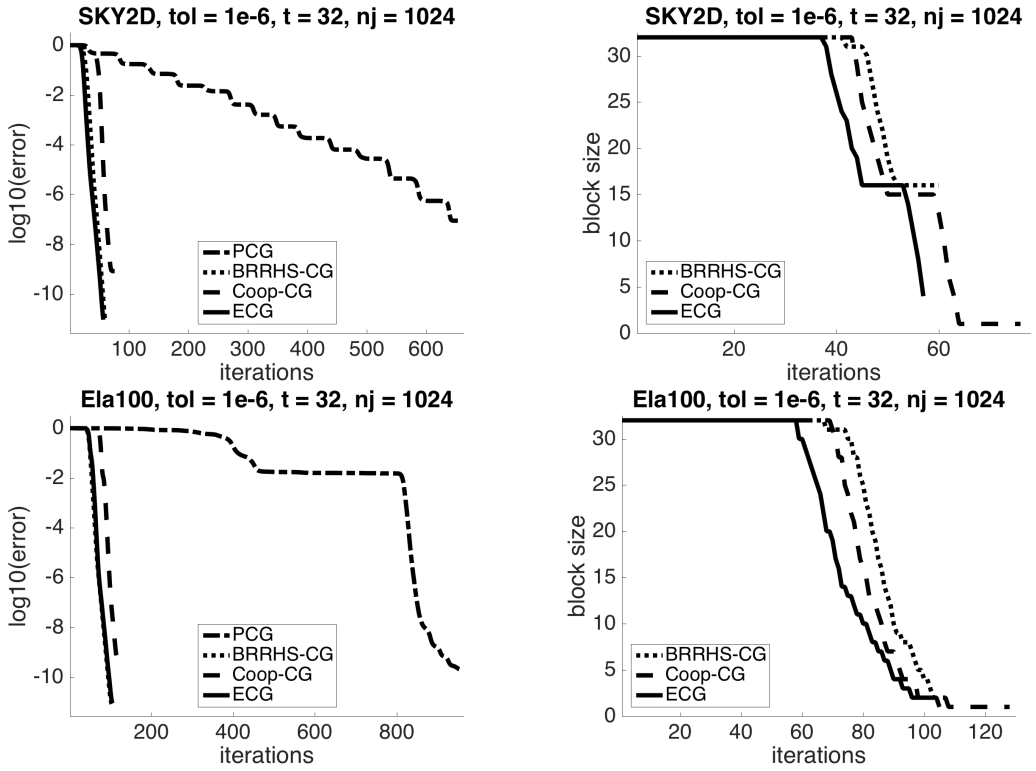


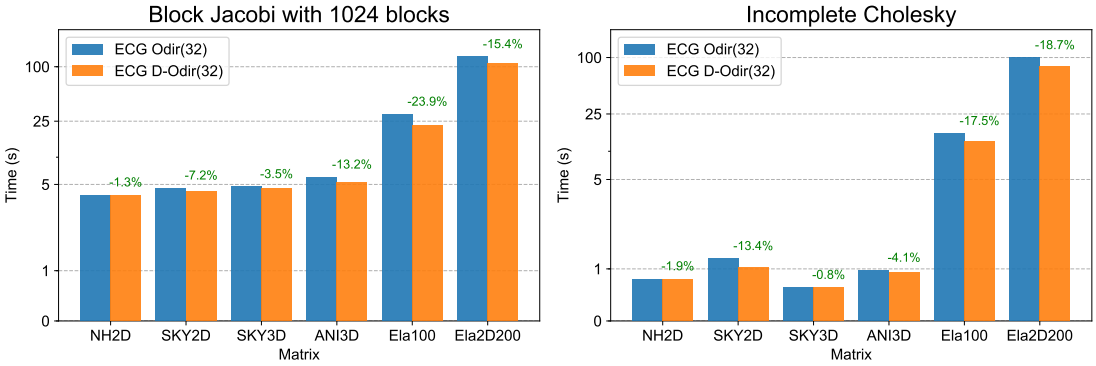
Table 2.6 – Number of iterations to get the solution with Orthomin with dynamic reduction of the search directions (Algorithm 12, $\varepsilon_{\text{solver}} = 10^{-6}$ and $\varepsilon_{\text{def}} = \frac{\varepsilon_{\text{solver}}\|r_0\|_2}{\sqrt{t}}$), the stopping criterion is the nomalized residual. The initial the block size is 32 and the number of block Jacobi is 1024. The symbol ✓ means that we reduced the size of the block and × means that we used the usual algorithm, and er = $\frac{\|x^*-x_k\|_2}{\|x^*\|_2}$ where x^* is the exact solution computed with LU.

		PCG		BRRHS-CG			Coop-CG			ECG		
	red. size	iter	er	iter	er	dim($\mathcal{K}_k^{\triangle}$)	iter	er	dim($\mathcal{K}_k^{\triangle}$)	iter	er	dim($\mathcal{K}_k^{\triangle}$)
NH2D	×	175	7.2e-08	51	3.2e-09	1632	57	2.4e-09	1824	46	4.0e-09	1472
	✓	175	7.2e-08	51	3.5e-09	1571	57	4.8e-09	1599	47	3.5e-09	1346
SKY2D	×	653	9.0e-08	60	2.9e-10	1920	105	3.3e-09	3360	56	7.8e-12	1792
	✓	653	9.0e-08	63	5.6e-10	1760	+1000	-	-	62	1.0e-10	1583
SKY3D	×	393	1.2e-06	70	2.1e-09	2240	77	2.3e-09	2464	68	1.9e-09	2176
	✓	393	1.2e-06	70	3.8e-09	2164	81	1.8e-06	2195	69	4.0e-08	2051
ANI3D	×	108	2.4e-07	84	1.3e-07	2688	85	1.7e-07	2720	83	1.4e-07	2656
	✓	108	2.4e-07	84	1.3e-07	2680	85	1.9e-07	2607	84	1.7e-07	2327
Ela100	×	954	2.1e-10	102	1.6e-09	3264	183	1.2e-09	5856	101	8.5e-10	3232
	✓	954	2.1e-10	108	1.4e-09	2796	+1000	-	-	773	1.4e-07	3571
Ela2D200	×	4526	2.4e-09	254	2.9e-09	8128	+1000	-	-	252	2.5e-10	8064
	✓	4526	2.4e-09	263	2.9e-09	7539	+1000	-	-	+1000	-	-

a good reduction of the final search space. For SKY2D, Ela100, and Ela2D200, when reducing dynamically the block size the convergence can be very delayed. In particular, Coop-CG and ECG (for Ela100 and Ela2D200 only) have a huge convergence delay. On the contrary, BRRHS-CG's convergence is not significantly impacted by the reduction of the block size. However, for the elasticity matrices we notice that the number of iterations of BRRHS-CG with Orthomin and dynamic reduction of the search directions is larger than the number of iterations of BRRHS-CG with Orthodir and dynamic reduction of the search directions (263 versus 257 for Ela2D200). Moreover, it is also larger the number of iterations of ECG with Orthodir (262 for Ela2D200) and dynamic reduction of the search directions which has a smaller final search space than BRRHS-CG. Extra numerical experiments suggest that this behavior is due to the loss of orthogonality between the search directions. Indeed, we tried to A-orthonormalize $Z_{k+1,1}$ against H and this decreases the number of iterations. However, it is not possible to do it in practice because the additional costs in terms of flops and communications would prevent any improvement in terms of performance compared to Orthomin without dynamic reduction of the search directions. Orthomin with the dynamic reduction of the search directions can suffer from a huge convergence delay. Even with BRRHS-CG, which does not have significant convergence delay, it appears that Orthodir is more effective in reducing the search space — especially ECG with Orthodir. Hence we conclude that it is preferable to use Orthodir with dynamic reduction of the search directions.

Finally, we report the runtimes obtained with the approach that gives the best experimental results, ECG with Orthodir, in Fig. 2.3. More precisely, we compare the runtimes when the block size is reduced and when it is constant. We want to point out

Figure 2.3 – Runtimes for ECG with Orthodir with a constant block size (ECG Odir), and when the block size is dynamically reduced (ECG D-Odir). The initial block size is set to 32. The preconditioner is either block Jacobi with 1024 blocks (left), or an Incomplete Cholesky decomposition with no fill-in (right).



that our implementation is done is Matlab, so it is not as optimal in terms of runtime as a compiled code written in C or C++, and it is sequential. We also do not compare with the built-in PCG method because it is very likely that it is more optimized than our implementation. First, we use a block Jacobi preconditioner with 1024 blocks, and we set the initial block size to 32. We observe that the dynamic variant is always faster. Although the gain is small for NH2D and SKY3D (less than 5%), it is quite significant with the elasticity matrices, and ANI3D (around 15% and up to more than 20% for Ela_100). Then, we assess the impact of the preconditioner on the method by using an Incomplete Cholesky decomposition with no fill-in instead of the block Jacobi preconditioner. As in the previous case, we observe that the dynamic variant is always faster. The change in the preconditioner does not have a strong impact on the runtime gain when reducing the block size. In particular, the gain remains less than 5% for NH2D and SKY3D, and up to nearly 20% for the elasticity matrices. We note that the runtime improvement is strongly related to the reduction of the final search space that we previously observed.

2.6.4 Numerical comparison with a two–level preconditioner

We have shown that ECG can converge significantly faster than the usual CG method. More precisely, we have proven Theorem 2.4.1. It can be a huge improvement over the usual CG method because it does not involve the condition number of A, but the *deflated* one $\frac{\lambda_{\max}}{\lambda_t}$ where t is the enlarging factor. However, the constant in front of the right-hand side in (2.57) is also a bit more complicated, difficult to estimate *a priori*, and possibly depending on the value of t. This is why we perform a numerical comparison between ECG and the deflated Conjugate Gradient [45, 48, 85, 115] where the effect of the smallest eigenvalues on the convergence is annihilated.

As explained in the review paper [115], there exists a lot of algorithmic variants

in order to perform the deflation of the smallest eigenvalues. In what follows, we use the so–called A-DEF1 variant. More precisely, given a preconditioner M (a block Jacobi preconditioner for instance), and Z a matrix representing a space to deflate (the eigenvectors associated to the smallest eigenvalues for instance), we use the following preconditioner,

$$\mathcal{P}_{\text{A-DEF1}} = M^{-1}P + Q, \tag{2.154}$$

where $Q = Z(Z^\top AZ)^{-1}Z^\top$ and $P = I - AQ$ [115]. In the following experiments, M is a block Jacobi preconditioner with 128 blocks and Z contains the eigenvectors associated to the smallest eigenvalues of A computed using the built–in function of Matlab. We use Orthodir without reduction of the search directions in order to simplify the comparison.

First, we compare ECG and CG using $\mathcal{P}_{\text{A-DEF1}}$ preconditioner (Deflated CG). We set the number of deflated vectors in Z equals to t the enlarging factor in ECG, and we vary this parameter. The results are summarized in Figures 2.4, 2.6, 2.5, and 2.7.

For NH2D and ANI3D (Figures 2.4 and 2.5), we observe that both methods behave similarly. The convergence speed of the deflated CG method is almost constant for these matrices, whereas it is slightly super linear with ECG with a small plateau at the beginning. Thus the deflated CG method has a sharper convergence during the first iterations, but ECG reaches first a higher accuracy. The influence of the parameter t is not significant in the comparison of the two methods for NH2D: even if the number of iterations is decreased the behavior of the methods is not really affected by the value of t. However, for ANI3D we observe that for small values of t (4 and 8) the deflated CG performs slightly better than ECG: the plateau is not compensated by a higher asymptotic speed of convergence; whereas increasing t (to 16 and 32 for instance) allows ECG to converge a bit faster than the deflated CG method.

For SKY2D and Ela50 (Figures 2.6 and 2.7), we observe that the ECG method can perform significantly better than the deflated CG method. In particular, when t is relatively small (4 or 8) ECG performs around two times less iterations than the deflated CG method. Indeed, the convergence of the deflated CG method is erratic when the number of deflated vectors is small (see Figure 2.6). When t increases, the convergence of the deflated CG method becomes smoother: the oscillations of the residual norm during the iterations tends to disappear. On the other hand, ECG still has a plateau at the beginning of the iterations, thus the deflated CG method becomes more and more competitive as t increases (see Figure 2.6). As for the previous test cases, we observe that the asymptotic convergence speed of ECG is higher than that of the deflated CG method.

As pointed out in [115], several approaches exist for performing deflation. In particular the authors have shown that $\mathcal{P}_{\text{A-DEF1}}$ is not always the most robust approach and they advice for the so–called A-DEF2 approach,

$$\mathcal{P}_{\text{A-DEF2}} = P^\top M^{-1} + Q. \tag{2.155}$$

Although $\mathcal{P}_{\text{A-DEF2}}$ gives better results from a numerical point of view, it is also more

Figure 2.4 – Comparison of Orthodir ECG with a 2–level preconditioner on NH2D. The tolerance for the computation of the eigenvectors is set to 10^{-9}.

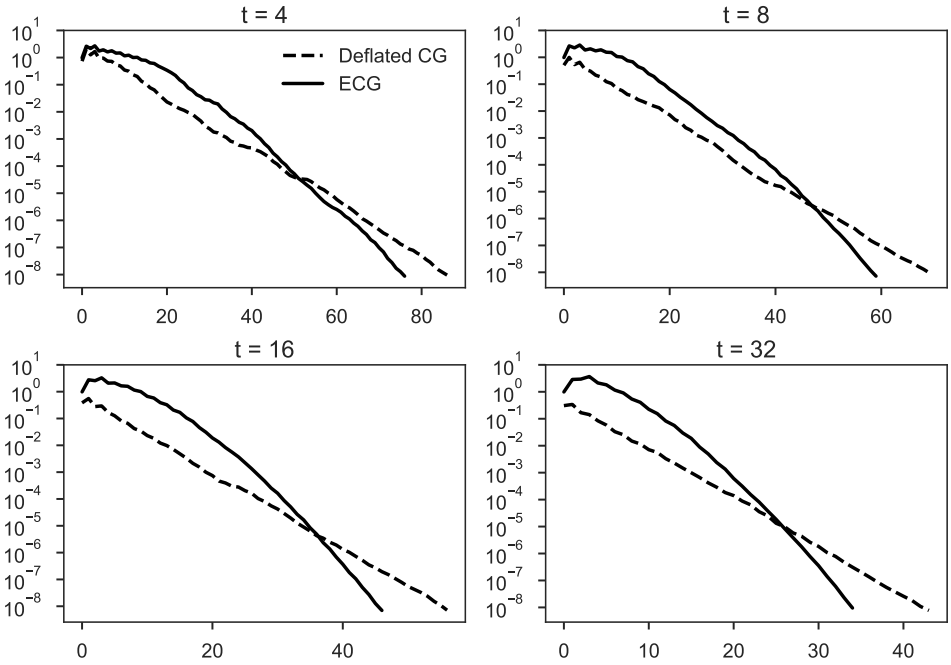


Figure 2.5 – Comparison of Orthodir ECG with a 2–level preconditioner on ANI3D. The tolerance for the computation of the eigenvectors is set to 10^{-9}.

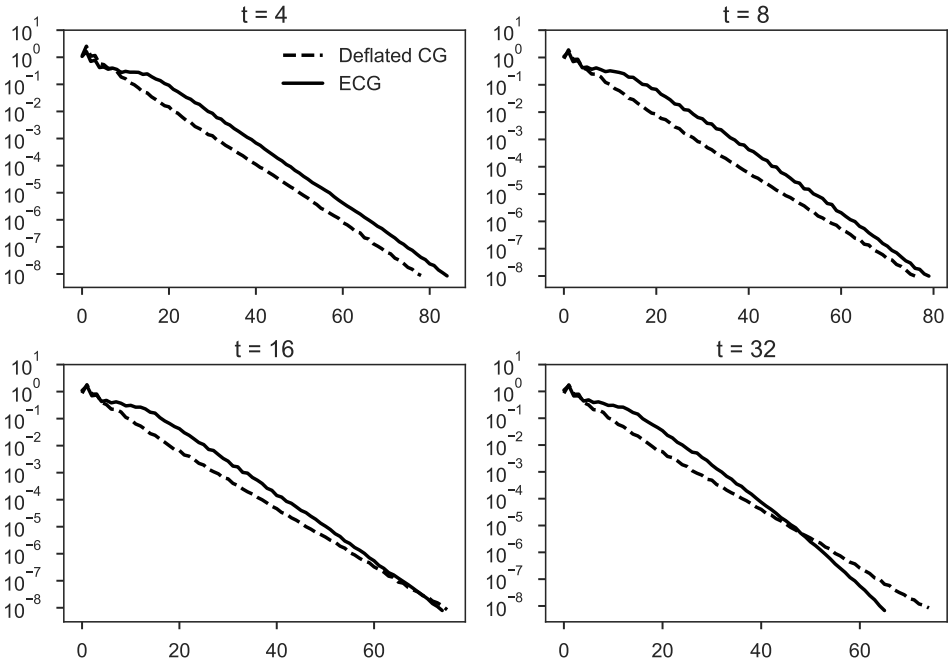


Figure 2.6 – Comparison of Orthodir ECG with a 2–level preconditioner on SKY2D. The tolerance for the computation of the eigenvectors is set to 10^{-9}.

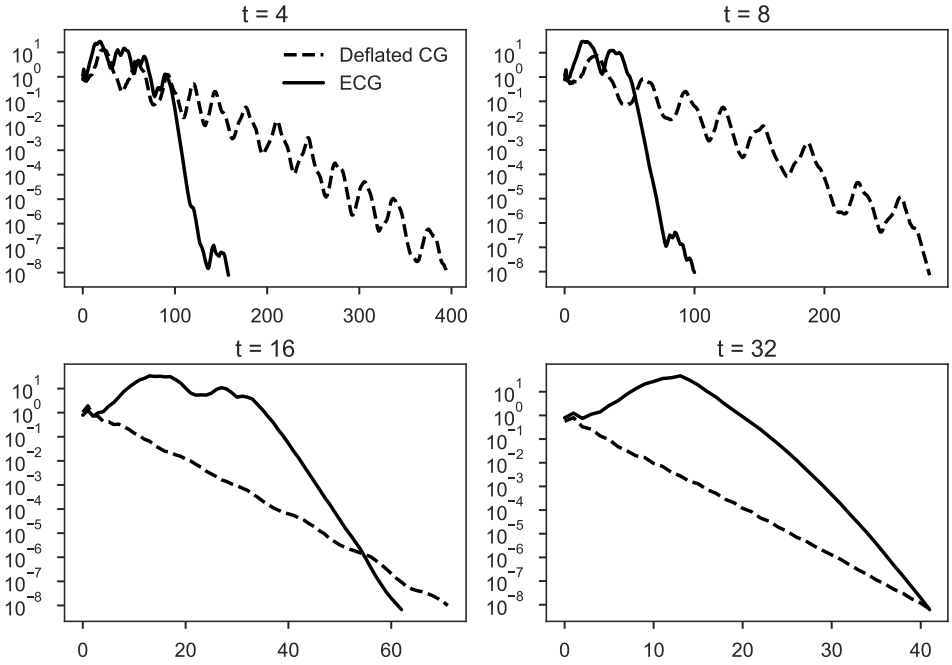


Figure 2.7 – Comparison of Orthodir ECG with a 2–level preconditioner on Ela50. The tolerance for the computation of the eigenvectors is set to 10^{-9}.

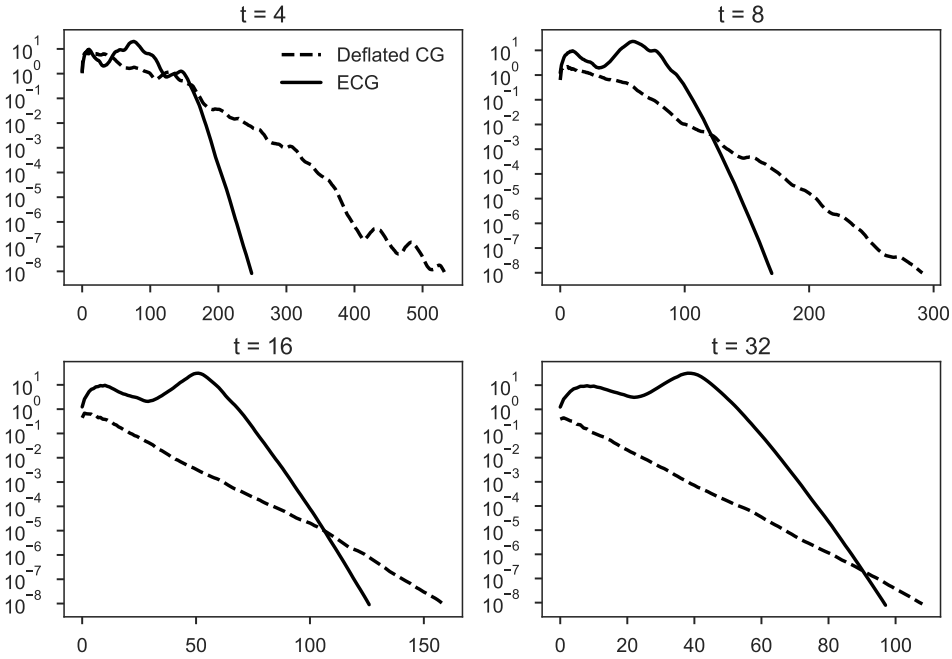
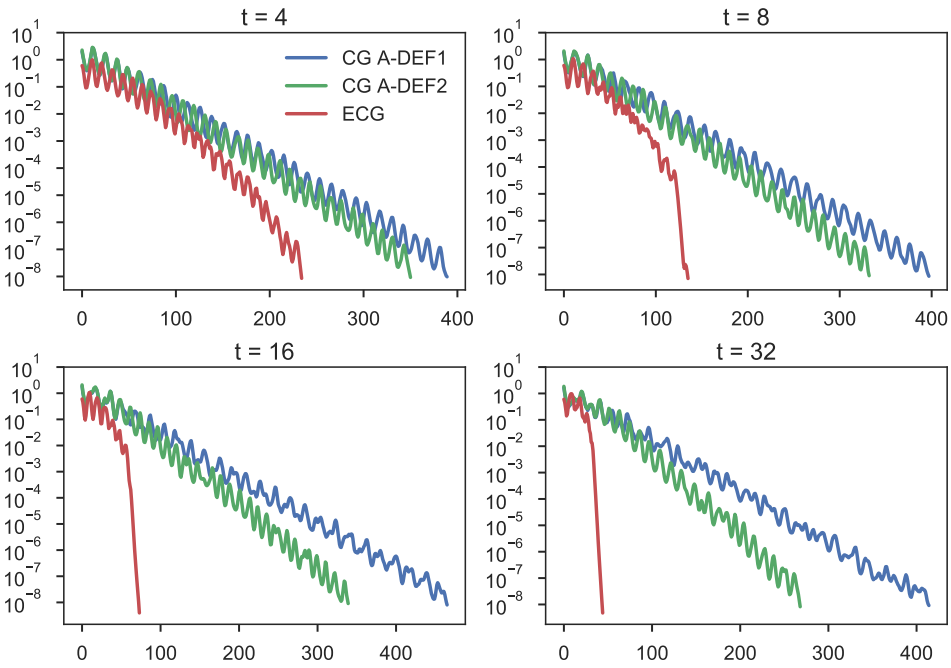


Figure 2.8 – Comparison of Orthodir ECG with several 2–level preconditioners on SKY3D. The tolerance for the computation of the eigenvectors is set to 10^{-9}.



costly to apply because it requires two coarse problem solves $Z(Z^\top AZ)^{-1}Z^\top u$ which is a communication intensive operation [74]. We compare $\mathcal{P}_{\text{A-DEF1}}$, $\mathcal{P}_{\text{A-DEF2}}$, and Orthodir ECG on the SKY3D test case. We summarize the results we obtained in Figure 2.8. We observe that for this problem ECG always performs much better than the two–level approaches. In particular, $\mathcal{P}_{\text{A-DEF1}}$ is not suitable because the number of iterations is not decreased when more vectors are deflated (t increases): the number of iterations is almost the same when $t = 8$ and $t = 32$. On the other hand, $\mathcal{P}_{\text{A-DEF2}}$ improves the convergence of the deflated method, especially when t is large. However, in all cases ECG has the best convergence behavior as it performs much less iterations than the two–level approaches: for instance around 4 times less iterations than $\mathcal{P}_{\text{A-DEF2}}$ for $t = 16$.

3

Parallel Design

Outline of the current chapter

Abstract

In this chapter, we present the parallel design of the Enlarged Conjugate Gradient method introduced in the previous chapter. It includes both Orthomin and Orthodir variant, as well as the dynamic reduction of the search directions with Orthodir, and the prevention of breakdowns with Orthomin using Breakdown-free scheme. The performance of this design is then assessed up to several thousands of processors.

Résumé

Dans ce chapitre, nous présentons le design parallèle de la méthode du Gradient Conjugué Élargi introduite au chapitre précédent. Cela inclut à la fois Orthomin et Orthodir, ainsi que la

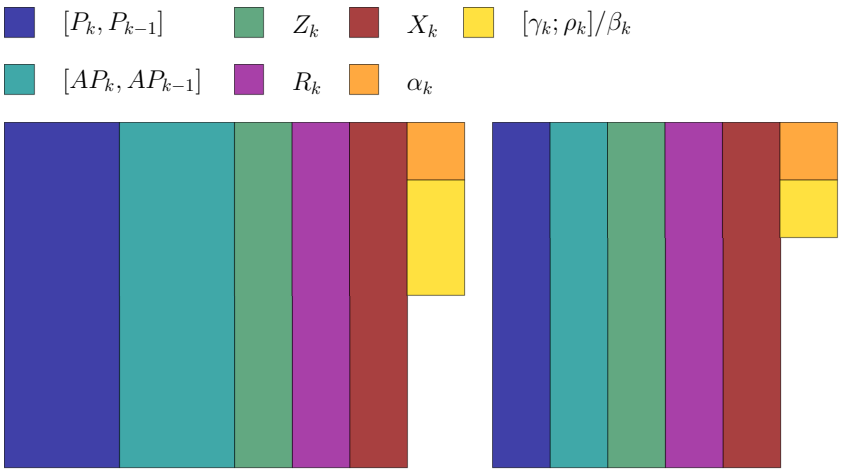


Figure 3.1 – Local distribution of the data: Orthodir on the left and Orthomin on the right.

réduction dynamique des directions de recherche, et la gestion des breakdowns éventuels avec Orthomin en utilisant la variante Breakdown-free. Les performances de cette implémentation sont ensuite évaluées jusqu'à plusieurs milliers de processeurs.

3.1 Data distribution

As it is usually the case in parallel implementations of Krylov methods, we assume that the unknowns are distributed among the processors. We also assume that each processor owns different unknowns. Thus all the variables whose size scales as the size of the linear system (X_k, R_k, P_k, AP_k, Z_k) are distributed row-wise among the processors according to the distribution of the unknowns. All variables whose size scales as the enlarging factor t (α_k, β_k, γ_k, ρ_k) are replicated on all the processors. Locally they are stored contiguously and column by column. There is no allocation or deallocation of memory during the iterations. In particular, when using Dynamic Orthodir or Breakdown-Free Orthomin the memory is not freed when the block size is reduced. The local memory consumption of preconditioned Orthodir and Orthomin on N_{proc} processors is summarized in Table 3.2. For completeness, we also add the local memory consumption of the classical CG algorithm, described in [101] for instance, where only 5 vectors and 2 scalars are needed.

3.2 Kernel operations

For one iteration of ECG, our implementation requires external routines to apply the sparse matrix product and the preconditioner to a set of vectors. Indeed the implementation of these routines highly depends on the linear system to be solved. This is why we do not take into account these operations in our cost analysis.

Given n, t such that $t \ll n$, we denote V, W tall and skinny matrices of size $n \times t$ whose rows are distributed among the processors, and α is a matrix of size $t \times t$ replicated on the N_{proc} processors. Following [79], it is possible to decompose the iterations of ECG (and more generally block CG) into the following kernels:

- $V \leftarrow V + W\alpha$ (`tsmm` in [79]),
- $\alpha \leftarrow V^\top W$ (`tsmtsm` in [79]),
- Cholesky factorization of α (`potrf`),
- triangular solve of α with several right-hand sides (`trsm`).

Indeed it is possible to rewrite the iterations of Orthodir and Orthomin with explicit calls to these kernels (Algorithms 13 and 14). In particular, the line 5 of the algorithm (Algorithm 8) can be decomposed as,

$AZ_k \leftarrow A * Z_k$	*sparse matrix set of vectors*
$C \leftarrow$ `tsmtsm`(Z_k, AZ_k)	*form* $Z_k^\top AZ_k$
$C \leftarrow$ `potrf`(C)	*Cholesky factorization*
$P_k \leftarrow$ `trsm`(Z_k, C)	
$AP_k \leftarrow$ `trsm`(AZ_k, C)	*update* P_k *and* AP_k

Doing so allows us to avoid calling the sparse matrix set of vectors product for computing AP_k at the price of an extra `trsm`.

Algorithm 13 ECG-Omin iteration

1: $Q \leftarrow AZ$
2: $\mu \leftarrow$ `tsmtsm`(Z, Q)
3: $\mu \leftarrow$ `potrf`(μ)
4: $P \leftarrow$ `trsm`(μ, P)
5: $Q \leftarrow$ `trsm`(μ, Q)
6: $\alpha \leftarrow$ `tsmtsm`(P, R)
7: $X \leftarrow$ `tsmm`(X, P, α)
8: $R \leftarrow$ `tsmm`(R, Q, α)
9: $\mu \leftarrow$ `tsmtsm`(R, R)
10: **if** $\sum_{i=1}^{t} \mu(i,i) < \varepsilon$ **then**
11: stop
12: **end if**
13: $Z \leftarrow M^{-1}R$
14: $\beta \leftarrow$ `tsmtsm`(Q, Z)
15: $Z \leftarrow$ `tsmm`(Z, P, β)
16: $P \leftarrow Z$

Kernel	# flops	# messages	# words
`tsmm`	$2\frac{nt^2}{P}$	0	0
`tsmtsm`	$\frac{nt(2t+1)}{P}$	$\ln(P)$	t^2
`potrf`	$\frac{1}{3}t^3$	0	0
`trsm`	$\frac{nt^2}{P}$	0	0

Table 3.1 – Complexity of the kernels where t is the enlarging factor.

Algorithm 14 ECG-Odir iteration

1: $Q(:, 1:t) \leftarrow AZ$
2: $\mu \leftarrow \texttt{tsmtsm}(Z, Q(:, 1:t))$
3: $\mu \leftarrow \texttt{potrf}(\mu)$
4: $P(:, 1:t) \leftarrow \texttt{trsm}(\mu, P(:, 1:t))$
5: $Q(:, 1:t) \leftarrow \texttt{trsm}(\mu, Q(:, 1:t))$
6: $\alpha \leftarrow \texttt{tsmtsm}(P(:, 1:t), R)$
7: $X \leftarrow \texttt{tsmm}(X, P(:, 1:t), \alpha)$
8: $R \leftarrow \texttt{tsmm}(R, Q(:, 1:t), \alpha)$
9: $\mu \leftarrow \texttt{tsmtsm}(R, R)$
10: **if** $\sum_{i=1}^{t} \mu(i,i) < \varepsilon$ **then**
11: stop
12: **end if**
13: $Z \leftarrow M^{-1}Q(:, 1:t)$
14: $\beta \leftarrow \texttt{tsmtsm}(Q, Z)$
15: $Z \leftarrow \texttt{tsmm}(Z, P, \beta)$
16: $P(:, t+1:2t) \leftarrow P(:, 1:t)$
17: $P(:, 1:t) \leftarrow Z$
18: $Q(:, t+1:2t) \leftarrow Q(:, 1:t)$

3.3 Cost analysis

In order to detail the complexities in terms of flops and communications of one iteration of the different variants of ECG, we start by making the analysis for the kernel operations described in the previous section. Then the overall complexities are obtained by summing the contributions of the kernels.

We focus on the dense kernels because the SpMM and the application of the preconditioner should be done by the user. Assuming that the tall and skinny matrices of size $n \times t$ are equally distributed among P processors, the complexities of the kernels in terms of FLOPs, words and messages are summarized in Table 3.1

Following ECG algorithm (Algorithm 8), each iteration of Orthodir and Orthomin consists of 3 `tsmm` (lines 7, 8, and 12), 4 `tsmtsm` (lines 5, 6, 9, and 12), 1 `potrf` (line 5)

and 2 `trsm` (line 5). Hence the difference between the two algorithms is the construction of Z_{k+1} (line 12). The `tsmtsm` and `tsmm` for constructing Z_{k+1} in Orthodir (equations (2.24)-(2.26)) are twice as costly as for Orthomin (equations (2.22)-(2.23)).

More precisely, for Orthomin we have,

$$\#flops(\text{Omin}) = 3 \times 2\frac{nt^2}{P} + 4 \times \frac{nt(2t+1)}{P} + \frac{1}{3}t^3 + 2 \times \frac{nt^2}{P}, \tag{3.1}$$

$$= 16\frac{nt^2}{P} + 4\frac{nt}{P} + \frac{1}{3}t^3. \tag{3.2}$$

And for Orthodir,

$$\#flops(\text{Odir}) = 4 \times 2\frac{nt^2}{P} + 5 \times \frac{nt(2t+1)}{P} + \frac{1}{3}t^3 + 2 \times \frac{nt^2}{P}, \tag{3.3}$$

$$= 20\frac{nt^2}{P} + 5\frac{nt}{P} + \frac{1}{3}t^3. \tag{3.4}$$

As matrices of size $t \times t$ are replicated among the processors, we notice that `tsmm`, Cholesky factorization of α and triangular solve of α are local operations without any communication. Hence we use the corresponding LAPACK routines: `gemm`, `potrf` (dense Cholesky factorization) and `trsm` (dense triangular solve with several right-hand sides). However V and W are distributed and `tsmtsm` is not a local operation. The LAPACK routine `gemm` is called to compute the local product $V_i^\top W_i$ followed by a call to `MPI_Allreduce`.

Thus, the only kernel operation that requires a communication is `tsmtsm` and 4 calls to `MPI_Allreduce` are done per iteration. It is usually assumed that during a call to `MPI_Allreduce` the number of messages sent and received on the network is equal to $\log_2(N_{\text{proc}})$ – although the exact number depends on the MPI implementation [116]. Moreover it is a blocking operation: when completed all the processors are synchronized. This is why in practice, as in the classical CG, the communication cost is dominated by 2 calls to `MPI_Allreduce`: the one after the sparse matrix set of vectors product (line 5) and the one after the preconditioner (line 12), because they occur after operations with a potential load imbalance between processors.

In summary, the detailed costs of one iteration of Orthodir and Orthomin in terms of flops, words, and messages are indicated in Table 3.2. For the sake of comparison, we recall the complexity of the CG algorithm described in [101]. We also report the number of `MPI_Allreduce` in parenthesis, in addition to the order of magnitude of the number of messages. In summary, one iteration of ECG is approximately t^2 times more costly in terms of flops than one iteration of CG. While the number of messages is of the same order, the number of words is also t^2 times larger. Indeed there is a trade-off between these extra costs and the reduction of the number of iterations due to the enlargement of the search spaces.

The implementation of the dynamic reduction of the search directions within Orthodir follows Algorithm 15 — it is nothing but a slight rewriting of Algorithm 11.

	# flops	# messages	# words	memory
Omin	$16\frac{nt^2}{N_{\text{proc}}} + 4\frac{nt}{N_{\text{proc}}} + \frac{1}{3}t^3$	$4\log_2(N_{\text{proc}})$ (4)	$4t^2$	$5\frac{nt}{N_{\text{proc}}} + 2t^2$
Odir	$20\frac{nt^2}{N_{\text{proc}}} + 5\frac{nt}{N_{\text{proc}}} + \frac{1}{3}t^3$	$4\log_2(N_{\text{proc}})$ (4)	$5t^2$	$7\frac{nt}{N_{\text{proc}}} + 3t^2$
CG	$10\frac{n}{P}$	$2\log_2(N_{\text{proc}})$ (2)	2	$5\frac{n}{N_{\text{proc}}}$

Table 3.2 – Complexity of Orthodir, Orthomin and CG where t is the enlarging factor.

Algorithm 15 ECG D-Odir algorithm.

$P_0 = 0$
$Z_1 = R_0^e$
$H = \emptyset$
$k = 1$
for $k = 1, \ldots, k_{\max}$ **do**
  $P_k = Z_k(Z_k^\top A Z_k)^{-1/2}$
  $\alpha_k = P_k^\top R_{k-1}$
  $\alpha_k = U_k \Sigma_k V_k^\top$
  let s_k be the number of singular values of α_k bigger than ε_{def}
  if $s_k < s_{k-1}$ **then**
    $\alpha_k = U_k^\top \alpha_k$
    $P_k = P_k U_k$
    $\alpha_k = \alpha_k(1 : s_k, :)$
    $H = [H, P(:, s_k : s_{k-1})]$
    $P_k = P_k(:, 1 : s_k)$
  end if
  $X_k = X_{k-1} + P_k \alpha_k$
  $R_k = R_{k-1} - A P_k \alpha_k$
  if $\| \sum_{i=1}^{t} R_k^{(i)} \|_2 < \varepsilon$ **then**
    stop
  end if
  $\gamma_k = (AP_k)^\top (AP_k)$
  $\rho_k = (AP_{k-1})^\top (AP_k)$
  $\delta_k = (AH)^\top (AP_k)$
  $Z_{k+1} = AP_k - P_k \gamma_k - P_{k-1} \rho_k - H \delta_k$
  $k = k + 1$
end for
$x_k = \sum_{i=1}^{t} X_k^{(i)}$

In practice, we use LAPACK routine `gesvd` and only compute the right singular vectors of α_k denoted U_k. We check the singular values obtained. If there are some lower than $\frac{\varepsilon}{\sqrt{t}}$, which is the criterion proposed in Section 2.5.2, we call `geqrf` on U in order to perform the updates PU_k, APU_k and $U_k^\top \alpha_k$ in-place with `ormqr`. Since P_k and AP_k are stored in a column major fashion the selection of the columns is done at no cost. Similarly H is not explicitly defined. However the selection of the first rows of α_k implies an in-place memory rearrangement.

The implementation of Breakdown-Free Orthomin is similar to Orthomin except the computation of a rank-revealing QR decomposition of Z_{k+1}. As Z_{k+1} is distributed, it is not reasonable to use a LAPACK kernel to compute it. Instead we use a modification of Chol-QR algorithm [123] which is a cheaper but less stable alternative to TS-RRQR [33, 34]. Its implementation is very easy using the LAPACK routine `pstrf` (Cholesky with pivoting) for computing (R, π) at line 2. Following [71] we use the default tolerance of `pstrf` for detecting exact rank deficiency of Z_{k+1}.

Algorithm 16 Chol-RRQR

Input: P, ε

Output: Q_1 orthogonal such that

$$P\pi = \begin{pmatrix} Q_1 & Q_2 \end{pmatrix} \begin{pmatrix} R_{11} & R_{12} \\ 0 & R_{22} \end{pmatrix}$$

where π is a permutation and all the diagonal elements of R_{11} are larger than ε

1: $\mu \leftarrow P^\top P$

2: Compute (R, π) such that $\pi^\top \mu \pi = R^\top R$ with $R = \begin{pmatrix} R_{11} & R_{12} \\ 0 & R_{22} \end{pmatrix}$ and all the diagonal elements of R_{11} are larger than ε^2

3: $P_1 \leftarrow P\pi(:, 1 : size(R_{11}))$

4: $Q_1 \leftarrow P_1 R_{11}^{-1}$

3.4 Performance results

In this section, we evaluate the performance of our parallel design of ECG on a set of matrices coming from Tim Davis' collection [32], as well as discretization of the linear elasticity problem with heterogeneous coefficients, up to several thousands of cores using a block Jacobi preconditioner. We also compare ECG with PETSc's implementation of the standard CG method.

3.4.1 Description of the parallel environment

In the experiments we use a block Jacobi preconditioner, associating at each block a MPI process. Before calling ECG, each MPI process factorizes the diagonal block of A

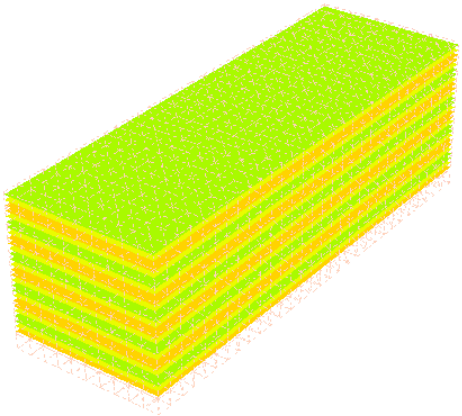


Figure 3.2 – Heterogeneity pattern of the Young's modulus and Poisson's ratio for elasticity matrices.

corresponding to the local row panel that it owns. At each iteration of ECG, each MPI process performs a backward and forward solve locally in order to apply the preconditioner. Hence the application of the preconditioner does not need any communication. It is likely that there exists better preconditioners than block Jacobi for our test cases, however we are interested in the iterative method rather than in the preconditioner. In particular, we do not want to target specific applications and aim at being as generic as possible. Although in theory it is possible to apply any preconditioner within this implementation, in practice it is essential that applying this preconditioner to several vectors at the same time is not too costly, *e.g*, a sublinear complexity with respect to the number of vectors.

The following experiments are performed on a machine located at Umeå University as part of High Performance Computing Center North (HPC2N), called Kebnekaise. It is a heterogeneous machine formed by a mix of Intel Xeon E5-2690v4 (Broadwell) with 2x14 cores (and E7-8860v4 for large memory computations), Nvidia K80 GPU and Intel Xeon Phi 7250 (Knight's Landing) with 68 cores. In our experiments, we use the so-called compute nodes, which are formed by Intel Xeon E5-2690v4 (Broadwell) with 2x14 cores. For a detailed description of the machine, we refer to the online documentation[1].

We compile the code (and its dependencies) using Intel toolchain installed on the machine: mpiicc' (based on `icc` version 18.0.1 20171018) and MKL [119] version 2018.1.163. We use PETSc [8] in order to compare ECG implementation to PETSc's CG implementation. In particular, PETSc is configured to use MKL-PARDISO as exact solver for sparse matrices in the block Jacobi preconditioner. For partitioning the matrix we are using the METIS library downloaded and installed by PETSc.

[1] https://www.hpc2n.umu.se/resources/hardware/kebnekaise

Name	Size	Nonzeros	Problem
Hook_1498	1,498,023	59,374,451	Structural problem
Flan_1565	1,564,794	117,406,044	Structural problem
Bump_2911	2,911,419	130,378,257	Reservoir simulation
Ela_20	2,118,123	74,735,397	Linear elasticity
Ela_30	4,615,683	165,388,197	Linear elasticity

Table 3.3 – Test matrices.

3.4.2 Test cases

The Ela matrices arise from the linear elasticity problem with Dirichlet and Neumann boundary conditions defined as follows

$$\operatorname{div}(\sigma(u)) + f = 0 \qquad \text{on } \Omega, \tag{3.5}$$

$$u = 0 \qquad \text{on } \partial\Omega_D, \tag{3.6}$$

$$\sigma(u) \cdot n = 0 \qquad \text{on } \partial\Omega_N, \tag{3.7}$$

where Ω is a unit cube. The matrices Ela_N correspond to this equation discretized with FreeFem++ [67] using a triangular mesh with $1600 \times N \times N$ points on the corresponding vertices and P1 finite elements scheme. $\partial\Omega_D$ is the Dirichlet boundary, $\partial\Omega_N$ is the Neumann boundary, f is some body force, u is the unknown displacement field. $\sigma(.)$ is the Cauchy stress tensor given by Hooke's law: it can be expressed in terms of Young's Modulus E and Poisson's ratio ν. For a more detailed description of the problem see [62, 74, 108]. We consider a heterogeneous beam made of several layers of a hard material $(E_1, \nu_1) = (2 \times 10^{11}, 0.25)$ and a soft material $(E_2, \nu_2) = (10^7, 0.45)$, *i.e.*, discontinuous E and ν. This test case is known to be difficult because the matrix is ill conditioned. In particular, the classical one-level preconditioners are not expected to be very effective [36].

As previously pointed out, ECG is an algebraic method that does not rely on any particular assumption on the matrix, except that it is symmetric positive definite. As an illustration, we also test the implementation on several SPD matrices coming from the Sparse Matrix Collection of *Tim Davis* [32]: Hook_1498, Flan_1564 and Bump_2911. Numerical properties of the test matrices are summarized in Table 3.3.

3.4.3 Results

In all the experiments the tolerance is set as the default tolerance of PETSc, *i.e.*, 10^{-5} and the maximum number of iterations is set to 5000. The right-hand side is chosen uniformly random and normalized and the initial guess is set to 0. We do not use any kind of threading and use 28 MPI processes per node. Unless otherwise stated, we use one OpenMP thread per MPI process — we also perform numerical experiments to

observe the effect of threading.

Impact of the enlarging factor

First we study the impact of the enlarging factor t on the methods. We fix the number of processors and we vary the value of t for the 4 methods: Orthodir (Odir), Orthodir with dynamic reduction of the search directions (D-Odir), Orthomin (Omin) and Breakdown-Free Orthomin (BF-Omin). The results obtained are summarized in Table 3.4

For Hook_1498 we set the number of MPI processes to 56. The runtime is effectively reduced when t is larger than 1 — which corresponds to the usual CG method. The minimal runtime is attained when $t = 4$, then it progressively increases especially when t becomes larger than 16. Indeed, at some point the extra arithmetic operations become prohibitive and for instance with $t = 24$ and $t = 28$, the runtimes are higher than when $t = 1$. We observe that the dynamic reduction of the search directions becomes interesting when t is relatively large (12 to 28), otherwise the gain is marginal. For this matrix, the 4 variants behave quite similarly and Omin, and D-Odir are the most efficient. There is a slight advantage in favor of Omin when t is small (4 or 8), and D-Odir when t is large (12 to 28).

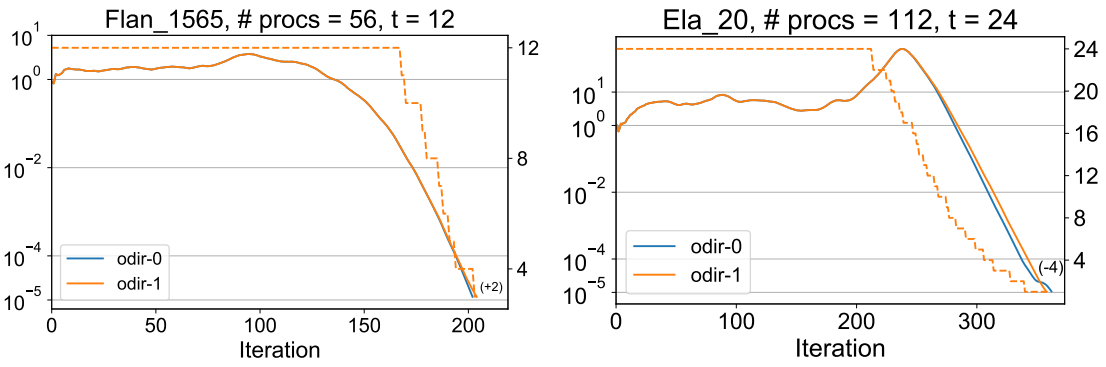
For Flan_1565 the number of MPI processes is fixed to 56. We remark that the runtime is decreasing until $t = 12$ and then it increases slightly. When t is relatively small the 4 methods are comparable. However as t increases the effect of dynamic reduction becomes more visible. With $t = 28$, D-Odir is almost 10% faster than Odir. On the other hand, as we are detecting exact rank deficiency of Z_{k+1}, BF-Omin did not reduce the size of the block and as no breakdown occurs it is slightly more costly than Omin. We also tested $\frac{\varepsilon}{\sqrt{t}}$ as the tolerance for detecting breakdowns but this does not allow the method to converge. Overall, for this matrix, the best method is D-Odir with $t = 12$.

For Bump_2911 we fix the number of MPI processes to 112. For this matrix the reduction of the number of iterations is not balancing the increase in flops. For instance, the number of iterations for D-Odir(8) is 695, and for D-Odir(12) it is 665, which represents a decrease of only 4% in the number of iterations. According to Theorem 2.4.1, it is very likely that in this case the preconditioned matrix does not have a cluster of small eigenvalues, hence the convergence of the method is not significantly improved when enlarging the Krylov subspace. However, we also notice that using the dynamic Orthodir variant (D-Odir) allows to reduce significantly the runtime when t is large: D-Odir is around 20% faster than Odir.

We also perform this study for Ela_20 with 112 MPI processes. First we remark that a breakdown occurs with Orthomin for all the values of t that we tested. This behavior of elasticity matrices had also been seen previously in Section 2.6. Using BF-Omin effectively cures the breakdowns but does not allow the method to converge within the prescribed maximum number of iterations. Similarly, D-Odir does not converge when $t = 4$, but performs very well when t is larger. On the contrary, Odir is very stable and

	t	Odir	D-Odir	Omin	BF-Omin
Hook_1498	1	20.6	20.8	20.3	20.2
	4	12.9	12.8	12.7	12.8
	8	13.8	13.1	13.2	13.4
	12	15.6	14.5	15.0	15.2
	16	18.2	16.4	17.6	17.9
	20	20.9	18.0	20.3	20.5
	24	21.5	18.8	20.7	21.2
	28	23.9	20.4	22.6	23.2
Flan_1565	1	56.9	62.8	56.7	56.7
	4	36.3	36.4	35.5	35.9
	8	30.0	29.6	29.0	29.1
	12	30.2	29.1	29.8	29.4
	16	31.3	29.3	30.2	30.7
	20	33.1	30.7	32.0	32.7
	24	37.9	33.7	36.2	36.9
	28	39.2	34.9	37.6	38.5
Bump_2911	1	54.4	53.3	53.4	53.0
	2	64.9	62.7	64.5	65.5
	4	76.9	72.4	75.4	77.0
	8	93.6	85.4	91.5	91.5
	12	123.1	104.1	122.1	122.5
	16	151.2	123.6	147.1	148.6
	20	179.7	143.3	174.0	178.4
	24	198.3	158.3	195.5	199.3
	28	223.6	171.8	219.0	223.5
Ela_20	1	++	++	++	++
	4	97.6	++	-	++
	8	72.8	55.0	-	++
	12	56.8	51.5	-	++
	16	53.6	47.5	-	++
	20	56.3	47.2	-	++
	24	57.8	46.6	-	++
	28	59.9	47.5	-	++

Table 3.4 – Runtime results (in seconds) for Hook_1498 ($N_{\text{proc}} = 56$), Flan_1565 ($N_{\text{proc}} = 56$), Ela_20 ($N_{\text{proc}} = 112$), and Bump_2911 ($N_{\text{proc}} = 112$). The ++ means that the maximum number of iterations (5000) was reached and the - means that a breakdown occurred.



(a) (b)

Figure 3.3 – Convergence of D-Odir (odir-1) compared to Odir (odir-0). The dash line represents the number of search directions for D-Odir. In parenthesis the difference of iteration count to reach convergence between D-Odir and Odir (+ means that D-Odir took more iterations to converge).

converges for all the values of t tested. As for Bump_2911 we observe that D-Odir is around 20% faster than Odir when $t = 24$. Overall, for this matrix, the best method is D-Odir with $t = 24$.

In order to better understand how the dynamic reduction of the search directions affects the convergence we plot the normalized residual and the block size (dash line) as a function of the iteration count for Flan_1565 (Fig. 3.3a) and Ela_20 (Fig. 3.3b). We notice that the convergence is not really affected by the reduction of the search directions because the number of iterations remains almost the same. However the block size is effectively reduced as soon as the method starts to converge. We note that the Ela_20 test case is very favorable: the block size is reduced even a bit before the convergence and the number of iterations is lower than when the search reductions are not reduced.

In conclusion, D-Odir is the best method over the different variants of ECG that we tested: it is a good compromise between the stability of Odir and the efficiency of the classical CG. Nevertheless, there exists matrices such as Bump_2911 for which the reduction of the number of iterations does not compensate the extra cost of ECG compared to the classical CG, even when using the dynamic reduction of the search directions. These results support the theoretical convergence study that was done in the previous section. ECG(t) is acting as if the t smallest eigenvalues of the matrix were deflated. Finally, we notice that values of t between 8 to 24 are good default parameters. Indeed, such values allow to effectively reduce the number of iterations while maintaining an affordable cost per iteration.

	D-Odir(12)		PETSc's CG		
# MPI	# iter	time (s)	# iter	time (s)	speed-up
252	332	12.0	1,709	14.8	1.2
504	405	6.1	2,430	8.4	1.4
1,008	519	4.1	3,179	4.9	1.2
2,016	637	3.6	2,687	2.6	0.7

Table 3.5 – Strong scaling study for Flan_1565. The speed-up is the ratio between PETSc runtime and ECG runtime.

	D-Odir(24)		PETSc's CG		
# MPI	# iter	time (s)	# iter	time (s)	speed-up
252	513	77.9	13,626	406.8	5.2
504	531	45.5	15,819	258.9	5.7
1,008	606	23.7	17,023	94.7	4.0
2,016	696	14.5	19,047	34.5	2.5

Table 3.6 – Strong scaling study for Ela_30. The speed-up is the ratio between PETSc runtime and ECG runtime.

Strong scaling study

Following the parameter study, we perform a strong scaling study on Flan_1565 and Ela_30. As Bump_2911 does not seem particularly well suited for the method we do not perform the strong scaling study on this matrix.

For Flan_1565 we compare PETSc's CG and D-Odir with $t = 12$, the best choice over the parameters we tested. The resulting runtimes are presented in Table 3.5. When the number of MPI processes is relatively low ECG scales as well as PETSc, *i.e.*, almost linearly. As the number of iterations is significantly reduced with D-Odir(24), there is about 20% speed-up compared to PETSc at such scales. Nevertheless, we notice that for 2,016 MPI processes PETSc is significantly faster than ECG. This is likely because the number of iterations with PETSc is reduced with respect to 1,008 MPI processes. This behavior is not expected because it is known that block Jacobi preconditioners are not scalable (see [36] for instance). Indeed, we observe that the number of iterations is effectively increasing both for ECG and CG when the number of MPI processes increases.

Then we make the same comparison on Ela_30 test case for which we use D-Odir and $t = 24$, as discussed previously. The resulting runtimes are summarized in Table 3.6. We observe that both PETSc and ECG are scaling very well. Enlarging the Krylov subspaces allows us to reduce drastically the number of iterations: D-Odir(24)

performs around 25 times fewer iterations than CG. As a consequence, D-Odir(24) is more than 5 times faster than PETSc's CG at small scale and around 2.5 times faster at large scale. We believe that this relatively poor scaling of D-Odir(24) compared to PETSc's CG at large scale is due to the implementation that is not as optimized as PETSc which has been developed over many years. For instance, the routine we use for computing the sparse matrix–set of vectors multiplication is certainly not as optimized as that of PETSc for computing the sparse matrix–vector multiplication. Also, we mentioned that we are currently performing 4 calls to `MPI_Allreduce` per iteration, but that could be reduced to 2 by fusing them. Furthermore, we could use Pipelining [54] or Communication-Avoiding based on s-step methods [22, 69] on top of ECG — that would require taking into account a possible loss of numerical stability of the method.

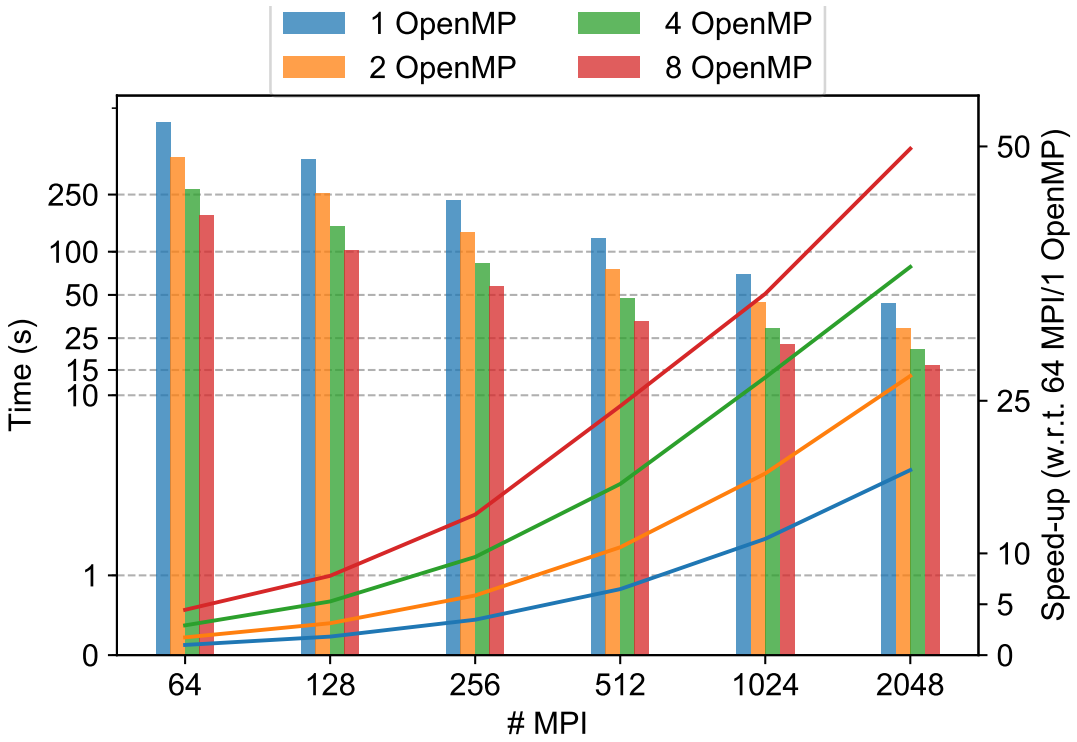
Dependence on the mesh size

Given the importance of the parameter t regarding the efficiency of the method, we perform a study of the convergence of the method with respect to the mesh size for the elasticity test case. More precisely, we consider the mesh used for generating the Ela_30 matrix, then we coarsen it by dividing the number of points in each dimension by $2^{1/3}$. Thus, we generate 3 additional elasticity matrices on which we perform a weak scaling experiment. Our major focus is not the weak scaling of ECG, but rather the comparison between PETSc's CG and D-Odir in terms of runtime.

The results are summarized in Table 3.7. First, we fix $t = 24$ and we perform a weak scaling study. We observe that D-Odir(24) is always between 2.5 to 2.9 times faster than PETSc CG, but the gap tends to slightly decrease when the number of MPI processes increases. As ECG(t) is acting as if the t smallest eigenvalues of the matrix were deflated, it seems natural to use smaller values of t for the smaller matrices. Indeed, we perform another set of experiments where we vary t as well as the size of the problem. We observe that this slightly improves the results for the smaller matrices, we obtain for example an overall speed-up of 3.1 compared to 2.9 for the smallest matrix. As expected, this does not have a significant effect on the larger matrices. Another interesting observation is that even if the number of iterations is almost constant when increasing both t and the number of MPI processes, it does not improve the scaling because the cost of one iteration also increases. Overall, D-Odir is still at least 2.5 times faster than PETSc's CG.

Impact of threads on performance

One motivation for enlarging the Krylov subspaces is to increase the arithmetic intensity of the resulting methods. This is particularly interesting to take advantage of the so-called manycore architecture as Nvidia GPUs, Intel Xeon Phi, or Sunway SW26010 used in the Sunway TaihuLight supercomputer. As the implementation relies on the MKL library which is multi-threaded [119], it is straightforward to assess its efficiency on the Xeon Phi processors.



(a) The bars represent the runtime (left) and the lines represent the corresponding speed-up with respect to 64 MPI with 1 thread each (right).

	Omin(1)		Odir(24)	
# omp	time (s)	speed-up	time (s)	speed-up
1	89	-	44	-
2	74	1.2	29	1.5
4	80	1.1	21	2.1
8	79	1.1	16	2.8

(b) Comparison between Omin(1) and Odir(24) with 2048 MPI processes. We indicate the speed-up when increasing the number of threads for each method.

Figure 3.4 – Strong scaling study for Ela_30 matrix on Cori (# omp stands for the number of threads assigned to each MPI processes).

			D-Odir		PETSc's CG		
# MPI	n	t	# iter	time (s)	# iter	time (s)	sp.-up
252	$6.15e^5$	24	323	4.5	8,842	13.2	2.9
504	$1.21e^6$	24	418	6.9	11,652	18.3	2.7
1008	$2.38e^6$	24	538	9.8	14,487	24.6	2.5
2016	$4.61e^6$	24	696	14.5	19,047	34.5	2.5
252	$6.15e^5$	12	506	4.2	8,842	13.2	3.1
504	$1.21e^6$	16	536	6.4	11,652	18.3	2.9
1008	$2.38e^6$	20	538	9.7	14,847	24.6	2.5

Table 3.7 – Weak scaling study. The dimension of the matrix is denoted n, and t denotes the enlarging factor. The speed-up (sp.-up) is the ratio between PETSc runtime and ECG runtime.

In order to do so, we perform the following experiments on NERSC's supercomputer Cori. It consists in two partitions, one with Intel Haswell processors and another one with the last generation of Intel Xeon Phi processors: Knights Landing (KNL). More precisely, the second partition consists in 9,688 single-socket Intel Xeon Phi 7250 (KNL) processors with 68 cores each. For a detailed description of the machine, we refer to the online documentation[2]. We compile the code (and its dependencies) using the default compilers and libraries installed on the machine: `icc` version 18.0.1, `cray-mpich` version 7.6.2, MKL version 2018.1.163 and METIS version 5.1.0. We consider Ela_30 test case and we study the impact of threads on the strong scaling of Odir(24). We do not use the dynamic reduction of the search directions in order to keep the cost of one iteration constant during the solve to better understand the effect of threading. We both increase the number of MPI processes from 64 to 2048 and the number of threads from 1 to 8 – this means at most $2,048 \times 8 = 16,384$ threads, each one being bound to one physical core.

The results obtained are summarized in Fig. 3.4a. First of all, we notice that there is a trade-off between using threads or MPI processes because the number of MPI processes dictates the preconditioner. Indeed, there are as many blocks in the block Jacobi preconditioner as the number of MPI processes, thus increasing the number of MPI processes deteriorates the quality of the preconditioner. For instance, using 512 MPI processes takes 123s, and using 64 MPI processes with 8 threads each takes 179s: it is an increase of 50% compared to 512 MPI processes. Nevertheless, we observe that using more than 2 threads, and up to 8, has always a significant effect on the speed-up, even when the number of MPI processes is high. For instance, as shown in Table 3.4b, increasing the number of threads from 1 to 8 with a fixed number of 2,048 MPI processes leads to an decrease in runtime of nearly 3. Of course, we are not close to full efficiency when using multiple threads, but we are still taking advantage of the BLAS 3

[2] http://www.nersc.gov/users/computational-systems/cori/configuration/

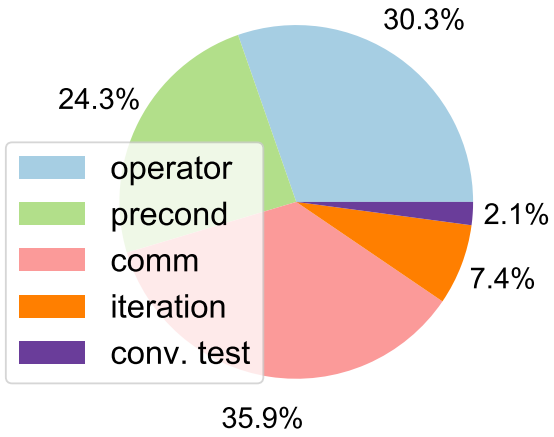


Figure 3.5 – Comparison of the time spent in various steps of one iteration of D-Odir(24) for the Ela_30 matrix with 2016 MPI processes on Kebnekaise.

routines. This is illustrated by the Table 3.4b where we compare the speed-up obtained by using threads for Omin(1), which corresponds to the classical CG, and Odir(24). We observe that using more than 2 threads is not effective at all with Omin(1) whereas it always significantly increases the speed-up with Odir(24).

Finally, we are able to obtain an overall speed-up of 50 when using 16,384 cores with respect to 64 cores. Compared to an ideal speed-up of 256, it may seem that this result is not very good (around 20% of efficiency), however it is well-known that Krylov methods may face efficiency issues at very large scale[3] — in practice, such difficulties are overcome by using preconditioning strategies well adapted to the underlying problem. Furthermore, it is important to notice that the matrices tested are relatively small, but they allow us to simulate extreme scale situation: with 16,384 cores the average number of unknowns per core is around 280. We have shown that in such cases, using enlarged Krylov subspaces allows to increase arithmetic intensity while decreasing the communication by drastically decreasing the overall iterations. Thus it takes advantage of the current trend in hardware architecture for reaching exascale.

3.5 Fusing global communications

If we break down the timings obtained during the previous strong scaling study, we observe that the major bottleneck of ECG at large scale is the communication involved by the calls to `MPI_Allreduce` (Figure 3.5). In this section, we focus on Orthodir method and we explain how to reformulate the algorithm in order to fuse these calls within an iteration.

[3]This is well illustrated by HCG benchmark: http://www.hpcg-benchmark.org/custom/index.html?lid=155&slid=293

3.5.1 Derivation of the algorithm

If we decompose one iteration of Orthodir there are 4 synchronizations,

$$\mu_k = Z_k^\top A Z_k, \tag{3.8}$$
$$\alpha_k = P_k^\top R_{k-1}, \tag{3.9}$$
$$\beta_k = \begin{pmatrix} AP_k & AP_{k-1} \end{pmatrix}^\top M^{-1} AP_k, \tag{3.10}$$
$$\xi_k = R_k^\top R_k. \tag{3.11}$$

Furthermore, we have $P_k = Z_k \mu_k^{-1/2}$. Thus we can reformulate α_k, and β_k,

$$\alpha_k = \mu^{-\top/2} Z_k^\top R_{k-1}, \tag{3.12}$$
$$\beta_k = \begin{pmatrix} AZ_k \mu^{-1/2} & AP_{k-1} \end{pmatrix}^\top M^{-1} AZ_k \mu^{-1/2}. \tag{3.13}$$
$$\tag{3.14}$$

If we postpone the convergence test, *i.e.*, we compute ξ_{k-1} instead of ξ_k, we can compute the following quantities at the same time,

$$\mu_k = Z_k^\top A Z_k, \tag{3.15}$$
$$Z_k^\top R_{k-1}, \tag{3.16}$$
$$\begin{pmatrix} AZ_k & AP_{k-1} \end{pmatrix}^\top M^{-1} AZ_k, \tag{3.17}$$
$$R_{k-1}^\top R_{k-1}, \tag{3.18}$$

and then it is possible to update α_k, β_k, P_k, and AP_k using the Cholesky factorization of μ_k.

The resulting Algorithm 17 is a *Fused* version of the preconditioned Orthodir method. The volume of communication remains constant but the different synchronizations performed at each iterations are fused in order to have only one synchronization (`MPIAllreduce` call) per iteration. Unlike *Pipelined* methods it does not overlap this global communication with computations.

It is straightforward to derive a *Fused* version of D-Odir algorithm where the number of search directions is reduced dynamically during the iterations. Indeed, we have,

$$\underline{Z}_{k+1} = AP_{k,1} - \begin{pmatrix} P_{k,1} & P_{k-1,1} & H \end{pmatrix} \beta_{k,1}, \tag{3.19}$$

where

$$\beta_{k,1} \equiv \begin{pmatrix} P_{k,1}^\top AM^{-1} AP_{k,1} \\ P_{k-1,1}^\top AM^{-1} AP_{k,1} \\ H^\top AM^{-1} AP_{k,1} \end{pmatrix}. \tag{3.20}$$

Algorithm 17 Iteration of preconditioned Fused Orthodir

1: $Q(:, 1:t) \leftarrow A * P(:, 1:t)$
2: $Z \leftarrow M^{-1} Q(:, 1:t)$
3: `// Beginning of synchronization`
4: $\mu \leftarrow P(:, 1:t)^\top Q(:, 1:t)$
5: $\alpha \leftarrow P(:, 1:t)^\top R$
6: $\beta \leftarrow Q^\top Z$
7: $\xi \leftarrow R^\top R$
8: `// End of synchronization`
9: $\mu \leftarrow \mathrm{chol}(\mu)$
10: $P(:, 1:t) \leftarrow P(:, 1:t)\mu^{-1}$
11: $Q(:, 1:t) \leftarrow Q(:, 1:t)\mu^{-1}$
12: `// Beginning of additional triangular solves`
13: $\alpha \leftarrow \mu^{-\top}\alpha$
14: $\beta \leftarrow \beta\mu^{-1}$
15: $\beta(1:t, :) \leftarrow \mu^{-\top}\beta(1:t, :)$
16: $Z \leftarrow Z\mu^{-1}$
17: `// End of additional triangular solves`
18: $X \leftarrow X + P(:, 1:t)\alpha$
19: $R \leftarrow R - Q(:, 1:t)\alpha$
20: **if** $\sum_{i=1}^{t} \xi(i, i) < \varepsilon$ **then**
21: stop
22: **end if**
23: $Z \leftarrow Z - P\beta$
24: $P(:, t+1:2t) \leftarrow P(:, 1:t)$
25: $P(:, 1:t) \leftarrow Z$
26: $Q(:, t+1:2t) \leftarrow Q(:, 1:t)$

Thus, $\beta_{k,1}$ is obtained by updating locally β_k,

$$\beta_{k,1} = \begin{pmatrix} U_{k,1}^\top & & \\ & I & \\ & & I \end{pmatrix} \beta_k U_{k,1}, \tag{3.21}$$

where $U_{k,1}$ denotes the left singular vectors of α_k that are kept in its low-rank approximation (see Section 2.5).

Remark 3.5.1: I

practice, our implementation uses the following relationship $\underline{Z}_{k+1} = Z_{k+1} U_{k,1}$. We construct Z_{k+1} using β_k and then reduce its size by multiplying it with $U_{k,1}$. It is not as optimal as the previous discussion, but it is simpler to implement and the extra flops are expected to be few with respect to the overall iteration cost.

3.5.2 Cost analysis

Following Algorithm 17, each iteration of the Fused Orthodir variant consists of 3 `tsmm` (lines 18, 19, and 23), 1 `tsmtsm` (lines 4–7), 1 `potrf` (line 9) and 6 `trsm` (lines 10, 11, and 13–16). More precisely, the LAPACK routine `gemm` is called to compute the local dot products lines 4–7, and the resulting (of dimension of the order $t \times t$) matrices are put contiguously in the memory so that the reduction is done using one call to `MPI_Allreduce`. The lines 13–16 consist of the additional updates which do not involve any communication. In summary and using the previous discussion in Section 3.3, we have,

$$\#flops(\text{Fused Odir}) = 4 \times 2\frac{nt^2}{P} + 5 \times \frac{nt(2t+1)}{P} + \frac{1}{3}t^3 + 6 \times \frac{nt^2}{P}, \tag{3.22}$$

$$= 24\frac{nt^2}{P} + 5\frac{nt}{P} + \frac{1}{3}t^3. \tag{3.23}$$

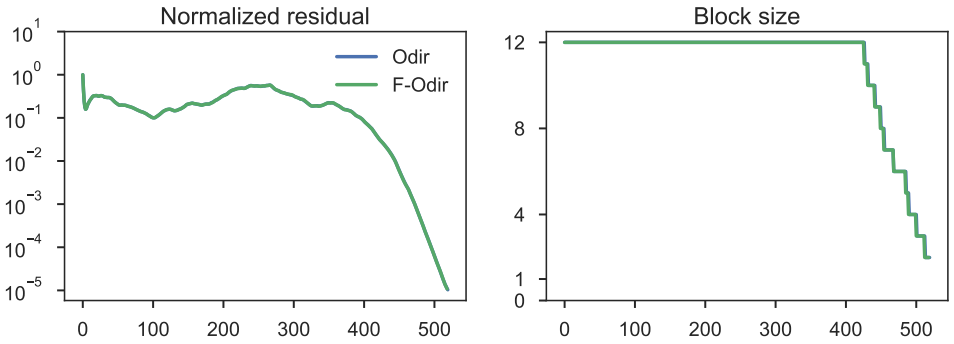
As before, matrices of size $t \times t$ are replicated among the processors, thus `tsmm`, Cholesky factorization of α and triangular solve of α are local operations without any communication. Thanks to the reformulation of the algorithm, the only communication phase occurs at lines 4–7.

In summary, the detailed costs of one iteration of Orthodir and Fused Orthodir in terms of flops, words, and messages are indicated in Table 3.8. For the sake of comparison, we recall the complexity of the CG algorithm described in [101]. We also report the number of `MPI_Allreduce` in parenthesis, in addition to the order of magnitude of the number of messages. In summary, one iteration of Fused Orthodir involves 20% more flops — neglecting the applications of the matrix, and the preconditioner — but it reduces the number of messages by factor of 4, with respect to Orthodir. Fused Orthodir also reduces the number of messages by a factor of 2 with respect to the usual CG method.

	# flops	# messages	# words	memory
Orthodir	$20\frac{nt^2}{N_{\text{proc}}} + 5\frac{nt}{N_{\text{proc}}} + \frac{1}{3}t^3$	$4\log_2(N_{\text{proc}})$ (4)	$5t^2$	$7\frac{nt}{N_{\text{proc}}} + 3t^2$
Fused Orthodir	$24\frac{nt^2}{P} + 5\frac{nt}{P} + \frac{1}{3}t^3$	$\ln(P)$ (1)	$5t^2$	$7\frac{nt}{N_{\text{proc}}} + 5t^2$
CG	$10\frac{n}{P}$	$2\log_2(N_{\text{proc}})$ (2)	2	$5\frac{n}{N_{\text{proc}}}$

Table 3.8 – Complexity of Orthodir, Fused Orthodir, and CG where t is the enlarging factor.

Figure 3.6 – Numerical comparison between dynamic Orhodir and the fused dynamic Orthodir variant on Flan_1565. The number of MPI processes is 1008 and the enlarging factor t is set to 12. We observe that the two plots coincide exactly.



In Table 3.8, we also report the memory consumption of the Fused Orthodir method. It requires an additional $2t^2$ memory space. Indeed, it requires storing at the same time the results of the 4 reductions (3.15)–(3.18). However, this memory overhead is not significant with respect to the overall memory consumption of both methods ($\mathcal{O}(\frac{nt}{N_{\text{proc}}})$).

3.5.3 Numerical experiments

In order to evaluate the gain of this reformulation, we perform several experiments both on Kebnekaise and Cori.

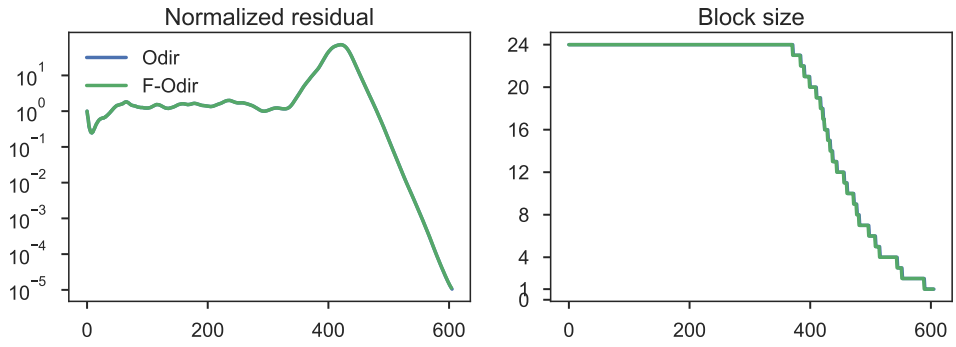
Kebnekaise

The following experiments are performed on Kebnekaise. We use the same parameters and test cases as in Section 3.4. In particular, the tolerance is set to 10^{-5}, and we use a block Jacobi preconditioner where each MPI process inverts its corresponding diagonal block of the matrix A. We do not remake the whole parameter study but rather compare the performance of the fused algorithms against the non-fused ones, and with PETSc's CG, when the number of MPI processes is relatively high.

Case	# MPI	Fused D-Odir comm	Fused D-Odir total	D-Odir comm	D-Odir total	PETSc's CG total
Hook_1498	1,008	1.1	1.6	1.4	2.1	0.9
	2,016	0.9	1.2	1.2	1.5	0.9
	3,024	0.7	0.9	2.0	2.3	1.0
Flan_1565	1,008	1.9	3.8	2.6	4.1	4.9
	2,016	1.3	2.2	2.6	3.6	2.6
	3,024	1.4	1.9	2.2	2.9	1.6
Ela_30	1,008	11.1	21.7	12.2	23.7	94.7
	2,016	6.9	13.0	8.2	14.5	34.5
	3,024	8.0	11.9	7.7	11.6	19.5

Table 3.9 – Timings in seconds of the fused dynamic Orthodir variant with D-Odir and PETSc's CG. The enlarging factor t is set to 8 for Hook, 12 for Flan, and 24 for Ela_30, and "comm" stands for the time spend in the global communication during the iterations of ECG.

Figure 3.7 – Numerical comparison between dynamic Orhodir and the fused dynamic Orthodir variant on Ela30. The number of MPI processes is 1008 and the enlarging factor t is set to 24.



In Table 3.9, we summarize the results obtained when we compare the Fused D-Odir method with D-Odir, and PETSc's CG. More precisely, we compare these 3 methods on Hook_1498, Flan_1565, and Ela_30 respectively using the same parameters as in the strong scaling study, and $t = 8$ for Hook, 12 for Flan, and 24 for Ela_30. Thus in any case the Fused version is faster than the non–Fused one. Indeed, it does not suffer from numerical instabilities and the number of iterations remains exactly the same (see Figures 3.6 and 3.7) but the time spent in the communication is reduced. For example, with the largest number of MPI processes, the Fused variant is more than twice faster than the non–Fused variant for the Hook test case, and around 1.5 times faster than the non–Fused variant for the Flan test case. Depending on the test case, fusing the global communications even allows us to be slightly faster than PETSc's CG at large scale; this is the case for Hook, but not for Flan.

In summary, fusing the global communications almost always leads to a decrease in the runtime with respect to the non–Fused variant. This decrease is significant at large scale because the communication phase usually dominates the overall runtime. Furthermore, the Fused variant can also be slightly better than PETSc's CG when the MPI processes count is the highest (3024).

Cori

We now make the exact same experiments on Cori using KNL processors: both the matrices and the parameters are the same as just before. In all the experiments we use 2 threads per MPI processes, each one being bound to one physical core. It means that the total number of processors used is # MPI $\times 2$. Once again, we are interested in comparing the performance of the fused algorithms against the non-fused ones, and with PETSc's CG, when the number of MPI processes is relatively high: starting to $2,048$, and up to $16,385$ — meaning up to $32,768$ cores at the largest scale.

We summarize the results we obtained in Table 3.10. As before, we compare the Fused D-Odir method with D-Odir and PETSc's CG on Hook_1498, Flan_1565, and Ela_30. For Cori, a much larger machine than Kebnekaise, we use a larger number of MPI processes in these experiments. For the Hook test case, we observe that the Fused variant is always faster than PETSc, unlike the non–Fused. Indeed, the communication time is drastically reduced: up to a factor almost 2 for the largest count of MPI processes we considered. For the Flan matrix, the same behavior is observed. The Fused variant allows to obtain a speed–up for the largest count of MPI processes considered with respect to PETSc's CG. Thus it scales slightly better than PETSc's CG. Finally for the Ela_30 test case, we observe a significant improvement of the scalability on the largest count of MPI processes. Fused D-Odir is indeed 33% faster using $16,384$ MPI processes than when using $8,192$. On the other hand, PETSc's CG is around 8% times faster using $16,384$ MPI processes than when using $8,192$. As a consequence, the Fused D-Odir method is almost 4 times faster than PETSc's CG when using $16,384 \times 2 = 32,768$ physical processors. In summary, we observe that the Fused variant is always faster that the non–Fused one. However, on this machine the Fused variant is at least as fast (for the Flan test case) as PETSc's CG when the number of MPI

		Fused D-Odir		D-Odir		PETSc's CG
Case	# MPI	comm	total	comm	total	total
Hook_1498	2,048	1.6	3.0	2.3	3.8	3.4
	4,096	1.8	2.8	2.7	3.8	3.4
	8,192	0.8	1.4	1.4	2.2	1.6
Flan_1565	2,048	2.0	5.5	3.0	6.4	5.5
	4,096	1.6	3.7	2.6	4.9	3.7
	8,192	1.4	2.9	2.4	4.1	3.2
Ela_30	2,048	18.2	32.4	24.6	39.0	151.7
	4,096	6.5	14.3	8.3	16.2	53.7
	8,190	5.1	11.2	7.0	13.5	34.6
	16,384	4.0	8.0	6.3	10.1	31.7

Table 3.10 – Timings in seconds of the fused dynamic Orthodir variant with D-Odir and PETSc's CG. The enlarging factor t is set to 8 for Hook, 12 for Flan, and 24 for Ela_30, and "comm" stands for the time spend in the global communication during the iterations of ECG.

processes is relatively low (2,048 and 4,096), but slightly faster than PETSc when using more than 8,192 MPI processes. Thus, the Fused variant scales better than PETSc's CG on this machine, and this scalability has been assessed up to a large number of MPI processes (16,384 at the largest scale).

3.6 Reproducibility of the numerical experiments

In this section, we provide the informations needed to reproduce the results presented in the previous section. We start by exposing some technical details of the implementation. Then, we explain how to install and run the executables we used. We also explain how to generate the elasticity matrices using Freefem++. Eventually, we provide further practical details that allow to reproduce our results on KNL processors.

3.6.1 Implementation details

Our implementation of ECG is based on Reverse Communication Interface [41] and written in C and MPI. Following this scheme we provide 4 routines:

- `preAlps_ECGInitialize(preAlps_ECG_t* ecg, double* rhs, int* rci_request)`,
- `preAlps_ECGIterate(preAlps_ECG_t* ecg, int* rci_request)`,
- `preAlps_ECGStoppingCriterion(preAlps_ECG_t* ecg, int* stop)`,
- `preAlps_ECGFinalize(preAlps_ECG_t* ecg, double* solution)`.

In order to ease its usage, we encapsulate all the required information by ECG in the structure `preAlps_ECG_t`. More precisely, this structure is defined as,

```
typedef struct {
  /* Input variable */
  double* b;  /* Right hand side */

  /* Internal symbolic variables */
  CPLM_Mat_Dense_t* X; /* Approximated solution */
  CPLM_Mat_Dense_t* R; /* Residual */
  CPLM_Mat_Dense_t* V; /* Descent directions ([P,P_prev] or P) */
  CPLM_Mat_Dense_t* AV; /* A*V */
  CPLM_Mat_Dense_t* Z; /* Preconditioned R (Omin) or AP (Odir) */
  CPLM_Mat_Dense_t* alpha; /* Descent step */
  CPLM_Mat_Dense_t* beta; /* Step to construt search directions */

  /** User interface variables */
  CPLM_Mat_Dense_t* P; /* Search directions */
  CPLM_Mat_Dense_t* AP; /* A*P */
  double* R_p; /* Residual */
  double* P_p; /* Search directions */
  double* AP_p; /* A*P_p */
  double* Z_p; /* Preconditioned R (Omin) or AP (Odir) */

  /** Working arrays */
  double* work;
  int* iwork;

  /* Single value variables */
  double normb; /* norm_2(b) */
  double res; /* norm_2 of the residual */
  int iter; /* Iteration */
  int bs; /* Block size */
  int kbs; /* Krylov basis size */

  /* Options and parameters */
  int globPbSize; /* Size of the global problem */
  int locPbSize; /* Size of the local problem */
  int maxIter; /* Maximum number of iterations */
  int enlFac; /* Enlarging factor */
  double tol; /* Tolerance */
  preAlps_ECG_Ortho_Alg_t ortho_alg; /* A-ortho algorithm */
  preAlps_ECG_Block_Size_Red_t bs_red; /* Block size reduction */
  MPI_Comm comm; /* MPI communicator */
} preAlps_ECG_t;
```

For the sake of simplicity, we did not add their explicit definition here, but `Ortho_Alg_t` and `Block_Size_Red_t` are `enum`. The structure `CPLM_Mat_Dense_t` represents local

dense matrices, it contains the pointer to the data (`val`), the number of rows (`info.m`), the number of columns (`info.n`) and an `enum` to specify the matrix storage type (`ROW_MAJOR` or `COL_MAJOR`).

First, the user has to declare a variable of type `preAlps_ECG_t` and set values for the parameters: `comm`, `globPbSize`, `locPbSize`, `maxIter`, `enlFac`, `tol`, `ortho_alg`, and `bs_red` (`ADAPT_BS` for dynamic reduction of the search directions, and `NO_BS_RED` otherwise). The variable `globPbSize` corresponds to the number of unknowns of the global solution x (the dimension of A) and `locPbSize` corresponds to the number of unknown owned locally (the number of rows of A that are stored locally).

Then, in order to allocate memory and initialize the structure, the user has to call, `preAlps_ECGInitiliaze(&ecg, rhs, &rci_request)`, where `rhs` is a `double *` array representing the right hand side and `rci_request` is an integer. After this call, he has to apply the preconditioner to `ecg.R` and put the result into `ecg.P`. And then, he has to apply the operator A to `ecg.P` and put the result into `ecg.AP`. These two operations has to be executed in parallel assuming that `ecg.R`, `ecg.P`, `ecg.AP` contains local rows of R, P and AP which are distributed as row panel over the processors.

Afterwards, the user has to call `preAlps_ECGIterate(&ecg,&rci_request)` until convergence of the method. Following RCI scheme, after this call the user has to check the value of `rci_request`. If `rci_request = 0`, then the user is requested to apply A on `ecg.P` and put the resul into `ecg.AP`. If `rci_request = 1`, then the user can check for convergence of the method. If the method did not converge, then depending on the choice of the orthogonalization algorithm (`ORTHODIR` or `ORTHOMIN`) he has to apply the preconditioner to `ecg.R` (`ORTHOMIN`) or `ecg.AP` (`ORTHODIR`) and put the result into `ecg.Z`.

When convergence is reached, it is possible to recover the solution and free the structure `ecg` by calling `preAlps_ECGFinalize(&ecg,sol)` where `sol` is a `double*` array already allocated.

To sum up, for solving a linear system with a block Jacobi preconditioner, the general calling sequence will be

```
// Set parameters
ecg.comm = MPI_COMM_WORLD; /* MPI Communicator */
ecg.globPbSize = M; /* Size of the global problem */
ecg.locPbSize = m; /* Size of the local problem */
ecg.maxIter = maxIter; /* Maximum number of iterations */
ecg.enlFac = 2; /* Enlarging factor */
ecg.tol = tol; /* Tolerance of the method */
ecg.ortho_alg = ORTHODIR; /* Orthogonalization algorithm */
ecg.bs_res = ADAPT_BS; /* Reduce number of search directions */
// Allocate memory and initialize variables
preAlps_ECGInitialize(&ecg,rhs,&rci_request);
// Finish initialization
preAlps_BlockJacobiApply(ecg.R,ecg.P);
preAlps_BlockOperator(ecg.P,ecg.AP);
```

```
// Main loop
while (stop != 1) {
  ierr = preAlps_ECGIterate(&ecg,&rci_request);
  if (rci_request == 0) {
    // AP = A*P
    preAlps_BlockOperator(ecg.P,ecg.AP);
  }
  else if (rci_request == 1) {
    ierr = preAlps_ECGStoppingCriterion(&ecg,&stop);
    if (stop == 1) break;
    if (ecg.ortho_alg == ORTHOMIN)
      // Z = M^-1*R
      preAlps_BlockJacobiApply(ecg.R,ecg.Z);
    else if (ecg.ortho_alg == ORTHODIR)
      // Z = M^-1*AP
      preAlps_BlockJacobiApply(ecg.AP,ecg.Z);
  }
}
// Retrieve solution and free memory
preAlps_ECGFinalize(&ecg,sol);
```

3.6.2 Installation and usage

The software can be downloaded here: https://who.rocq.inria.fr/Olivier.Tissot/ecg_code.zip or upon request by email. There is no particular hardware dependency, however, there are some software dependencies:

- C99, with or without OpenMP.
- MPI: we used both IntelMPI (Kebnekaise) and Cray MPICH (Cori).
- Intel MKL: we use BLAS and LAPACK routines, as well as PARDISO for applying the block Jacobi preconditioner.
- METIS: it is needed to partition the matrix before starting the computation in order to increase the load-balancing.
- PETSc: it is only used for the purpose of benchmarking, and the implementation does not rely on it.

The SuiteSparse matrices are available online. For generating the elasticity matrices, one needs to install FreeFem++, modify the file `ff++/elasticity-3d.edp` and then run the following command:

```
cd ff++ && FreeFem++ elasticity-3d.edp -nw
```

The output is not exactly MatrixMarket compliant and it is necessary to call the following patch to convert the output:

```
./ff++_to_mtx.sh ${matrix_filename}
```

Light installation without PETSc:

1. unzip the archive
2. modify the `make.inc` in order to link MPI, the MKL, METIS and deactivate the other dependencies
3. install and configure CPaLAMeM (distributed with the code):

   ```
   make install_cpalamem
   ```

   and follow the instructions
4. call `make` in the root directory.

Full installation with PETSc:

1. install PETSc 3.7.6 using the following configuration:

   ```
   ./configure -with-cc=mpiicc --with-cxx=0 --with-fc=0 COPTFLAGS="-O3
   ↪  -march=native -mtune=native" --with-debugging=0
   ↪  -with-blas-lapack-dir=${MKLROOT}
   ↪  --with-mkl_pardiso-dir=${MKLROOT} --download-metis
   ```

2. unzip the archive
3. modify the `make.inc` in order to link MPI, the MKL, PETSc, METIS and deactivate the other dependencies
4. install and configure CPaLAMeM (distributed with the code):

   ```
   cd utils/
   tar -xvzf CPALAMEM_2017-10-06-10_59_19.tar.gz
   cd cpalamem
   ./configure --cc mpiicc --with-metis --with-mkl --with-petsc -O
   ```

   and follow the instructions
5. call `make` in the root directory.

Examples of `make.inc` files used on Kebnekaise and Cori are available in the directory `install_examples`.

Once the compilation is finished, the executables `ecg_prealps_op` and `ecg_bench_petsc_pcg` should be in the `bin` directory. Please note that if the code has not been linked with PETSc `ecg_bench_petsc_pcg` will not do anything.

Both takes the same parameters as arguments (after a space) during the call:

- `-e` the enlarging factor (cannot exceed the number of MPI processes)
- `-i`: the maximum number of iterations
- `-m`: the file containing the matrix (MatrixMarket format)
- `-o`: if 0 Orthodir is used, if 1 Orthomin is used
- `-r`: if 0 NO dynamic reduction of the search directions, if 1 dynamic reduction of the search directions
- `-t`: the tolerance of the method.

For instance, in order to reproduce the result at row 3 and column 2 (D-Odir(8)) in Table 3.4 the call is:

```
mpirun -np 56 ./bin/ecg_prealps_op -m Flan_1565.mtx -e 8 -o 0 -r 1 -t
↪  1e-5 -i 5000
```

3.6.3 Evaluation and expected result

In order to check that the light installation (without PETSc) is working, it is possible to try to reproduce the results of Table 3.4 as shown in the previous subsection.

Similarly, it is possible to try to reproduce the results in Table 3.5 to check that the full installation is working. For instance, the call to reproduce the results in row 1 of Table 3.5 is:

```
mpirun -np 252 ./bin/ecg_bench_petsc_pcg -m
↪  elasticite3d_1600x30x30_var.mtx -e 24 -o 0 -r 1 -t 1e-5 -i 25000
```

Examples of submission scripts used on Kebnekaise and Cori are available in the directory `scripts`.

Please note that the case where t is larger than the number of MPI processes is not handled; an error message is output. Nevertheless, it is possible to set t equals to 1 and in that case the method corresponds to the usual CG. Thus, a simple debug test is the following call:

```
mpirun -np 4 ./bin/ecg_bench_petsc_pcg -m matrix/1DLaplacian16.mtx -e 1
↪  -o 0 -r 0 -t 1e-2 -i 16
```

The expected output should contain the following lines:

```
=== Matrix informations ===
        size: 16
        nnz : 46
=== Petsc ===
```

```
        iterations: 6
        norm(res): 5.310086e-03
=== ECG ===
        iterations: 6
        norm(res) : 5.310086e-03
        block size: 1
```

It solves a simple 1D Laplacian linear system of size 16 with PETSc's CG and with Omin(1). Both PETSc and ECG converge within the same number of iterations and both get the same approximate residual norm. Please note that in practice, the norm of the approximate residuals are not necessarily the same for every matrix because of the round-off errors.

On Cori, we used the following submission parameters:

```
#SBATCH -C knl,quad,cache
# core specialization
#SBATCH -S 4

export OMP_PLACES=threads
export OMP_PROC_BIND=true
```

We also did not use hyperthreading. For example, the submission script to reproduce the result in Table 3.4b is:

```
#!/bin/bash -l
#SBATCH -N 256
#SBATCH -C knl,quad,cache
# core specialization
#SBATCH -S 4
#SBATCH -q regular
#SBATCH -J ecg
#SBATCH -t 02:30:00
#SBATCH -L SCRATCH

export OMP_NUM_THREADS=8
export MKL_NUM_THREADS=8
export OMP_PLACES=threads
export OMP_PROC_BIND=true

srun -n 2048 -c 32 --cpu_bind=cores ./bin/ecg_prealps_op -m
 ↪  elasticite3d_1600x30x30_var.mtx -e 24 -o 0 -r 1 -i 35000 >
 ↪  res/ecg.log 2>&1
```

During the setup phase, the matrix is read by one MPI process (rank 0). It partitions the matrix using METIS and then distributes it row-wise among the other MPI processes. Of course, this setup phase can be quite time consuming if the matrix be-

comes large, especially on KNLs.

When the number of processes becomes large (> 100) and the number of iterations is high (> 1000), it is very likely that the number of iterations will not be exactly the same from one run to another. This is due to the numerical sensibility of the CG method to round-off errors and the fact that MPI collectives are not reproducible. In practice, for our test cases, the difference is not relevant and does not have any significant impact.

4

Recycling strategies

Outline of the current chapter

Abstract

In this chapter, we study so–called recycling strategies in order to increase the efficiency of the linear solver in the context of the Cosmic Microwave Background (CMB) analysis. We recall several methods for computing eigenvalues and the associated eigenvectors of a matrix from an already computed Krylov basis and how to deflate them, *i.e.*, remove their possibly bad effect on the convergence of the Krylov method. These two ingredients are the basis of the general framework of the recycling methods presented. Numerical experiments are performed on problems coming from the CMB analysis, and the efficiency of the methods is assessed from a qualitative point of view. The multiplicity of the smallest eigenvalue of the underlying matrix

may have a bad effect on the efficiency of these methods. In order to overcome this difficulty, we proposed a cheap procedure to adapt the initial guess before the solve, which permits to reduce the overall number of iterations.

Résumé

Dans ce chapitre, nous étudions les stratégies dites de recyclage afin d'améliorer l'efficacité de la résolution du système linéaire dans le contexte de l'analyse des observations du fond diffus cosmologique (CMB). Nous rappelons plusieurs méthodes pour calculer les valeurs propres et les vecteurs propres associés d'un matrice à partir d'une base de Krylov déjà calculée et comment les déflater, *i.e.*, masquer leur effet éventuellement négatif sur la convergence de la méthode de Krylov. Ces deux ingrédients sont la base du cadre général des méthodes de recyclage que nous présentons ensuite. Des expériences numériques sont menées sur des problèmes provenant de l'analyse du CMB and l'efficacité des méthodes est évaluée d'un point de vue qualitatif. La multiplicité de la plus petite valeur propre de la matrice sous–jacente peut avoir un effet négatif sur ces méthodes. Afin de surmonter cette difficulté, nous proposons finalement une procédure peu coûteuse pour adapter la solution initiale avant la résolution, qui permet de réduire le nombre total d'itérations.

4.1 Motivation — application to CMB data analysis

The Cosmic Microwave Background (CMB) has been extensively studied over the last decades leading to remarkable progress in our understanding of the beginning of the Universe. The amount of data collected by the observatories has been increasing tremendously, and its analysis currently requires the optimized usage of the most modern supercomputers. Our focus will be on the parametric component separation in CMB experiments which relies on the solution of very large linear systems.

4.1.1 The map–making problem

We start by explaining the so–called map–making problem [94] as it is an important building–block of the parametric component separation. At a given time the detectors of a CMB experiment are performing N_{data} measurements; the resulting vector is denoted d. These measurements are modeled as the sum of an unknown astrophysical map s[1] and an instrumental noise n. This map s is a vector of $\mathcal{N}$ pixels, each of these being itself Stokes parameters that describe the polarization state of electromagnetic radiation. We want to point out that the number of data collected is in general much larger than the number of pixels in the unknown map ($\mathcal{N} \ll N_{\text{data}}$). The physical strategy that makes the link between the data collected and the unknown map is encoded into the pointing matrix P. This matrix is tall and skinny, and it is usually very sparse because there are few parameters to estimate for each pixel, *e.g.*, 3 parameters would mean 3 non-zeros per row [94]. We are eventually able to write the unknown map as a function of the data and the noise,

$$d = Ps + n. \tag{4.1}$$

[1] It is denoted s because it corresponds to a signal.

It is not possible to solve this problem in this form, of course, because the noise n is not known. In order to overcome this difficulty, it is usually assumed that the noise can be modeled by a random variable with a vanishing mean and a noise covariance matrix denoted N. Thus (4.1) becomes a linear statistical problem whose solution is given by solving a generalized least square equation,

$$P^\top \mathsf{N}^{-1} P s = P^\top \mathsf{N}^{-1} d. \tag{4.2}$$

If the noise is moreover assumed to be Gaussian, one can show that the solution is an unbiased estimator with minimum variance for the unknown quantity s [94].

In real experiments the number of data collected can reach $\mathcal{O}(10^{12-15})$ whereas the number of pixels in the map is $\mathcal{O}(10^6)$. Since the matrix N is then a square matrix of the dimension of the data, one may wonder how to compute N^{-1}. This difficulty is usually overcome by assuming that N has a special structure, *e.g.*, it is a (block) circulant matrix, such that its inverse can be applied at an almost linear cost through Fast Fourier Transform. However, given the size of the unknown map, it is infeasible to assemble the matrix of the system (4.2) because the product $\mathsf{N}^{-1}P$ would be too costly to compute. That is why the traditional method to tackle this problem is to use a standard CG method with the following Jacobi–like preconditioner,

$$M^{-1} = (P^\top \mathrm{diag}(\mathsf{N}^{-1}) P)^{-1}. \tag{4.3}$$

This very simple preconditioner has shown limitations in some cases, especially when the noise has high off–diagonal contributions, and a two–level approach has been found successful in such cases [60, 94, 114].

4.1.2 The parametric component separation (PCS) problem

We now review the method proposed in [110] which relies on a Maximum likelihood algorithm. The main idea is to find a way to mix the component to be estimated from the data. More precisely, the previous framework is slightly generalized by considering the following problem,

$$d = P_\beta s + n, \tag{4.4}$$

where the matrix P, that encodes the link between the Stokes parameters and the data, is now depending on a set of unknown parameters $\{\beta_i\}$. Moreover, d now contains the data collected for different frequencies, and the solution s contains some Stokes parameters according to these frequencies.

In [110], the authors propose to fit both s and β according to the data. More precisely, they maximize the corresponding log-likelihood which reads,

$$-2\ln(\mathcal{L}(s,\beta)) = C + \left(d - P_\beta s\right)^\top \mathsf{N}^{-1} \left(d - P_\beta s\right), \tag{4.5}$$

with the same notations as before, and C denotes an constant. For each step of the

maximization process one has to solve the following linear system of equations,

$$P_\beta^\top \mathsf{N}^{-1} P_\beta s = P_\beta^\top \mathsf{N}^{-1} d. \tag{4.6}$$

which means that $P_\beta \equiv PM_\beta$, where P is a pointing matrix as in the map–making problem, *i.e.*, a very sparse tall and skinny matrix, and M_β is called the mixing matrix. Thus (4.6) eventually reads

$$M_\beta^\top P^\top \mathsf{N}^{-1} P M_\beta s = M_\beta^\top P^\top \mathsf{N}^{-1} d, \tag{4.7}$$

In our case, the mixing matrix M_β transforms the full signal s that consists of 3×2 components for each pixel into a mixed signal that consists of 2 components for each pixel. More precisely, given (u_i, q_i) with $1 \leqslant i \leqslant 3$ that represents the 3×2 components for the full sky, *i.e.*, u_i, $q_i \in \mathbb{R}^{\mathcal{N}}$, we have the following

$$M_\beta \begin{pmatrix} u_1 & q_1 \\ u_2 & q_2 \\ u_3 & q_3 \end{pmatrix} = \begin{pmatrix} \alpha_1 u_1 + \alpha_2 u_2 + \alpha_3 u_3 & \gamma_1 q_1 + \gamma_2 q_2 + \gamma_3 q_3 \end{pmatrix}, \tag{4.8}$$

where α_i and γ_i $(1 \leqslant i \leqslant 3)$ are some weights that depends on β. Thus, M_β has the following tensorized structure

$$M_\beta = \begin{pmatrix} \alpha_1 I & \alpha_2 I & \alpha_3 I \\ \gamma_1 I & \gamma_2 I & \gamma_3 I \end{pmatrix} \tag{4.9}$$

$$= \begin{pmatrix} \alpha_1 & \alpha_2 & \alpha_3 \\ \gamma_1 & \gamma_2 & \gamma_3 \end{pmatrix} \otimes I \tag{4.10}$$

$$\equiv K_\beta \otimes I, \tag{4.11}$$

where $\otimes$ denotes the Kronecker product and I denotes the identity matrix of order $\mathcal{N}$.

4.1.3 The algebraic framework

In fact, the problem (4.6), *i.e.*, solving a sequence of linear algebraic systems, is of interest in many applications. When the system matrices are in some suitable sense close to each other ("changing slowly"), exploiting this property can bring significant computational effort and energy savings. One can aim at building an efficient (yet more computationally costly) preconditioner for all the system matrices. However, this may not be always possible and recomputation of the preconditioner may increase the overall cost significantly. Another idea, present in the literature and put forward in this chapter, is to recycle the information from solution of each system to the next system. Such recycling is straightforward to apply in the context of projection-type methods, represented in what follows by Krylov methods with a two–level preconditioner.

There exists a plethora of methods and ideas that can be used in the solution of a sequence of linear algebraic systems. It seems natural that an effective procedure must be properly tuned for a particular application. We recall the methods for solving sequences of systems proposed in the literature, present a unified framework, and

comment on possible choices of particular ingredients of the solution procedure.

Whenever possible, our presentation will concern systems with general (nonsingular) matrices and a general (Krylov) solver. However, since the system matrix in (4.6) is symmetric positive definite (SPD), we will later focus on sequences with SPD matrices and (preconditioned) conjugate gradient solver. To clearly distinguish when the discussion assumes that the system matrix A and the preconditioner M are SPD, we use in such case sans serif font, giving A and M, respectively. We will also discuss in particular the preconditioning from left, which is mostly used in the context of solving (4.6).

Remark 4.1.1

Each system taken independently fits in the framework of the ECG method, however we are interested in taking into account the fact that these systems are somehow related to each others. As explained before, the approach relies on the construction on-the-fly of a two–level preconditioner. We have shown in Chapter 2 that ECG is acting as a two–level preconditioner, but its application is also more costly. Thus, it is very likely that, except for the first system for which there is no two–level preconditioner available yet, the usual CG method used in conjunction with a two–level preconditioner is more effective than the ECG method.

4.2 Ingredients of the methods

To facilitate the forthcoming description of the methods, we present in this section two ingredients of the methods. Namely, the ways how the eigenpairs are estimated using the computed basis of Krylov subspace and the ways how the deflation of the approximate eigenvectors can be combined with another preconditioner.

4.2.1 Eigenvalues approximation using Krylov subspace methods

Arnoldi and Lanczos algorithms for approximating the eigenvalues of a general nonsingular or, respectively, a Hermitian matrix, are based on Rayleigh-Ritz approximation that will be presented first. Then, we recall the Arnoldi and Lanczos algorithms and, finally, we briefly comment on their restarted variants.

The methods discussed below do not represent an exhaustive overview of methods for approximating several eigenvalues and the associated eigenvectors. Among the omitted methods, there is, *e.g.*., the Jacobi–Davidson method [103], which proved to be particularly efficient for approximating the inner part of the spectrum. For a survey on the methods and a list of references see, *e.g.*., [106].

Ritz values and harmonic Ritz values approximations

For a subspace $\mathcal{S} \subset \mathbb{C}^n$, we call $y \in \mathcal{S}$ a *Ritz vector* of A with *Ritz value* θ if

$$Ay - \theta y \perp \mathcal{S}.$$

Using a (computed) basis V_j of $\mathcal{S}$ and setting $y = V_j w$, the above relation is equivalent to solving

$$V_j^\top A V_j w = \theta V_j^\top V_j w. \tag{4.12}$$

Ritz values are known to approximate well the extremal eigenvalues of A. If an approximation to the interior eigenvalues is required, computing the harmonic[2] Ritz values can be preferable. Following [93], we define harmonic Ritz values as the Ritz values of A^{-1} with respect to the space $A\mathcal{S}$,

$$\widetilde{y} \in A\mathcal{S}, \quad A^{-1}\widetilde{y} - \widetilde{\mu}\widetilde{y} \perp A\mathcal{S}.$$

We call $\widetilde{\theta} \equiv 1/\widetilde{\mu}$ a *harmonic Ritz value* and $\widetilde{y}$ a *harmonic Ritz vector*. If V_j is a basis of $\mathcal{S}$ and $\widetilde{y} = V_j \widetilde{w}$, the above relation can be represented as

$$V_j^\top A^\top V_j \widetilde{w} = \widetilde{\mu} V_j^\top A^\top A V_j \widetilde{w} \quad \Longleftrightarrow \quad V_j^\top A^\top A V_j \widetilde{w} = \widetilde{\theta} V_j^\top A^\top V_j \widetilde{w}. \tag{4.13}$$

For the properties of the harmonic Ritz values approximations and the relationship with the iteration polynomial in MINRES method, see [91].

There are various ways how the harmonic (Rayleigh–)Ritz procedure is presented and defined in the literature; often it is introduced to approximate eigenvalues close to a target $\tau \in \mathbb{C}$. For example, [121] prescribes the procedure by

$$\widetilde{y} \in \mathcal{S}, \quad A\widetilde{y} - \widetilde{\theta}\widetilde{y} \perp (A - \tau I)\mathcal{S}, \tag{4.14}$$

where I is the identity matrix. With $\widetilde{y} = V_j \widetilde{w}$ this corresponds to the generalized eigenvalue problem

$$V_j^\top (A - \tau I)^\top (A - \tau I) V_j \widetilde{w} = \left(\widetilde{\theta} - \tau\right)\left(V_j^\top (A - \tau I)^\top V_j\right)\widetilde{w},$$

which becomes for $\tau = 0$ exactly the right equality in (4.13).

We note that harmonic Ritz approximation is often used in the Krylov subspace recycling methods also for approximating the smallest (in magnitude) eigenvalues and the associated eigenvectors; see the description of the methods in the corresponding section below.

Finally, we comment on the (harmonic) Ritz approximation in the case we want to compute the eigenvalues of the matrix A preconditioned from the left by M. In a general case, assuming only that A, M are nonsingular, the Ritz and harmonic Ritz approximations are applied as above just replacing in the formulas A by $M^{-1}A$. When the matrix A is Hermitian and the preconditioner M is SPD, there is also another option. First, we note that the matrix $\mathsf{M}^{-1}A$ is not Hermitian but it is self-adjoint with respect to the inner product induced by M, that is

$$(v, \mathsf{M}^{-1}Aw)_{\mathsf{M}} = (\mathsf{M}^{-1}Av, w)_{\mathsf{M}}, \qquad \forall v, w,$$

[2] The term *harmonic Ritz values* was introduced in [91], where the references to previous works using this approximation can be found.

where $(v,w)_{\mathsf{M}} \equiv v^\top \mathsf{M} w$. This allows, in the definition of Ritz and harmonic Ritz approximation, to replace A by $\mathsf{M}^{-1}A$ and the standard inner product by the inner product induced by the matrix M, giving

$$y \in \mathcal{S}, \quad \mathsf{M}^{-1}Ay - \theta y \perp_{\mathsf{M}} \mathcal{S}, \quad \text{respectively} \quad \widetilde{y} \in \mathsf{M}^{-1}A\mathcal{S}, \quad (\mathsf{M}^{-1}A)^{-1}\widetilde{y} - \widetilde{\mu}\widetilde{y} \perp_{\mathsf{M}} \mathsf{M}^{-1}A\mathcal{S}.$$

The corresponding algebraic problems with $y = V_j w$, $\widetilde{y} = V_j \widetilde{w}$, are

$$V_j^\top A V_j w = \theta V_j^\top \mathsf{M} V_j w, \quad \text{respectively} \quad V_j^\top A^\top \mathsf{M}^{-1} A V_j \widetilde{w} = (1/\widetilde{\mu}) V_j^\top A^\top V_j \widetilde{w}.$$

Note that the problems above involve Hermitian matrices only.

Arnoldi and Lanczos methods

Arnoldi and Lanczos algorithms for approximating the eigenvalues of a general nonsingular or, respectively, a Hermitian matrix, are based on Ritz approximation with setting $\mathcal{S} = \mathcal{K}_j(A, v_1) = \operatorname{span}(v_1, Av_1, \ldots, A^{j-1}v_1)$, the jth Krylov subspace. The methods compute an orthogonal basis V_j of $\mathcal{S}$ such that

$$AV_j = V_j T_j + \beta v_{j+1} e_j^\top,$$

where e_j is the last column vector of the identity matrix (of size j) and $V_j^\top V_j = I$, $V_j^\top v_{j+1} = 0$. Consequently, the eigenvalue problem (4.12) corresponding to the Ritz approximation reads

$$T_j w = \theta w.$$

The matrix T_j is available during the iterations. The standard use of Arnoldi and Lanczos method for eigenvalue approximation consists of solving the above problem and setting the pairs $(\theta, V_j w)$ as the computed approximations.

One can naturally replace the Ritz approximation by the harmonic Ritz approximation. Then, the matrices in the problem (4.13) become

$$V_j^\top A^\top A V_j = T_j^\top T_j + \beta^2 e_j e_j^\top, \qquad V_j^\top A^\top V_j = T_j^\top.$$

The Lanczos algorithm is a variant of the Arnoldi algorithm for a Hermitian A. The matrix $T_j = V_j^\top A V_j$, which is in Arnoldi method upper Hessenberg, is then also Hermitian. Consequently, it is tridiagonal, which means that in each step of Lanczos method we orthogonalize the new vector only against the two previous vector. This ensures that the computational cost of each iteration is fixed and, if one is interested in approximating the eigenvalues only, storing three vectors v_{j-1}, v_j and v_{j+1} is sufficient instead of handling the full matrix V_j. The assumption on exact arithmetic is, however, crucial here. In finite precision computations, the global orthogonality is typically quickly lost, which can cause several stability issues [90].

As noted above, an orthonormal basis V_j of $\mathcal{S}$ is advantageous for the Ritz approximation. For the harmonic Ritz approximation applied to an SPD matrix A, one

can instead aim at constructing an A-orthonormal basis, which assures that the matrix $V_j^\top \mathsf{A}^\top V_j = V_j^\top \mathsf{A} V_j$ on the right-hand side of (4.13) is equal to the identity. An A-orthonormal basis of a Krylov subspace can be constructed within the iterations of conjugate gradient method by using the search direction vectors.

The Arnoldi method can be naturally applied also to the preconditioned matrix $M^{-1}A$ to compute an orthonormal basis V_j of the associated Krylov subspace $\mathcal{K}_j(M^{-1}A, M^{-1}v_1)$, giving

$$M^{-1}AV_j = V_jT_j + \beta v_{j+1}e_j^\top, \qquad V_j^\top V_j = I, \quad V_j^\top v_{j+1} = 0.$$

For a Hermitian A and an SPD preconditioner M, we can apply the Lanczos method following the comment made above – using the matrix $\mathsf{M}^{-1}A$ and the inner product induced by M instead of the standard euclidean one, giving

$$\mathsf{M}^{-1}AV_j = V_jT_j + \beta v_{j+1}e_j^\top, \qquad V_j^\top \mathsf{M} V_j = I, \quad V_j^\top \mathsf{M} v_{j+1} = 0.$$

The computed basis V_j is therefore M-orthonormal.

Restarted variants

The number of iterations necessary to converge is not a priori known in Arnoldi and Lanczos algorithms and, in general, it can be very high. High iteration counts require a large memory to store the basis vectors and, whenever a full-reorthogonalization is used, a high computational effort because of the growing cost of the reorthogonalization in each step. The idea behind implicitly restarted variants is to limit the dimension of the search space $\mathcal{S}$. This means that the iterations are stopped after a (fixed) number of steps, the dimension of the search space is reduced while maintaining its (Krylov) structure, and the Arnoldi/Lanczos iterations are resumed.

There are several restarted variants described in the literature (a detailed description is, however, beyond the scope of this chapter): the implicitly restarted Arnoldi (IRA, [105]), the implicitly restarted Lanczos (IRL, [16]), or the Krylov–Schur method ([109, 122]).

The estimation of the spectrum of A is possible within the GMRES, MINRES and CG iterations (applied to solve a system with A) because they are based on Arnoldi, respectively Lanczos algorithms. In contrast, a combination of restarted variants with solving a linear algebraic system is, to the best of our knowledge, not described in the literature. We therefore do not consider the restarted variants in the sequel.

4.2.2 Deflation and two–level preconditioners

In this section we first discuss a deflation preconditioner for Krylov subspace methods that can be regarded as eliminating the effect of several (given) vectors from the operator or, equivalently, augmenting by these vectors the space where we search for an approximation. Then we describe a combination of the deflation preconditioner with another preconditioner that is commonly used in practice.

The Krylov subspace methods (in particular CG [68] and GMRES [99]) are well-known for their minimization (optimal) properties over the consecutively build *Krylov subspace*

$$\mathcal{K}_j(A, v) = \text{span}\{v, Av, A^2v, \ldots, A^{j-1}v\}.$$

A question arises; given some other subspace $\mathcal{U}$, can we modify the methods such that they have the same optimal properties over the union of $\mathcal{K}_j(A, v)$ and $\mathcal{U}$, which is often called an *augmented* Krylov subspace? The answer is positive and the implementation differs on the method — it is straightforward for GMRES and it requires more attention for CG. Hereafter, we denote by I the identity matrix and by Z the basis of $\mathcal{U}$.

The deflation in GMRES method is often (see, *e.g.*., GMRES-DR [83]) considered as a remedy to overcome the difficulties caused by restarts: for computational and memory restrictions, only a fixed number of GMRES iterations is typically performed giving an approximation that is then used as the initial vector for a new GMRES run. The GMRES method with deflation has been used for solving a sequence of linear algebraic systems, *e.g.*., in [93].

The augmentation of the Krylov subspace in CG is more delicate, since the original CG method can only be applied to an SPD matrix. The first such algorithm was proposed in [45] and [85]. We note that it includes the construction of the conjugate projector

$$P_{c.proj.} = Z(Z^\top \mathsf{A} Z)^{-1} Z^\top \mathsf{A}$$

and, in each iteration, the computation of the preconditioned search direction $q_i = (I - P_{c.proj.})\, p_i$ and of the vector $\mathsf{A}q_i$. The latter can be avoided at the price of storing Z and $\mathsf{A}Z$ and performing additional multiplication with $\mathsf{A}Z$. In both variants, the cost of a single iteration is significantly higher than the cost of one standard CG iteration.

The combination of a preconditioner with a deflation is widely studied in the literature and therefore we present this only briefly; more details and extensive list of references can be found, *e.g.*., in the review paper [115]. The preconditioner stemming from the combination of a (typically relatively simple) traditional preconditioner with the deflation is called a *two–level preconditioner*[3]. While the traditional preconditioner aims at removing the effect of the largest (in magnitude) eigenvalues, the deflation (projection-type preconditioner) is intended to get rid of the effect of the smallest eigenvalues. Common choices for the traditional preconditioner are block Jacobi, (restricted) additive Schwarz method, and incomplete LU or Cholesky factorizations. Among many applications, two–level preconditioners proved to be efficient in the CMB data analysis; see, *e.g.*., [60, 113].

We now present the combination of the traditional and projection-type (deflation) preconditioners following the discussion and notation of [115]. Hereafter, we assume that the system matrix A and the traditional preconditioner M are SPD. We note that some of the below mentioned preconditioners $\mathcal{P}_\square$ are not symmetric. However, their properties allow us to use them (with possible modification of the initial vector) as left

[3] As shown in [115], one can see here an analogy with multilevel (multigrid) and domain decomposition methods.

preconditioners in PCG; see [115] for details.

Let the deflation space span the columns of the matrix Z. We denote

$$P \equiv I - AQ, \qquad Q \equiv Z\left(Z^\top AZ\right)^{-1} Z^\top. \tag{4.15}$$

Two-level preconditioners based on deflation are given as

$$\mathcal{P}_{\mathrm{DEF1}} \equiv M^{-1}P, \qquad \mathcal{P}_{\mathrm{DEF2}} \equiv P^\top M^{-1}.$$

Other preconditioners can be determined using the *additive* combination of two (SPD) preconditioners C_1, C_2 as

$$\mathcal{P}_{\mathrm{add}} \equiv C_1 + C_2,$$

or, using the *multiplicative* combination of the preconditioners, as

$$\mathcal{P}_{\mathrm{mult}} \equiv C_1 + C_2 - C_2AC_1.$$

Three variants of two–level preconditioners are derived by choosing additive or multiplicative combination and setting $C_1 = M^{-1}$, $C_2 = Q$, or $C_1 = Q$, $C_2 = M^{-1}$. Other preconditioners can be derived using the multiplicative combination of three SPD matrices; see [115, Section 2.3.4].

The variants of two–level preconditioner mentioned above differ in the implementation cost and also in the numerical stability; see [115]. The variant $\mathcal{P}_{\mathrm{DEF1}}$, which is often used in the procedures for solving the sequences of linear systems (see, *e.g.*., [100]), was found cheap but less robust, especially with respect to the accuracy of solving the coarse problem with the matrix $Z^\top AZ$ and with respect to the requested accuracy. The conclusion drawn in [115] is that "A-DEF2 seems to be the best and most robust method, considering the theory, numerical experiments and the computational cost". Therefore the preconditioner $\mathcal{P}_{\text{A-DEF2}}$,

$$\mathcal{P}_{\text{A-DEF2}} \equiv M^{-1} + Q - QAM^{-1} = P^\top M^{-1} + Q, \tag{4.16}$$

is of interest, in particular in the cases where the dimension of the deflation space (equal to the number of columns in Z) is high and/or the matrix $M^{-1}A$ is ill-conditioned,

As noted in [100], the gradual loss of orthogonality of computed residuals with respect to the columns of Z can cause stagnation, divergence or erratic behavior of errors within the iterations; see also the comment in [115, Section 4.7]. The remedy seems to be the reorthogonalization of computed residuals as

$$r_j \equiv W r_j, \qquad W \equiv I - Z(Z^\top Z)^{-1}Z^\top. \tag{4.17}$$

For completeness, we recall the general implementation of these two–level PCG methods presented in [48] (Algorithm 18), as well as the corresponding choices of the parameters for several methods (Table 4.1).

Algorithm 18 General two– PCG [115]

Input: $\mathcal{V}_s$, $\mathcal{M}_1$, $\mathcal{M}_2$, $\mathcal{M}_3$, $\mathcal{V}_e$

$x_0 = \mathcal{V}_s$
$r_0 = \mathcal{M}_3(b - Ax_0)$
$z_0 = \mathcal{M}_1 r_0$
$p_1 = \mathcal{M}_2 z_0$
for $k = 1, \ldots,$ convergence **do**
  $w_k = \mathcal{M}_3 A p_k$
  $\alpha_k = p_k^\top z_{k-1} / (p_k^\top w_k)$
  $x_k = x_{k-1} + p_k \alpha_k$
  $r_k = r_{k-1} - w_k \alpha_k$
  $z_k = \mathcal{M}_1 r_k$
  $\beta_k = r_k^\top z_k / r_{k-1}^\top z_{k-1}$
  $p_{k+1} = \mathcal{M}_2 z_k - p_k \beta_k$
end for
$x_k = \mathcal{V}_e$

Method	$\mathcal{V}_s$	$\mathcal{M}_1$	$\mathcal{M}_2$	$\mathcal{M}_3$	$\mathcal{V}_e$
DEF1	x_0	M^{-1}	I	P	$Qb + P^\top x_k$
DEF2	$Qb + P^\top x_0$	M^{-1}	$P^\top$	I	x_k
A-DEF1	x_0	$M^{-1}P + Q$	I	I	x_k
A-DEF2	$Qb + P^\top x_0$	$P^\top M^{-1} + Q$	I	I	x_k

Table 4.1 – Possible choices of the parameters $\mathcal{V}_s$, $\mathcal{M}_1$, $\mathcal{M}_2$, $\mathcal{M}_3$, $\mathcal{V}_e$ in Algorithm 18 and the corresponding two–level method.

4.3 Unified framework for the solution procedures

There already exist several variants for solving a sequence of linear systems such as [75, 77, 88, 93, 100]. In this section, we propose a unified framework that generalizes these methods.

Let us recall that we are interested in solving the following sequence of linear systems,

$$A^{(s)} x^{(s)} = b^{(s)}, \qquad s = 1, 2, \ldots, \nu,$$

where each $A^{(s)}$ is supposed to be symmetric positive definite. Furthermore, we denote $Z^{(s)}$ a matrix whose columns define a deflation space for $A^{(s)}$ (see Section 4.2.2). The core problem of recycling methods is the construction of $Z^{(s)}$. More precisely, the methods rely on three steps:

1. Given $A^{(s)}$, adapt *a priori* $Z^{(s-1)}$ for constructing the new deflation vectors $Z^{(s)}$.

2. Solve $A^{(s)}x^{(s)} = b^{(s)}$ with a two–level preconditioner involving $Z^{(s)}$.
3. Update *a posteriori* $Z^{(s)}$ using the (spectral) information about $A^{(s)}$ obtained during the step 2.

In particular, the steps 1 and 3 are critical because this is where the deflation space is defined: a trade-off has to be made between its computational cost and its efficiency during the solve (step 2).

4.3.1 A priori adaptation of the previous deflation space

Given $A^{(s)}$ and $Z^{(s-1)}$, we construct in this step the new deflation vectors $Z^{(s)}$ that are then used in the solution of $A^{(s)}x^{(s)} = b^{(s)}$. The aims of the construction could be to

1. improve the deflation space,
2. improve the numerical stability of the two–level preconditioner,
3. reduce the dimension of the deflation space if some of the vectors can be omitted.

Ideally, we would like to construct $Z^{(s)}$ such that,

$$A^{(s)}Z^{(s)} = Z^{(s)}\Lambda,$$

where Λ is a diagonal matrix containing the smallest eigenvalues of $A^{(s)}$, *i.e.*, the columns of $Z^{(s)}$ are the eigenvectors of $A^{(s)}$. However, solving this problem is challenging and possibly very costly; see, *e.g.*., [98].

The simplest idea is to set $Z^{(s)} = Z^{(s-1)}$. This is considered in [100], where it is assumed that $\mathsf{A}^{(s)} = \mathsf{A}$, $\forall s$, and therefore $\mathsf{A}^{(s)}Z^{(s)} = \mathsf{A}^{(s-1)}Z^{(s-1)}$. As explained in Section 4.2.2, the two–level preconditioner requires factorizing/inverting ${Z^{(s)}}^{\top}\mathsf{A}^{(s)}Z^{(s)}$. As a consequence, even in the minimal case it is mandatory, for $\mathsf{A}^{(s)} \neq \mathsf{A}^{(s-1)}$, to compute $\mathsf{A}^{(s)}Z^{(s)}$, then a reduction operation that involves a global communications between the processors in the distributed computing framework, and eventually the Cholesky factorization of a small dense matrix ${Z^{(s)}}^{\top}\mathsf{A}^{(s)}Z^{(s)}$.

In [93], the authors propose to define $Z^{(s)}$ as $Z^{(s)} \equiv Z^{(s-1)}R^{-1}$, where $A^{(s)}Z^{(s-1)} = Z_{qr}^{(s)}R$ represents a (thin) QR factorization of $A^{(s)}Z^{(s-1)}$. We note that this procedure is not significantly more costly than setting simply $Z^{(s)} \equiv Z^{(s-1)}$. First, one computes $A^{(s)}Z^{(s-1)}$, and then the QR factorization of a tall and skinny matrix that can be computed using CholQR algorithm, for instance. The new deflation vectors satisfy $A^{(s)}Z^{(s)} = Z_{qr}^{(s)}$. This is preferable in the context used in [93], where the original operator $A^{(s)}$ is replaced by $\left(I - Z_{qr}^{(s)}{Z_{qr}^{(s)}}^{\top}\right)A^{(s)}$. We note that $\mathcal{P} = I - Z_{qr}^{(s)}{Z_{qr}^{(s)}}^{\top}$ is nothing but the orthogonal projector along the range of $Z_{qr}^{(s)}$. This is particularly interesting when dealing with minimum residual Krylov methods, such as GMRES or MINRES, where the Krylov basis V_j is build to be orthogonal and the augmented Krylov basis is then $\mathcal{P}V_j$. In case of the Conjugate Gradient, this is not the case anymore because the Krylov basis is built to

be $A^{(s)}$-orthogonal. For instance, there is no particular simplification in the expression of the deflation preconditioner (4.16): ${Z_{qr}^{(s)}}^{\top} A^{(s)} Z_{qr}^{(s)}$ has still to be formed and factorized.

4.3.2 Solving the system

There exists a plethora of methods and deflation variants (see the discussion in Section 4.2.2) for solving a single system in (4.6). Here, one should take into account a possible inaccuracy of the approximations to eigenvectors that define the deflation space. As a consequence, numerical stability issues can occur during the iterations. This includes the loss of orthogonality and the corresponding delay of convergence (especially if a solver based on short recurrences such as CG or MINRES is used) or reaching a maximal attainable accuracy; for an illustration in CMB application see [94, Section 7]. A remedy might be the full-reorthogonalization of the computed residual against the residuals from the previous steps and against the basis $Z^{(s)}$ of the deflation space (as in (4.17)). However, such reorthogonalization significantly increases the computational and memory requirements and may not be possible in many situations.

4.3.3 A posteriori update of the deflation space

As presented in Section 4.2.1, there is a close connection between Ritz values and harmonic Ritz values approximations, and Krylov iterative solvers because both rely on Arnoldi or Lanczos algorithms. In [100] *Saad et al.* propose a so-called deflated version of the Conjugate Gradient in which approximate harmonic Ritz values and eigenvectors of the matrix are computed at the end of the solve. As for Krylov methods, the loss of orthogonality of the computed basis may have a huge impact on the quality of the approximate eigenvectors.

4.3.4 Existing methods

We now review two existing methods: the Deflated CG method [100], and the GCRO-DR method [93]. In particular, we show how they fit into our framework.

Deflated CG[100]

In [100] *Saad et al.* propose a deflated version of the Conjugate Gradient for solving one linear system with several right-hand sides each given at a time.

We first instantiate the method into our framework:

1. *A priori adaptation:* none, the matrices are assumed to be the same,
2. *Solve:* CG with augmentation of the search space with the previous deflation space,
3. *A posteriori adaptation:* **harmonic** eigenvalue problem including the previous Z.

We now detail how the eigenvectors are approximated. Let $Z = [W, P_l]$ where W denotes the previous eigenvector estimates and P_l denotes the first l search directions of CG. The authors propose to solve the following generalized eigenvalue problem

$$Gu = \lambda Fu, \tag{4.18}$$

where $F = Z^\top AZ$ and $G = Z^\top AAZ$. Then the corresponding eigenvector v of A is given by,

$$v = Zu. \tag{4.19}$$

The eigenvectors associated to eigenvalues below a given threshold are then deflated. We want to point out that the authors use the so–called *augmentation* approach but it is equivalent to using a projection approach [52]. The first l directions are kept in order to limit the memory overhead. This parameter is important because it somehow dictate the accuracy of the approximated eigenvectors. In [100, Part 5.1], it also is explained in details how to compute F and G with the coefficients computed within CG iterations, *i.e.*, avoiding an explicit computation of AZ. We do not recall here the complete algebra as it is not very complicated, but rather long and technical.

Remark 4.3.1

When considering left preconditioning, instead of solving (4.18) we need to solve,

$$Z^\top AM^{-1}AZu = \lambda Z^\top AZu. \tag{4.20}$$

The derivation is done by replacing A by $M^{-1}A$ and the euclidean dot product by the M-inner product in the algorithm as suggested in [101]. This approach is a bit different to what is done in [100] where a split preconditioner is used in order to derive the preconditioned algorithms, but both approaches lead to the same formula.

GCRO-DR [93]

In [93] *de Stuler et al.* propose a hybrid method in between GMRES-DR of *Morgan* [83] and GCROT of *de Sturler* [111]:

- as GMRES-DR it relies on computing an approximate invariant subspace associated to the smallest eigenvalues before restarting,
- as GCROT it uses a deflated restarting that does not rely on any assumption on the subspace to be deflated,

Although the method has been designed for non-symmetric linear systems and it relies on GMRES iterations, in [77] *de Stuler et al.* adapt GCRO-DR to the SPD case by replacing GMRES by MINRES.

As for the previous method, this one also fits into our framework:

1. *A priori adaptation:* it is done through the following process:

$$C = AZ \tag{4.21}$$
$$[Q, R] = qr(C, 0) \tag{4.22}$$
$$Z = ZR^{-1} \tag{4.23}$$

2. *Solve:* a fixed number of steps of GMRES, or MINRES if the matrix is SPD [77, 88], on the deflated operator $(I - ZZ^\top)A$,

3. *A posteriori adaptation:* **harmonic** eigenvalue problem including the previous Z.

GCRO-DR is "algebraically equivalent" to GMRES-DR [83]: for a given x_0 and A (and no coarse space) they both produce the same iterates. The a priori adaptation of Z (step 1) is inspired by GCROT method [111] that was designed to minimize the orthogonality error when restarting GMRES — it was called an "optimal truncation".

In [88], the authors precompute 10 eigenvectors of a "representative" matrix and then adapt this space during the solves. However the authors consider a very specific application. They need to construct a basis of the space spanned by the solutions of the systems so they know that some solves are not useful. The coarse space is then enriched with the approximate solutions and not eigenvectors.

4.4 Numerical experiments

In order to assess the possible benefits of using recycling ideas in the context of CMB analysis we perform several numerical experiments. The framework we have presented in Section 4.3 being very general, the aim of this section is to evaluate the influence of the parameters on the numerical results. However, we restrict ourselves to the Projected Preconditioned CG (PPCG) method (Algorithm 18) as solver (step 2) because the matrices are all SPD, and it is very likely that a MINRES–like algorithm would behave essentially the same as the PPCG method. This problem is solved for 5 different frequencies at the same time. We used the same data as that used in [92], *i.e.*, its size is roughly 10^8 ($N_{\text{data}} \approx 10^8$), and the number of pixels is $\mathcal{N} = 256 \times 256 = 65,536$. The experiments were performed using a sequential code written in python which relies on `numpy` and `scipy` packages. The tolerance for the stopping criterion is set to 10^{-8}.

4.4.1 A simplified case

We consider a first simplified situation where the matrix associated to the system and the right-hand side are kept constant but several initial guesses are used to generate an artificial sequence of linear systems' solves. Although this simplification makes the solve quite artificial, from a numerical point of view it is very similar to the experiments performed in [100] because in both cases the initial residuals are random perturbations of each other. The initial guesses are chosen to be uniformly random, and constructed using `numpy.random`.

In Figure 4.1, we summarize the results we obtained using two configurations for 3 initial guesses. In both cases, we used the harmonic Ritz approximations in order to compute approximations of the eigenvectors, and these are then deflated using DEF1 variant (whereas DEF2 is used in [100]). The first configuration, denoted "partial", mimics that used in [100]: the first 20 search directions are kept and 10 approximated eigenvectors associated to the smallest eigenvalues are computed using these search directions. In the second configuration, denoted "full", the whole basis is kept and 20 eigenvectors associated to the smallest eigenvalues are then computed using these search directions. For the "partial" configuration, we observe that the number of iterations is slightly decreasing from one initial guess to another. This is in accordance with [100] where the authors observe that the quality of the coarse space is dynamically improved through the sequence. For the "full" configuration, we observe that the number of iterations is significantly decreasing from the first initial guess to the second one, by a factor of $\frac{122}{34} \approx 4$. Then, the third initial guess is converging as the second one. This means that a good coarse space is already computed after the first solve if the whole basis is kept. In our case, the memory and computational overhead of the "full" configuration is affordable because the dimension of the unknown s is much smaller than that of the data d. Thus in practice the most time consuming task is usually the computation of $\mathsf{N}^{-1}v$ that occurs at each iterations of the CG method and one can reasonably expect that the computational overhead will be more than compensated by the significant decrease of the number of iterations. Similarly, the memory overhead is negligible compared to the storage of the data. Hence, the "full" configuration is particularly interesting in our case.

We then consider a second simplified situation where the matrix associated to the system is kept constant but the right-hand sides are changing. In this case, it is the same experiments as those performed in [100]. Compared to the previous case, we are expecting that the random perturbations between two following initial residuals is larger. Thus, we expect to observe more or less the same numerical behavior as in the previous case, but the deflation should be less effective than before. The right-hand sides are chosen to be uniformly random, and constructed using `numpy.random`.

In Figure 4.2, we summarize the results we obtained using two configurations for 3 right-hand sides. In both cases, we used the harmonic Ritz approximations in order to compute approximations of the eigenvectors, and these are then deflated using A-DEF1 variant. The "partial" configuration corresponds to that used earlier: the first 20 search directions are kept and 10 approximated eigenvectors associated to the smallest eigenvalues are computed using these search directions and the previous deflated vectors. And the "full" configuration also corresponds to that used in the previous experiment: the whole basis is kept and 20 eigenvectors associated to the smallest eigenvalues are then computed using these search directions and the previous deflated vectors. For the "partial" configuration, we observe that the number of iterations is not decreasing anymore from one right-hand side to another. In this case, the approximation of the eigenvectors is not good enough to observe a proper deflation of these. For the "full" configuration, we observe that the number of iterations is decreasing from the

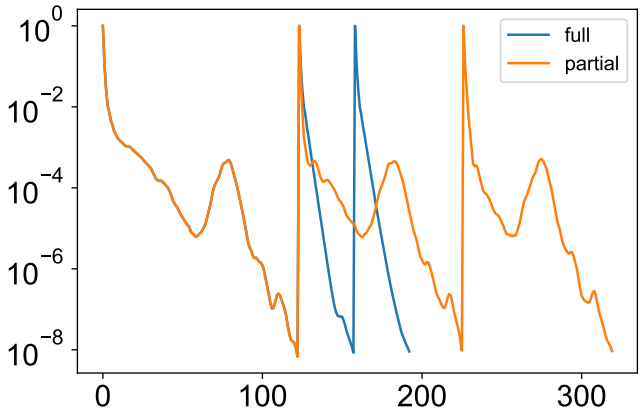


Figure 4.1 – The normalized residual as a function of the number of iterations during the sequence for the first simplified case experiment (the initial guess is changing through the sequence) with 2 configurations. "Full" means that all the search directions are kept and 20 vectors are deflated (blue), and "partial" means that 20 search directions are kept and 10 vectors are deflated (orange).

first right-hand side to the second one, by a factor of $\frac{108}{75} \approx 1.5$. We note that this decrease is less important than in the previous case, but it is still significant. This is in accordance with our observation for the "partial" configuration, and it illustrates the fact that in this case the random perturbation between the initial residuals is larger than in the previous case. Then, the third right-hand side is converging in almost the number of iteration as the second one (82). This means that keeping the whole search directions greatly improves the quality of the coarse space. Once again, this illustrates the potential benefits of the "full" configuration in this case.

For the second experiment, we notice that the convergence for the "full" and "partial" configurations are not exactly the same for the first right-hand side, whereas it was exactly the same for the first initial guess in the experiment. The reason is that the computations were ran on a supercomputer and thus the random numbers generated are not the same for the "full" and "partial" configurations for a given experiment. Although this is not visible for the first experiment, it becomes to be for the second experiment. We do not think this bias is important for the conclusion because the difference is still very small in the second experiment and both configurations are converging in the same number of iterations for the first right-hand side. However, this is another illustration of the fact that in the second experiment the initial residuals are less close to each other than in the first experiment.

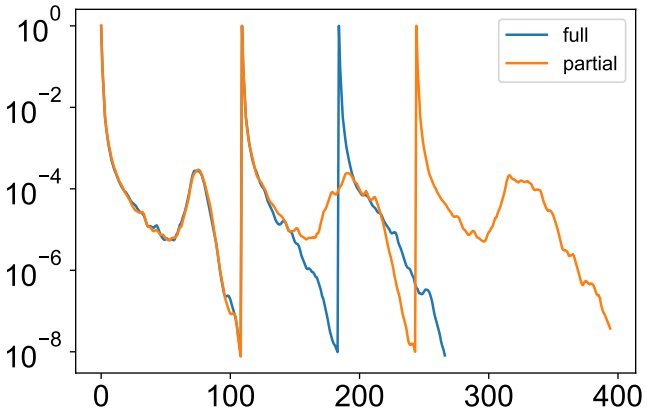


Figure 4.2 – The normalized residual as a function of the number of iterations during the sequence for the second simplified case experiment (the right-hand side is changing through the sequence) with 2 configurations. "Full" means that all the search directions are kept and 20 vectors are deflated (blue), and "partial" means that 20 search directions are kept and 10 vectors are deflated (orange).

4.4.2 Systems arising from the PCS problem

In what follows we consider a sequence coming from a parametric component separation (PCS) problem. The full sequence contains 26 linear systems with their corresponding right-hand sides to be solved.

First of all, we notice that when using a zero initial guess ($x_0 = 0$), all the systems converge in exactly the same number of iterations (Figure 4.3). This is rather unusual, but it might suggest that all the matrices have the same spectrum — although the convergence of the CG method in floating–point arithmetic is a very complex phenomenon [82].

One of the most natural method is the so-called continuation technique which consists in using the previous solution as an initial guess for the next system; it is both very simple and inexpensive. In Figure 4.4 we have plot the normalized residual norm through the sequence when using the continuation technique. The picks correspond to an increase of the residual because a new system has to be solved. However, there are consecutive systems whose solutions are very close. For instance, the first system needs 125 iterations to be solved, but the second one converges in only 2 iterations! Similarly, the fourth system converges in 1 iteration whereas the third one needs 122 to converge. We observe that the number of iterations tends to decrease through the sequence. This is a consequence of the underlying optimization process that is converging, thus the difference between two consecutive solutions is decreasing. From these observations, we decide to focus on the first four systems because they are representative of the overall behavior of the sequence. More precisely, we aim at improving the convergence of

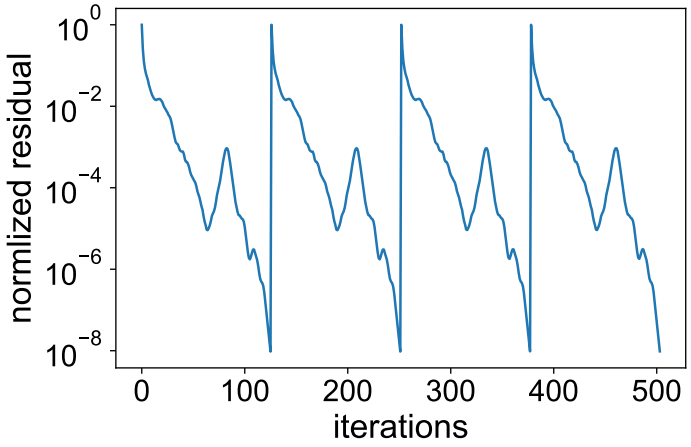


Figure 4.3 – Convergence results for the first 4 systems of the full sequence with $x_0 = 0$. The tolerance is set to 10^{-8}. We do not plot the 22 other systems but their convergence is exactly similar.

the third system where the continuation technique does not really improve the convergence.

The framework we have presented is quite general, leaving several parameters to be set according to the particular problem one is interested in. Namely, the two–level preconditioner can be built in different ways [115], the number of deflated vectors is free, and the method to approximate the eigenvectors associated to the smallest eigenvalues (Ritz or harmonic Ritz approximation) has to be decided. In what follows, we evaluate the influence of each parameter in our case of interest: the PCS problem.

Comparison between different types of two–level preconditioners

As pointed out in [115], the way of deflating the coarse space, *i.e.*, the construction of the two–level preconditioner, may have an impact on the convergence of the iterative method even if in theory these are all equivalent. In the first experiment, we compare DEF1, DEF2, and A-DEF1 types of two–level preconditioners in order to assess their respective stability. We fix the number of deflated vectors to 20. These vectors correspond to harmonic Ritz approximations of the eigenvectors associated to the smallest eigenvalues of the previous matrix[4] using all the search directions, *i.e.*, the "full" configuration. In Figure 4.5, we plot the residual convergence for the first four systems of the sequence for the DEF1, DEF2, and A-DEF1 types of two–level preconditioners; the residual convergence using the continuation technique is plot as a reference. We first observe that the 3 types of two–level preconditioners have the same convergence behavior. This is not really surprising because all these preconditioners are in some sense equivalent, and they usually behave similarly numerically [115]. However, we

[4]The first system is solved without deflation.

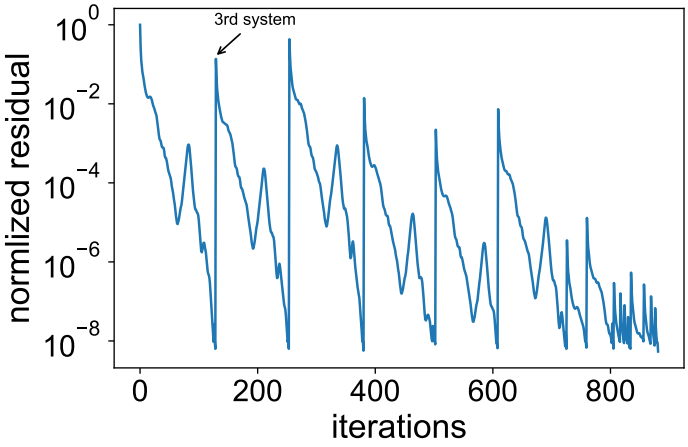


Figure 4.4 – Convergence results of the full sequence of 26 systems solved using continuation. The tolerance is set to 10^{-8}.

also observe that the convergence of the continuation technique is slightly better than that of the deflated versions. This is rather surprising because we would expect that the number of iterations decreases when using deflation. Nevertheless, we can conclude that this behavior is not coming from the two–level preconditioner because the 3 variants we have tested are converging similarly. Hence, we now study in more details the deflated subspace.

Influence of the number of deflated vectors

First, we study the influence of the number of deflated vectors. More precisely, we vary the number of approximated eigenvectors in the harmonic Ritz projection problem using all the search directions and the previous deflated vectors. These are then deflated using the DEF1 method. The results are plot in Figure 4.6. We observe that deflating 10 or 20 eigenvectors has the same effect on the convergence; it is very slightly worse that the continuation. On the other hand, the convergence of the method when 5 vectors are deflated is exactly the same as for the continuation. These results further suggest that the deflated subspace is not really appropriate to our test case. Thus increasing its size does not help, and it even increases the numerical round–off errors — although the overall effect of adding more deflated vectors is not significant.

Comparison between Ritz and harmonic Ritz approximations of the eigenvectors

Second, we compare the Ritz and harmonic Ritz approximations of the eigenvectors. In more details, we compute 20 vectors that approximate the eigenvectors associated the smallest eigenvalues of the underlying matrix using either Ritz or harmonic Ritz approximation. In both cases, these problems are solved using all the search directions

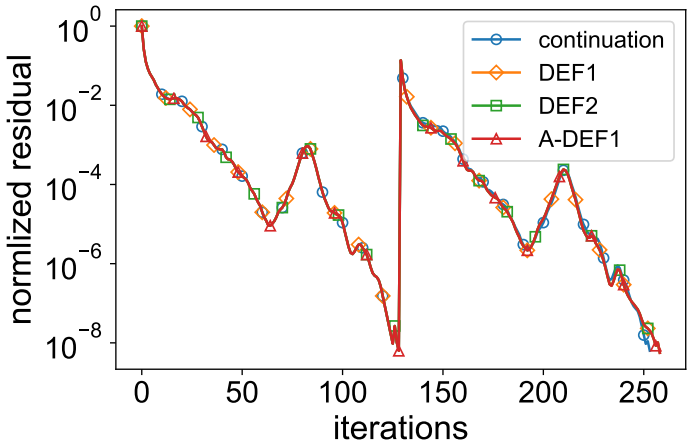


Figure 4.5 – Convergence results for different types of two–level preconditioners. The first four systems are solved using the following parameters: the tolerance is set to 10^{-8} and 20 eigenvectors are approximated using harmonic Ritz projection technique.

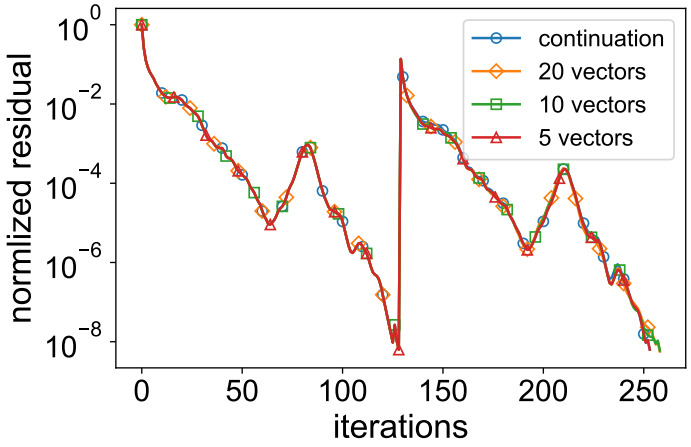


Figure 4.6 – Convergence results for different numbers of deflated vectors. The first four systems are solved using the following parameters: the tolerance is set to 10^{-8}, the eigenvectors are approximated using harmonic Ritz projection technique, and they are deflated using the DEF1 preconditioner.

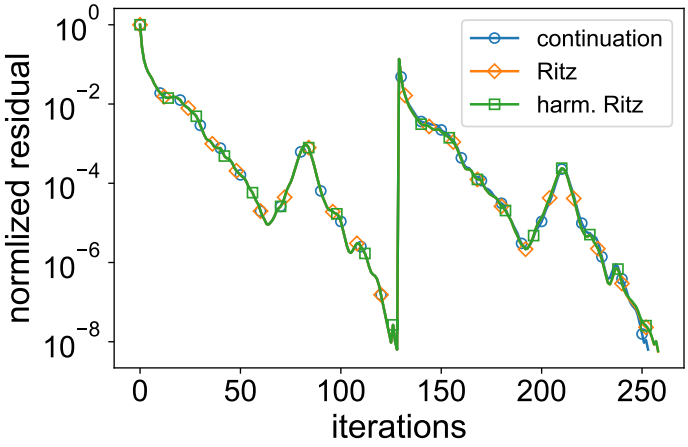


Figure 4.7 – Convergence results for Ritz and harmonic Ritz approximations of the eigenvectors. The first four systems are solved using the following parameters: the tolerance is set to 10^{-8}, 20 eigenvectors are approximated, and they are deflated using the DEF1 preconditioner.

and the previous deflated vectors. These approximations are then deflated using DEF1 algorithm. In Figure 4.7, we summarize the results obtained. We observe that both methods converge very similarly, thus using the Ritz projection method does not help to find a better subspace to deflate. In fact, this is not really surprising because it is usually claimed that the harmonic Ritz "approach yield[ed]s the best results in finding eigenvalues nearest zero" [100, p. 1918]. Once again, the conclusion reached is that the deflated subspace is not adapted to our case of interest.

Accurate computation of the eigenvectors associated to the smallest eigenvalues

Given the previous experiments, we suspect that the eigenvectors associated to the smallest eigenvalues of some matrices in the sequence, *e.g.*, the first one, are different than that of other matrices, *e.g.*, the third one, although all the matrices have the exact same convergence behavior (see Figure 4.3). In order to have a better understanding of the spectral properties of the underlying matrices, we use the built–in function of the `numpy.sparse.linalg` package[5] to compute the 20 smallest eigenvalues and their associated eigenvectors of the preconditioned operator, *i.e.*, the matrix $(\mathsf{M}_\beta^\top \mathcal{G} M_\beta)^{-1} M_\beta^\top \mathcal{F} M_\beta$ with the following notations:

$$\mathcal{F} \equiv P^\top \mathsf{N}^{-1} P, \qquad \mathcal{G} \equiv P^\top \mathrm{diag}(\mathsf{N}^{-1}) P. \tag{4.24}$$

We plot the 20 smallest eigenvalues obtained in Figure 4.8. We observe that the smallest eigenvalue of the preconditioned operator is around $\mu_1 \approx 8 \times 10^{-7}$, and it has

[5] It is in fact a binding to the ARPACK package (https://www.caam.rice.edu/software/ARPACK/).

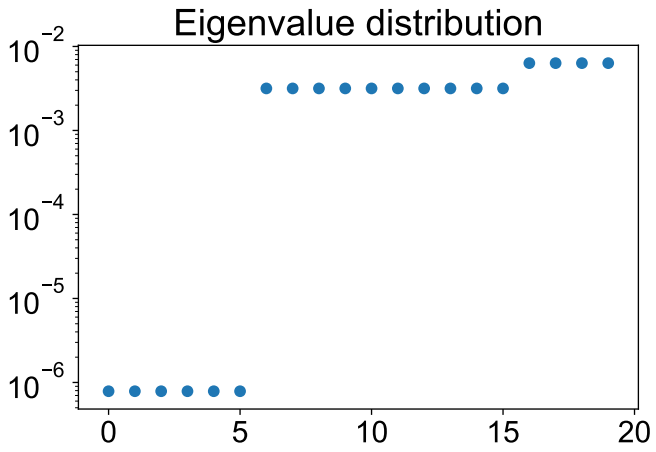


Figure 4.8 – Eigenvalues' distribution of the preconditioned operator for the first system computed with `scipy.sparse.linalg.eigsh`. The tolerance is set to 10^{-10}, and the maximum number of iterations is set to 100.

a multiplicity of 6. However, it is well–known that the Lanczos method, from which the Ritz and harmonic Ritz projections are deduced, does not allow to approximate multiple eigenvalues [98]. This is because the associated tridiagonal matrix from which the approximations are computed is unreduced, thus its eigenvalues are all different. That is why in practice even if we have approximated one eigenvector associated to μ_1 and then deflated it, there are still 5 eigenvectors associated to this eigenvalue and the corresponding eigenspace that may in fact not be discarded. This is, in general, not a problem when considering full orthogonalization methods such as GCRO-DR because at each restart one usually deflates the converged eigenvectors. In this case, it would mean that after 6 restarts one may expect to have completely deflated the eigenspace associated to the smallest eigenvalues. This is of course not the case for us, and this may explain why the convergence of the third system is not improved with our deflated vectors.

In order to confirm our hypothesis, we now solve the third system of the sequence while deflating the 20 approximate eigenvectors computed using the `scipy` built–in function. As the second system is converging in 2 iterations, it mimics a situation where we would have a good approximation of the eigenspace associated to the smallest eigenvalue at the end of the first solve. We compare the convergence with that obtained earlier with Ritz and harmonic Ritz approximations. In both cases we use the DEF1 algorithm. The corresponding results are plot in Figure 4.9. We observe that the deflated vectors computed with the `eigsh` of `scipy` are of higher quality because the overall number of iterations is around 40% lower when they are used in the two–level preconditioner. More precisely, we observe that the convergence is "smooth" and the residual is monotonically decreasing with the accurate approximation of the eigenspace associated to the smallest eigenvalues, whereas there is a "pick" with the

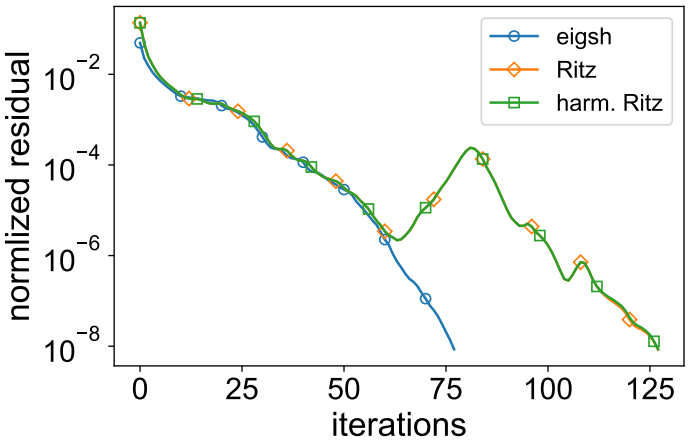


Figure 4.9 – Convergence of the normalized residual of the third system for different deflated vectors. In all cases, the number of deflated vectors is 20, and these are deflated using DEF1 preconditioner. The label `eigsh` means that the eigenvectors are computed using the `scipy` built–in function: the tolerance is set to 10^{-10}, and the maximum number of iterations is set to 100. The labels Ritz, respectively harmonic Ritz, mean that the eigenvectors are computed using Ritz, respectively harmonic Ritz, approximations where all the previous search directions are kept.

Ritz and harmonic Ritz approximations which delays the convergence.

Previously, we have claimed that "after 6 restarts one may expect to have completely deflated the eigenspace associated to the smallest eigenvalues". In order to support this statement, we now simulate this situation numerically. More precisely, we deflate 5 eigenvectors associated to the smallest eigenvalues, which were computed with `eigsh`, to solve the second system. Then we compute 6 eigenvectors associated to the smallest eigenvalues of the second matrix using harmonic Ritz projection with all the search directions, and the previous deflated vectors. We eventually deflated these 6 vectors to solve the third system. We plot the convergence for both systems in Figure 4.10. We observe that the second system is converging in 125 iterations whereas the third is converging in 78 iterations. This supports our claim: we were able to approximate the 6th vector associated to the smallest eigenvalue at the end of the solve of the second system, and that is why the convergence of the third system is much faster. Thus, in practice if the sequence of linear systems to solve is long enough, it is eventually possible to approximate the whole eigenspace associated to the smallest eigenvalue.

4.4.3 Adaptation of the initial guess for the PCS systems

The previous experiments have shown that the spectral information contained in the Krylov basis of the previous matrices does not accelerate the convergence of the current matrix. This difficulty may be overcome by adding an implicit or explicit deflation

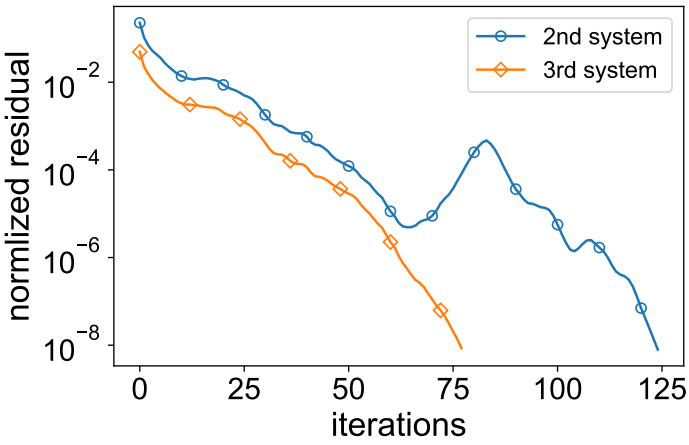


Figure 4.10 – Convergence of the normalized residual of the second and third systems when deflating first 5 accurate eigenvectors, and then 6 eigenvectors. The tolerance is set to 10^{-8}, and the DEF1 technique is used for the deflation.

procedure for the approximation of the eigenvalues. We want to point out that using such procedure would need a careful selection of some additional parameters, *e.g.*, a restart size, a locking tolerance. It would also induce an extra cost, that might be not profitable in all cases. On the other hand, the continuation technique, which is very simple and inexpensive, has shown very good results for some of the linear systems because some solutions are very close to each other. Thus, another idea, that we now put forward, is to take advantage of our knowledge about the underlying problem in order to find an improved initial guess.

More precisely, we now focus on the special structure of the underlying operator in order to find an improved initial guess. We denote $\mathcal{F} \equiv P^\top \mathsf{N}^{-1} P$ and $\widetilde{b} \equiv P^\top \mathsf{N}^{-1} d$. With these notations (4.7) reads,

$$M_\beta^\top \mathcal{F} M_\beta s_\beta = M_\beta^\top \widetilde{b}. \tag{4.25}$$

We notice that $M_\beta s_\beta$ is independent of β. Indeed, we have

$$M_\beta^\top (\mathcal{F} M_\beta s_\beta - \widetilde{b}) = 0 \iff \mathcal{F} M_\beta s_\beta = \widetilde{b}, \tag{4.26}$$

because M_β is full rank. In fact, $M_\beta s_\beta$ can be seen as the solution of an usual map–making problem. It directly follows that given any β and β', we have $M_\beta s_\beta = M_{\beta'} s_{\beta'}$. Let us assume that we already have an (approximated) expression of s_β, then if $M_{\beta'}$ was invertible, one would apply its inverse in order to find the (approximated) expression of $s_{\beta'}$. However as it is not the case, we simply propose to apply its pseudo-inverse instead, denoted $M_{\beta'}^\dagger$, in the hope that $M_{\beta'}^\dagger M_\beta s_\beta \approx s_{\beta'}$. We want to emphasize that this procedure is much cheaper than the application of the operator $M_{\beta'}^\top \mathcal{F} M_{\beta'}$ because: 1) it is independent of $\mathcal{F}$, 2) applying $M_{\beta'}^\dagger = (M_{\beta'}^\top M_{\beta'})^{-1} M_{\beta'}^\top$ is very cheap because $M_{\beta'}$ has

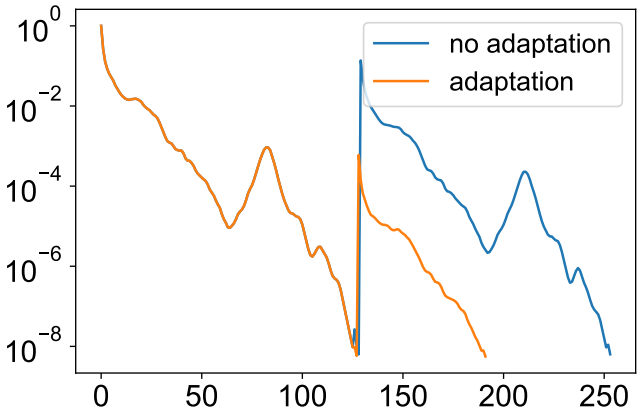


Figure 4.11 – The reduced sequence of 4 systems solved using continuation and adaptation of the initial guess. The tolerance is set to 10^{-8}.

a tensorized structure. Indeed, we have,

$$M_{\beta'}^{\top} M_{\beta'} = K_{\beta'}^{\top} K_{\beta'} \otimes I, \tag{4.27}$$

and,

$$(M_{\beta'}^{\top} M_{\beta'})^{-1} = (K_{\beta'}^{\top} K_{\beta'})^{-1} \otimes I, \tag{4.28}$$

where $K_{\beta'}^{\top} K_{\beta'}$ is a 3×3 matrix.

In Figure 4.11, we show the results we obtained when adapting the initial guess using this formula. As explained before, we focus on the first four systems of the sequence, and we use the continuation's convergence as a reference. We observe that the adaptation of the initial guess allows to reduce by a factor of around 2 the number of iterations for the third system. Thus, the overall number of iterations for solving the 4 linear systems is reduced by a factor of around 1.5. We want to emphasize that the adaptation of the initial guess is very cheap because of the tensorized structure of M_{β}, and it is also very simple to implement. Thus it should lead to a significant speed–up in realistic computations.

Conclusion

Outline of the current chapter

Summary

In order to effectively use the most powerful supercomputers — and prepare for the forthcoming generation of exascale machines — it is crucial to redesign from the ground up the linear algebra routines that constitute the cornerstone of many scientific software. Throughout this thesis, we thoroughly studied a reformulation of the Conjugate Gradient method that relies on so–called Enlarged Krylov subspaces: the Enlarged Conjugate Gradient method; both from a theoretical and practical points of view.

We propose a new derivation of the Enlarged Conjugate Gradient that allows us to get two variants of the method (Orthodir and Orthomin). We have described the link between these two variants, and thus explained their difference in terms of robustness. We also have studied its convergence rate and we have showed that the Enlarged Conjugate Gradient method is acting as if the smallest eigenvalues of the matrix were somehow deflated. Numerical experiments show that enlarging the Krylov subspaces allow to reduce significantly the number of iterations with respect to the classical PCG method. In fact, we show that the method is indeed acting numerically as a two–level preconditioner. Moreover, we have presented dynamic versions of Orthodir and Orthomin where the number of search directions is adaptively reduced during the iterations. These dynamic versions allow to reduce the cost of the extra arithmetic operations induced by the method without altering its efficiency.

Then, we have described the parallel design of the method, including the two variants as well as their dynamic versions. In order to further increase the scalability of this design we have proposed a reformulation of the algorithm in order to fuse the different synchronizations within an iteration. Also, as our implementation is based on BLAS3 kernels only, it offers a good scaling when increasing the number of threads per MPI process. Using the Fused variant of Orthodir we are able to scale up to 32,768 threads, each one bound to a physical core, while being almost 4 times faster than PETSc PCG for an elasticity matrix. Throughout this work, the only assumption we make is that

the matrix is symmetric positive definite. Hence the resulting methods are very generic and completely algebraic. For instance, it is straightforward to use D-Odir for solving linear systems with several right-hand sides. Similarly, ECG can be used with any preconditioner that could be used with CG. Thus, it can be used as a black-box solver that can be integrated very easily in any existing code. As it increases the arithmetic intensity and reduces the communication, it is well suited for modern and future architectures that exhibit massive parallelism. According to the theoretical study and the numerical experiments, it is particularly well adapted for matrices with few small eigenvalues.

Finally, we have applied recycling strategies to the parametric component separation problem (PCS) in CMB data analysis. The solution of a sequence of linear systems that are, in some sense, close to each other is of interest for many other applications. We have introduced a general framework that embeds several existing recycling methods based on two main ingredients: the construction of a two–level preconditioner, and the approximation of eigenvalues and the associated eigenvectors from an already computed Krylov basis. When the underlying matrix is fixed, and there are multiple right-hand sides, the recycling techniques allow to reduce the number of iterations significantly. However, as a result of the multiplicity of the smallest eigenvalue, these techniques are not very efficient when both the matrix and the right-hand sides are changing. Hence, we have proposed a cheap procedure to adapt the initial guess that has permitted to reduce the overall number of iterations by a factor of 1.5. This adaptation relies on the tensorized structure of the mixing matrix. Taking into account this special structure allows us to simplify considerably some computations that seemed *a priori* impossible to perform from a practical point of view. Thus in the case of a sequence of linear systems we recommend to use what little information is available about the underlying problem. It is indeed quite complicated for a given case of interest to formalize what "close to each other" exactly means in terms of the eigenvectors of the underlying matrices.

Perspectives

This thesis strived to investigate the benefits of using the Enlarged Conjugate Gradient method for solving linear systems on massively parallel machines. Of course, this work has also raised questions that would require further investigation.

- The way the Krylov subspaces are enlarged is very simple so a natural question is: is it possible to find a better way to enlarge the Krylov subspaces? It seems to be a difficult question because the Theorem 2.4.1 suggests that one should enrich the Krylov subspaces with a "good" approximation of the eigenvectors associated to the smallest eigenvalues of the matrix in order to bound the constant. And it is indeed well-known that these eigenvectors are difficult to compute [98]. Finding a cheap, yet acceptable in terms of accuracy, approximation would certainly require taking advantage of special properties of the underlying problem.

- It is well–known that round–off errors play an important role in the convergence of the Krylov methods in practice, and on particular on the maximum attainable accuracy [56, 82]. To our knowledge, there exists no theoretical study of the behavior of the block CG-like methods in floating–point arithmetic. In particular, it would be very interesting to study Orthodir and Orthomin in terms of maximum attainable accuracy. A first step might be to start from [72] where the authors make this study in the context of the GMRES method, and to extent their work to the (block) CG method. This theoretical work would be the first step in order to derive a *pipeline* ECG, or an *inexact* ECG [102] where the sparse matrix–vector product is computed inexactly.

- Of course, a natural extension of this work is to consider the case where A is not symmetric positive definite. In fact, an Enlarged GMRES (EGMRES) method has been derived by *Al Daas and Grigori* [5]. In order to make mitigate the memory overhead several techniques are employed: deflation of the smallest eigenvalues at restart, dynamic reduction of the search directions. However another direction of research is to derive an Enlarged BiCGSTAB method, generalizing [118]. A starting point is the block BiCGSTAB method [64], from which it should be straightforward to derive an Enlarged version. Nevertheless, a special attention should be taken in order to mitigate the effect of possible breakdowns. Also, the theory is much more complicated in the non–symmetric, and especially in the case of BiCG-like methods, thus extensive numerical experiments should be carried out. Finally, we want to point out that a parallel implementation should be quite straightforward because it would rely on the same kernels as those we used in the implementation of ECG.

- From a more practical point of view, it would be very interesting to test the method on GPUs as it seems that block Krylov methods are particularly effective on these [28]. Similarly, ECG should be well adapted for this special type of architecture.

- A direct follow up perspective would be to implement the method using task–based programming [7, 11] where the program is now split into tasks, and their dependencies. The scheduling of the tasks is then done by the runtime system in a very efficient way. This approach has been very successful in dense linear algebra packages such as PLASMA [4]. Of course the challenge would be to effectively schedule the tasks among the ten of thousands of physical processors while taking into account the hardware hierarchy of a supercomputer.

- As for the parameter component separation problem of CMB analysis, many research directions remain opened. The most natural one would be to study the potential benefits of the ECG method for solving the first system of the sequence. On the one hand, one could expect to converge in less iterations, and thus to reduce the runtime. And on the other hand, it should also be possible to approximate at the same time several eigenvectors associated to the same eigenvalue,

thus to increase tremendously the quality of the deflated subspace. This work has been started by *Thibault Cimic* during his Master's degree internship and the first results show that the ECG method indeed allows to decrease the number of iterations for Map–making problems. However, it remains to see if it also allows to compute a good approximation of the eigenspace associated to the smallest eigenvalues, and to eventually test the method in a massively parallel environment on realistic test cases. This should be one of the goal of *Thibault Cimic*'s Ph.D. that has just started in October.

A

Appendices

Outline of the current chapter

A.1 A convergence study of ECG using [89, Theorem 5]

In this section, we use Theorem 5 from *O'Leary* in [89] to derive a "naive" bound for the rate of convergence of ECG. In what follows we use the notations defined hereinafter. We denote $u^{(i)}$ the i^{th} component of a vector, and $U^{(i)}$ the vector representing the i^{th} column of a matrix. u_k is the vector u at the k^{th} iteration of the iterative process. A is a square symmetric definite positive (s.p.d.) matrix of size n whose eigenvalues are denoted $0 < \lambda_1 \leqslant \cdots \leqslant \lambda_i \leqslant \cdots \leqslant \lambda_n$. $\kappa = \frac{\lambda_n}{\lambda_1}$ is the condition number of A. $\Lambda = \text{diag}(\lambda_1, \ldots, \lambda_n)$ is a diagonal matrix containing the eigenvalues of A. $\phi_1, \ldots, \phi_n$ are the eigenvectors associated to $\lambda_1, \ldots, \lambda_n$ such that they form an orthonormal basis. $\Phi = \begin{pmatrix} \phi_1 & \ldots & \phi_n \end{pmatrix}$ is a matrix containing the eigenvectors of A. $\|u\|_A = u^\top A u$ is the norm induced by A. x_* is the exact solution of the system $Ax = b$, *i.e.* $x_* = A^{-1}b$. t is the block size, it corresponds to the enlarging factor in ECG and the number of right-hand sides in BCG.

We state the main result of this section.

Theorem A.1.1

Let x_k the approximation given by the ECG process at step k then we have:

$$\|x_k - x_*\|_A^2 \leqslant C\left(\frac{\sqrt{\kappa_t}-1}{\sqrt{\kappa_t}+1}\right)^{2k} \tag{A.1}$$

where $\kappa_t = \dfrac{\lambda_n}{\lambda_t}$ and C is a constant independent of k. More precisely we have,

$$\begin{aligned} C \leqslant 4\|\|x_0 - x_*\|_A^2 + 16\|x_0 - x_*\|_2^2\,\lambda_n\left(\frac{\lambda_n+\lambda_t-\lambda_1}{\lambda_n-\lambda_t}\right)^2 \max_{i\in\{1,\dots,t\}} \tan^2(\theta^{(i)}) \\ + 16\|x_0 - x_*\|_2^2\,\lambda_n\left(\frac{\lambda_n+\lambda_t-\lambda_1}{\lambda_n-\lambda_t}\right) \max_{i\in\{1,\dots,t\}} \tan(\theta^{(i)}) \end{aligned} \tag{A.2}$$

Proof. In [89] *O'Leary* demonstrated the following result on the convergence of Block Conjugate Gradient,

$$\|X_k^{(i)} - X_*^{(i)}\|_A^2 \leqslant \left(C_1^{(i)} + C_2^{(i)} + C_3^{(i)}\right)\left(\frac{\sqrt{\kappa_t}-1}{\sqrt{\kappa_t}+1}\right)^{2k} \tag{A.3}$$

with $C_1^{(i)}$, $C_2^{(i)}$ and $C_3^{(i)}$ some constants independent of k but that may depend on i ; and $X_*^{(i)}$ the exact solution of the i^{th} system. This results can be applied to each columns of X_k the solution of ECG because by definition ECG is a special case of BCG. So using (A.3) we have,

$$\|x_k - x_*\|_A^2 \leqslant \left(\sum_{i=1}^{t} C_1^{(i)} + C_2^{(i)} + C_3^{(i)}\right)\left(\frac{\sqrt{\kappa_t}-1}{\sqrt{\kappa_t}+1}\right)^{2k}. \tag{A.4}$$

First, we recall some notations introduced in [89] in order to take a closer look into the constants $C_1^{(i)}$, $C_2^{(i)}$ and $C_3^{(i)}$:

$$\Phi\xi^{(i)} = X_0^{(i)} - X_*^{(i)}, \qquad \xi^{(i)} = \begin{pmatrix} \Xi_1^{(i)} \\ \vdots \\ \Xi_n^{(i)} \end{pmatrix} \in \mathbb{R}^n \tag{A.5}$$

$$\xi_1^{(i)} = \begin{pmatrix} \Xi_1^{(i)} \\ \vdots \\ \Xi_{t-1}^{(i)} \end{pmatrix} \in \mathbb{R}^{t-1}, \qquad \xi_2^{(i)} = \begin{pmatrix} \Xi_t^{(i)} \\ \vdots \\ \Xi_n^{(i)} \end{pmatrix} \in \mathbb{R}^{n-t+1} \tag{A.6}$$

$$\Lambda_1 = \operatorname{diag}(\lambda_1,\dots,\lambda_{t-1}), \qquad \Lambda_2 = \operatorname{diag}(\lambda_t,\dots,\lambda_n) \tag{A.7}$$

In addition, we need to define $F^{(i)} \in \mathbb{R}^{n\times(t-1)}$ such that $\Phi F^{(i)}$ is an orthonormal basis of

$$\operatorname{span}^{\square}\left\{R_0^{(1)},\dots,R_0^{(i-1)},R_0^{(i+1)},\dots,R_0^{(n)}\right\} \tag{A.8}$$

and

$$F^{(i)} = \begin{pmatrix} f_1^{(i)} \\ \vdots \\ f_n^{(i)} \end{pmatrix} \text{with } f_i^{(i)} \text{ a row vector of size } t-1 \tag{A.9}$$

$$F_1^{(i)} = \begin{pmatrix} f_1^{(i)} \\ \vdots \\ f_{t-1}^{(i)} \end{pmatrix} \qquad F_2^{(i)} = \begin{pmatrix} f_t^{(i)} \\ \vdots \\ f_n^{(i)} \end{pmatrix} \tag{A.10}$$

$$\|F_2^{(i)}{F_1^{(i)}}^{-1}\|_2^2 = \tan^2(\theta^{(i)}) \tag{A.11}$$

$$\theta^{(i)} = \arccos\left(\lambda_{\min}\left(F_1^{(i)}\right)\right) \tag{A.12}$$

$C_1^{(i)}$, $C_2^{(i)}$ and $C_3^{(i)}$ are defined in [89] as,

$$C_1 = 4{\xi_2^{(i)}}^t \Lambda_2 \xi_2^{(i)}, \tag{A.13}$$

$$C_2^{(i)} = 16\left(\frac{\lambda_n+\lambda_t-\lambda_1}{\lambda_n-\lambda_t}\right)^2 \tan^2(\theta^{(i)})\|\Lambda_2\|_2\|\xi_1^{(i)}\|_2^2, \tag{A.14}$$

$$C_3^{(3)} = 16\left(\frac{\lambda_n+\lambda_t-\lambda_1}{\lambda_n-\lambda_t}\right)\tan(\theta^{(i)})\|\Lambda_2\xi_2^{(i)}\|_2\|\xi_1^{(i)}\|_2. \tag{A.15}$$

So we need to bound,

$$\sum_{i=1}^{t} C_1^{(i)} + C_2^{(i)} + C_3^{(3)}. \tag{A.16}$$

We split the sum in three according to the different constants involved.

- The first constant is easy to bound because,

$$C_1^{(i)} = \|x_0^{(i)} - x_*^{(i)}\|_A^2, \tag{A.17}$$

  hence,

$$\sum_{i=1}^{t} C_1^{(i)} = \|x_0 - x_*\|_A^2. \tag{A.18}$$

- $\sum_{i=1}^{t} C_2^{(i)}$ can be rewritten as,

$$\sum_{i=1}^{t} C_2^{(i)} = \sum_{i=1}^{t} 16\left(\frac{\lambda_n+\lambda_t-\lambda_1}{\lambda_n-\lambda_t}\right)^2 \tan^2(\theta^{(i)})\|\Lambda_2\|_2\|\xi_1^{(i)}\|_2^2 \tag{A.19}$$

$$= 16\left(\frac{\lambda_n+\lambda_t-\lambda_1}{\lambda_n-\lambda_t}\right)^2 \sum_{i=1}^{t} \tan^2(\theta^{(i)})\|\Lambda_2\|_2\|\xi_1^{(i)}\|_2^2 \tag{A.20}$$

$$\leqslant 16\left(\frac{\lambda_n+\lambda_t-\lambda_1}{\lambda_n-\lambda_t}\right)^2 \max_{i\in\{1,\dots,t\}} \tan^2(\theta^{(i)}) \sum_{i=1}^{t} \|\Lambda_2\|_2\|\xi_1^{(i)}\|_2^2 \tag{A.21}$$

$$\leqslant 16\left(\frac{\lambda_n+\lambda_t-\lambda_1}{\lambda_n-\lambda_t}\right)^2 \max_{i\in\{1,\dots,t\}} \tan^2(\theta^{(i)})\|\Lambda_2\|_2 \sum_{i=1}^{t} \|\xi_1^{(i)}\|_2^2 \tag{A.22}$$

$$\leqslant 16\left(\frac{\lambda_n+\lambda_t-\lambda_1}{\lambda_n-\lambda_t}\right)^2 \max_{i\in\{1,\dots,t\}} \tan^2(\theta^{(i)})\lambda_n\|x_0-x_*\|_2^2. \tag{A.23}$$

- $\sum_{i=1}^{t} C_3^{(i)}$ can be rewritten as,

$$\sum_{i=1}^{t} C_3^{(i)} = \sum_{i=1}^{t} 16\left(\frac{\lambda_n+\lambda_t-\lambda_1}{\lambda_n-\lambda_t}\right)\tan(\theta^{(i)})\|\Lambda_2\xi_2^{(i)}\|_2\|\xi_1^{(i)}\|_2 \tag{A.24}$$

$$= 16\left(\frac{\lambda_n+\lambda_t-\lambda_1}{\lambda_n-\lambda_t}\right)\sum_{i=1}^{t} \tan(\theta^{(i)})\|\Lambda_2\xi_2^{(i)}\|_2\|\xi_1^{(i)}\|_2 \tag{A.25}$$

$$\leqslant 16\left(\frac{\lambda_n+\lambda_t-\lambda_1}{\lambda_n-\lambda_t}\right)\max_{i\in\{1,\dots,t\}} \tan(\theta^{(i)}) \sum_{i=1}^{t} \|\Lambda_2\xi_2^{(i)}\|_2\|\xi_1^{(i)}\|_2 \tag{A.26}$$

$$\leqslant 16\left(\frac{\lambda_n+\lambda_t-\lambda_1}{\lambda_n-\lambda_t}\right)\max_{i\in\{1,\dots,t\}} \tan(\theta^{(i)})\lambda_n\|x_0-x_*\|_2. \tag{A.27}$$

Using (A.18), (A.23) and (A.27) the conclusion follows. □

A.2 Numerical experiments on the BUNDLE test case

We now test our implementation on a matrix generated using the CFD solver *Code_Saturne* that solves the Navier-Stokes equations for incompressible flows. This code is developed in-house by EDF and is distributed under the GPL open source license (available at `http://www.code-saturne.org`). *Code_Saturne* has been included in the Unified European Applications Benchmark Suite of the PRACE project (see `http://www.praceri.eu/ueabs`). It is used for a wide range of applications, many of which are re-

Name	Size	Nonzeros	Problem
BUNDLE	13,044,996	347,890,620	CFD

Table A.1 – The BUNDLE test case.

lated to nuclear engineering, but increasingly with applications related to renewables. We focus on simulations such as the computation of fluid flow in tube bundles, either cross-flow as in steam generators, or tangential as in Pressurized Water Reactor fuel assemblies, as these applications are numerically quite representative of a broader range of applications, and large-scale meshes for benchmarking are available. We have collaborated with Yvan Fournier from EDF to test ECG on a matrix corresponding to the BUNDLE test case.

In all the experiments the tolerance is set as the default tolerance of PETSc, *i.e.*, 10^{-5} and the maximum number of iterations is set to 5000. The right-hand side is chosen uniformly random, and then normalized. The initial guess is set to 0. The following experiments are performed on a Kebnekaise. We do not use any kind of threading and use 28 MPI processes per node, each one being bound to one OpenMP thread.

A.2.1 Impact of the enlarging factor

First we study the impact of the enlarging factor t on the methods. We fix the number of processors to 112 and we vary the value of t for the 4 methods: Orthodir (Odir), Orthodir with dynamic reduction of the search directions (D-Odir), Orthomin (Omin) and Breakdown-Free Orthomin (BF-Omin). The results obtained are summarized in Table 3.4

	t	Odir	D-Odir	Omin	BF-Omin
BUNDLE	1	20.3	20.4	20.0	20.2
	2	23.1	23.0	23.9	23.1
	4	28.7	28.8	28.8	28.8
	8	36.8	36.4	36.7	36.4

Table A.2 – Runtime results for the BUNDLE test case ($N_{\text{proc}} = 112$)..

For the BUNDLE matrix, the reduction of the number of iteration is not balancing the increase in flops and the runtime is slightly increasing when t increases. In fact, the number of iterations when $t = 1$ is already very low in this case (85) and increasing t does not allow to reduce it significantly. For instance, when $t = 8$ the number of iterations is 69, a decrease of only 20%. Using the dynamic reduction of the search directions is not very effective for this matrix because it generally occurs when the

method is about to converge.

A.2.2 Strong scaling study

Following the parameter study, we perform a strong scaling study on BUNDLE. We compare PETSc PCG and Omin with $t = 4$. We do not use the dynamic reduction of the search directions as it is not very effective for this matrix, and we use Omin because it is a bit cheaper that Odir and does not breakdown for this matrix. The results are summarized in Table A.3. We observe that both PETSc PCG and ECG are scaling very well. Nevertheless, in this case enlarging the Krylov subspaces does not allow to reduce significantly the number of iterations and ECG remains almost 2 times slower than PETSc PCG.

	Omin(4)			CG		
# MPI	# iter	res	time (s)	# iter	res	time (s)
252	78	1.3E-4	12.1	89	1.3E-4	7.4
504	88	1.8E-4	5.9	98	1.9E-4	3.4
1,008	95	2.6E-4	2.9	105	2.7E-4	1.5

Table A.3 – Strong scaling results for BUNDLE.

A.3 Numerical experiments on an elasticity problem discretized using PETSc

In order to demonstrate the versatility of our parallel design, we present results where we consider an elasticity problem which is discretized using PETSc.

A.3.1 Definition of the problem

The following experiments are performed on Cori. We use PETSc to discretize a linear elasticity problem of the form,

$$\mathrm{div}(\sigma(u)) + f = 0 \text{ on } \Omega, \tag{A.28}$$

where Ω is a unit cube, f is some body force, u is the unknown displacement field, and $\sigma(.)$ is the Cauchy stress tensor given by Hooke's law: it can be expressed in terms of Young's Modulus E and Poisson's ratio ν. For a more detailed description of the problem see [62, 74, 108]. We consider a cube $(E, \nu) = (1, 0.25)$ made of several spheres of a hard material $(E_+, \nu) = (10^i, 0.25)$ and a soft material $(E_-, \nu) = (10^{-i}, 0.25)$ where i is an integer, *i.e.*, discontinuous E (see Figure A.1). We modify the example number 56 of PETSc[1] in order to discretize this equation using $\mathbb{Q}_1$ finite elements. This means that

[1] http://www.mcs.anl.gov/petsc/petsc-current/src/ksp/ksp/examples/tutorials/ex56.c.html

there are 81 non-zero entries per row in the global matrix. Furthermore, we also use PETSc to apply the sparse matrix–set of vectors product, and the block Jacobi preconditioner. We use the default parameter of PETSc for the block Jacobi preconditioner which means that an imcomplete Cholesky factorization with no fill–in (ICC0) is performed on the diagonal blocks. This preconditioner is of course less effective in terms of iterations than a block Jacobi with the complete factorization, but it is also cheaper and it increases the load balancing between the processors. As before, the tolerance of the stopping criterion is set to 10^{-5}.

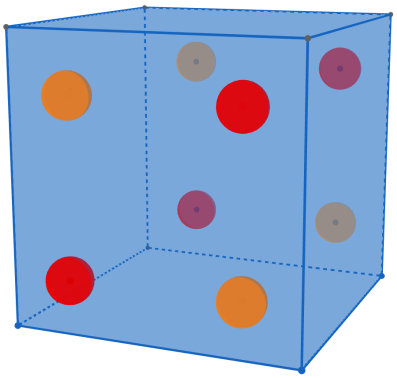


Figure A.1 – Pattern of the heterogeneity of E, there are spheres of a hard and a soft material.

A.3.2 Numerical results

In the following experiments, we are interested in assessing the robustness of the method with respect to the physical problem. More precisely, we vary the heterogeneity jump of the Young's Modulus, *i.e.*, we consider different values of i in the expressions of E_+ and E_-.

We use the following command line in order to define the linear system to be solved:

```
srun -n 8000 ./bin/test_ecg_petsc_ela -pc_type bjacobi -sub_pc_type icc -ne
↪  299 -e 28 -r 0
```

In fact, this generates a linear system of 81 million of unknowns, and around 4.5 billion of non-zeros. This system is then solved in parallel with 8,000 MPI processes using PETSc CG, Odir(28), and Fused Odir(28) for different values of i in the expression of E_+ and E_-. We do not use any kind of threading and use 64 MPI processes per KNL node — from the 68 possible cores, 4 cores are dedicated to the so–called core specialization. The results are summarized in Table A.4. We observe that the number of iterations

with Odir, and D-Odir is rather constant with respect to the increase of the jump of the heterogeneity of E. It is not the case for the usual CG method, and the number of iteration is almost doubling as the jump is increasing. This of course is somehow translated in the resulting runtimes. We observe that ECG's runtimes (Odir and Fused Odir) are almost constant (around 140 seconds) with respect to the jump. On the other hand, CG's runtime is increased by a factor $\frac{109}{58} \approx 1.9$. Despite this increase, we notice that PETSc's CG is still faster than Odir and Fused Odir. This is mainly due to the fact that we apply the preconditioner, and the sparse matrix–dense matrix product within PETSc but PETSc has not been designed to handle block operations. Thus, it is very likely that these operations could be optimized in order to mitigate the extra arithmetic costs. We want to point out that it is indeed the arithmetic costs which are dominating in this case, and the very slight difference in runtime between Odir and Fused Odir is an illustration of this. Also the difference in terms of iterations between Odir and Fused Odir is very likely due to the non–reproducible behavior of MPI, and not to the reformulation of the algorithm. At this scale it is visible although not really significant in the runtimes.

	Fused Odir		Odir		PETSc's CG	
(E_+, E_-)	# iter	time (s)	# iter	time (s)	# iter	time (s)
$(10^4, 10^{-4})$	587	139.4	589	142.2	6,202	58.8
$(10^5, 10^{-5})$	593	129.6	585	128.0	6,446	61.9
$(10^6, 10^{-6})$	639	140.9	630	138.8	9,482	85.4
$(10^7, 10^{-7})$	675	147.1	672	147.5	11,924	109.6

Table A.4 – Iteration count and runtimes obtained for an elasticity problem with 81 million of unknowns and around 6.5 billion of non-zeros. The enlarging factor is set to 28, and there is no dynamic reduction of the search directions.

Bibliography

[1] M. Abalenkovs et al. *Deliverable 5.2: Software integration*. NLAFET deliverable. 2018.

[2] Y. Achdou and F. Nataf. "Low frequency tangential filtering decomposition". In: *Numerical Linear Algebra with Applications* 14.2 (2007), pp. 129–147.

[3] E. Agullo, L. Giraud, and Y.-F. Jing. "Block GMRES Method with Inexact Breakdowns and Deflated Restarting". In: *SIAM Journal on Matrix Analysis and Applications* 35.4 (2014), pp. 1625–1651. eprint: https://doi.org/10.1137/140961912.

[4] E. Agullo et al. "Numerical linear algebra on emerging architectures: The PLASMA and MAGMA projects". In: *Journal of Physics: Conference Series* 180.1 (2009), p. 012037.

[5] H. Al Daas, L. Grigori, P. Hénon, and P. Ricoux. "Enlarged GMRES for solving linear systems with one or multiple right-hand sides". In: *IMA Journal of Numerical Analysis* (2018), dry054. eprint: /oup/backfile/content_public/journal/imajna/pap/10.1093_imanum_dry054/5/dry054.pdf.

[6] S. F. Ashby, T. A. Manteuffel, and P. E. Saylor. "A Taxonomy for Conjugate Gradient Methods". In: *SIAM Journal on Numerical Analysis* 27.6 (1990), pp. 1542–1568.

[7] C. Augonnet, S. Thibault, R. Namyst, and P.-A. Wacrenier. "StarPU: A Unified Platform for Task Scheduling on Heterogeneous Multicore Architectures". In: *Concurrency and Computation: Practice and Experience, Special Issue: Euro-Par 2009* 23 (2 Feb. 2011), pp. 187–198.

[8] S. Balay et al. *PETSc Users Manual*. Tech. rep. ANL-95/11 - Revision 3.8. Argonne National Laboratory, 2017.

[9] A. Bhaya, P.-A. Bliman, G. Niedu, and F. Pazos. "A cooperative conjugate gradient method for linear systems permitting multithread implementation of low complexity". In: *ArXiv e-prints* (2012). arXiv: 1204.0069 [math.NA].

[10] A. Bienz, R. Falgout, W. Gropp, L. Olson, and J. Schroder. "Reducing Parallel Communication in Algebraic Multigrid through Sparsification". In: *SIAM Journal on Scientific Computing* 38.5 (2016), S332–S357. eprint: https://doi.org/10.1137/15M1026341.

[11] G. Bosilca, A. Bouteiller, A. Danalis, M. Faverge, T. Herault, and J. J. Dongarra. "PaRSEC: Exploiting Heterogeneity to Enhance Scalability". In: *Computing in Science Engineering* 15.6 (2013), pp. 36–45.

[12] C. Brezinski. "Multiparameter descent methods". In: *Linear Algebra and its Applications* 296.1 (1999), pp. 113 –141.

[13] C. Brezinski and F. Bantegnies. "The multiparameter conjugate gradient algorithm". In: *Matapli* 75 (2004), pp. 67–85.

[14] R. Bridson and C. Greif. "A Multipreconditioned Conjugate Gradient Algorithm". In: *SIAM J. Matrix Analysis Applications* 27 (2006), pp. 1056–1068.

[15] H. Calandra, S. Gratton, R. Lago, X. Vasseur, and L. Carvalho. "A Modified Block Flexible GMRES Method with Deflation at Each Iteration for the Solution of Non-Hermitian Linear Systems with Multiple Right-Hand Sides". In: *SIAM Journal on Scientific Computing* 35.5 (2013), S345–S367. eprint: https://doi.org/10.1137/120883037.

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

[17] E. Carson and J. Demmel. "A Residual Replacement Strategy for Improving the Maximum Attainable Accuracy of *s*-Step Krylov Subspace Methods". In: *SIAM Journal on Matrix Analysis and Applications* 35.1 (2014), pp. 22–43. eprint: https://doi.org/10.1137/120893057.

[18] E. Carson, M. Rozloznik, Z. Strakos, P. Tichy, and M. Tuma. *On the numerical stability analysis of pipelined Krylov subspace methods*. Research Report NCMM/2016/08. Necas Center for Mathematical Modeling, 2016.

[19] E. Carson. "Communication-Avoiding Krylov Subspace Methods in Theory and Practice". PhD thesis. University of California, Berkeley, 2015.

[20] E. Carson. "The Adaptive *s*-step Conjugate Gradient Method". In: *CoRR* abs/1701.03989 (2017). arXiv: 1701.03989.

[21] E. Carson, N. Knight, and J. Demmel. "An efficient deflation technique for the communication-avoiding conjugate gradient method". In: *Electronic Transactions on Numerical Analysis* 43.125-141 (2014), p. 09.

[22] E. Carson, N. Knight, and J. Demmel. "Avoiding Communication in Nonsymmetric Lanczos-Based Krylov Subspace Methods". In: *SIAM Journal on Scientific Computing* 35.5 (2013), S42–S61. eprint: https://doi.org/10.1137/120881191.

[23] F. Charbel and F.-X. Roux. "A method of finite element tearing and interconnecting and its parallel solution algorithm". In: *International Journal for Numerical Methods in Engineering* 32.6 (), pp. 1205–1227. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/nme.1620320604.

[24] J. Chen. *A deflated version of the block conjugate gradient algorithm with an application to Gaussian process maximum likelihood estimation*. Preprint P1927-0811. 2011.

[25] A. T. Chronopoulos. "*s*-step Iterative Methods for (Non)Symmetric (In)Definite Linear Systems". In: *SIAM Journal on Numerical Analysis* 28.6 (1991), pp. 1776–1789. eprint: https://doi.org/10.1137/0728088.

[26] A. T. Chronopoulos and C. W. Gear. "*s*-step iterative methods for symmetric linear systems". In: *Journal of Computational and Applied Mathematics* 25.2 (1989), pp. 153 –168.

[27] A. T. Chronopoulos and C. D. Swanson. "Parallel iterative *s*-step methods for unsymmetric linear systems". In: *Parallel Computing* 22.5 (1996), pp. 623–641.

[28] M. Clark, A. Strelchenko, A. Vaquero, M. Wagner, and E. Weinberg. "Pushing memory bandwidth limitations through efficient implementations of Block-Krylov space solvers on GPUs". In: *Computer Physics Communications* (2018).

[29] S. Cools. "Numerical stability analysis of the class of communication hiding pipelined Conjugate Gradient methods". In: *ArXiv e-prints* (Apr. 2018). arXiv: 1804.02962 [cs.NA].

[30] S. Cools and W. Vanroose. "Numerically Stable Variants of the Communication-hiding Pipelined Conjugate Gradients Algorithm for the Parallel Solution of Large Scale Symmetric Linear Systems". In: *ArXiv e-prints* (June 2017). arXiv: 1706.05988 [cs.NA].

[31] J. Cornelis, S. Cools, and W. Vanroose. "The Communication-Hiding Conjugate Gradient Method with Deep Pipelines". In: *ArXiv e-prints* (Jan. 2018). arXiv: 1801.04728 [cs.DC].

[32] T. A. Davis and Y. Hu. "The University of Florida Sparse Matrix Collection". In: *ACM Trans. Math. Softw.* 38.1 (Dec. 2011), 1:1–1:25.

[33] J. W. Demmel, L. Grigori, M. Gu, and H. Xiang. "Communication Avoiding Rank Revealing QR Factorization with Column Pivoting". In: *SIAM Journal on Matrix Analysis and Applications* 36.1 (2015), pp. 55–89. eprint: https://doi.org/10.1137/13092157X.

[34] J. Demmel, L. Grigori, M. Hoemmen, and J. Langou. "Communication-optimal Parallel and Sequential QR and LU Factorizations". In: *SIAM Journal on Scientific Computing* 34.1 (2012), A206–A239. eprint: https://doi.org/10.1137/080731992.

[35] J. Demmel, M. F. Hoemmen, M. Mohiyuddin, and K. A. Yelick. *Avoiding Communication in Computing Krylov Subspaces*. Tech. rep. UCB/EECS-2007-123. EECS Department, University of California, Berkeley, 2007.

[36] V. Dolean, P. Jolivet, and F. Nataf. *An introduction to domain decomposition methods*. Algorithms, theory, and parallel implementation. Philadelphia: SIAM, 2015.

[37] S. Donfack, L. Grigori, and O. Tissot. *Deliverable 4.2: Analysis and algorithm design*. NLAFET deliverable. 2017.

[38] S. Donfack, L. Grigori, and O. Tissot. *Deliverable 4.3: Prototype software, phase 2*. NLAFET deliverable. 2017.

[39] S. Donfack, L. Grigori, and O. Tissot. *Deliverable 4.4: Performance evaluation*. NLAFET deliverable. 2018.

[40] S. Donfack, L. Grigori, and O. Tissot. *Deliverable 4.5: Integration*. NLAFET deliverable. 2018.

[41] J. Dongarra, V. Eijkhout, and A. Kalhan. *Reverse Communication Interface for Linear Algebra Templates for Iterative Methods*. Tech. rep. Knoxville, TN, USA, 1995.

[42] J. Dongarra et al. "With Extreme Computing, the Rules Have Changed". In: *Computing in Science Engineering* 19.3 (2017), pp. 52–62.

[43] J. J. Dongarra, J. Du Croz, S. Hammarling, and I. S. Duff. "A Set of Level 3 Basic Linear Algebra Subprograms". In: *ACM Trans. Math. Softw.* 16.1 (Mar. 1990), pp. 1–17.

[44] J. Dongarra, M. A. Heroux, and P. Luszczek. "High-performance conjugate-gradient benchmark: A new metric for ranking high-performance computing systems". In: *The International Journal of High Performance Computing Applications* 30.1 (2016), pp. 3–10. eprint: https://doi.org/10.1177/1094342015593158.

[45] Z. Dostál. "Conjugate gradient method with preconditioning by projector". In: *International Journal of Computer Mathematics* 23.3-4 (1988), pp. 315–323.

[46] A. Dubrulle. "Retooling the method of block conjugate gradients." eng. In: *ETNA. Electronic Transactions on Numerical Analysis [electronic only]* 12 (2001), pp. 216–233.

[47] I. S. Duff, A. M. Erisman, and J. K. Reid. *Direct Methods for Sparse Matrices*. New York: Clarendon Press, 1989.

[48] J. Frank and C. Vuik. "On the Construction of Deflation-Based Preconditioners". In: *SIAM Journal on Scientific Computing* 23.2 (2001), pp. 442–462. eprint: https://doi.org/10.1137/S1064827500373231.

[49] R. W. Freund and N. M. Nachtigal. "QMR: a quasi-minimal residual method for non-Hermitian linear systems". In: *Numerische Mathematik* 60.1 (1991), pp. 315–339.

[50] A. Frommer and H. Schwandt. "A Unified Representation and Theory of Algebraic Additive Schwarz and Multisplitting Methods". In: *SIAM Journal on Matrix Analysis and Applications* 18.4 (1997), pp. 893–912. eprint: https://doi.org/10.1137/S0895479896301212.

[51] A. Frommer and D. B. Szyld. "An Algebraic Convergence Theory for Restricted Additive Schwarz Methods Using Weighted Max Norms". In: *SIAM Journal on Numerical Analysis* 39.2 (2001), pp. 463–479. eprint: https://doi.org/10.1137/S0036142900370824.

[52] A. Gaul, M. H. Gutknecht, J. Liesen, and R. Nabben. "A Framework for Deflated and Augmented Krylov Subspace Methods". In: *SIAM Journal on Matrix Analysis and Applications* 34.2 (2013), pp. 495–518.

[53] P. Ghysels, T. Ashby, K. Meerbergen, and W. Vanroose. "Hiding Global Communication Latency in the GMRES Algorithm on Massively Parallel Machines". In: *SIAM Journal on Scientific Computing* 35.1 (2013), pp. C48–C71. eprint: https://doi.org/10.1137/12086563X.

[54] P. Ghysels and W. Vanroose. "Hiding global synchronization latency in the preconditioned Conjugate Gradient algorithm". In: *Parallel Computing* 40.7 (2014). 7th Workshop on Parallel Matrix Algorithms and Applications, pp. 224 –238.

[55] G. H. Golub and C. F. Van Loan. *Matrix Computations (3rd Ed.)* Baltimore, MD, USA: Johns Hopkins University Press, 1996.

[56] A. Greenbaum. "Estimating the Attainable Accuracy of Recursively Computed Residual Methods". In: *SIAM Journal on Matrix Analysis and Applications* 18.3 (1997), pp. 535–551. eprint: https://doi.org/10.1137/S0895479895284944.

[57] L. Grigori and S. Moufawad. "Communication Avoiding ILU0 Preconditioner". In: *SIAM Journal on Scientific Computing* 37.2 (2015), pp. C217–C246. eprint: https://doi.org/10.1137/130930376.

[58] L. Grigori, S. Moufawad, and F. Nataf. "Enlarged Krylov Subspace Conjugate Gradient Methods for Reducing Communication". In: *SIAM Journal on Matrix Analysis and Applications* 37.2 (2016), pp. 744–773. eprint: https://doi.org/10.1137/140989492.

[59] L. Grigori, J. Papez, R. Stompor, and O. Tissot. *Solving sequences of linear systems by recycling deflation subspaces - application to CMB data analysis*. In preparation. 2018.

[60] L. Grigori, R. Stompor, and M. Szydlarski. "A parallel two-level preconditioner for Cosmic Microwave Background map-making". In: *High Performance Computing, Networking, Storage and Analysis (SC), 2012 International Conference for*. 2012, pp. 1–10.

[61] L. Grigori and O. Tissot. *Reducing the communication and computational costs of Enlarged Krylov subspaces Conjugate Gradient*. Research Report (old version) RR-9023. Inria Paris, Laboratoire Jacques-Louis Lions, UPMC, Paris, 2017.

[62] L. Grigori, F. Nataf, and S. Yousef. *Robust algebraic Schur complement preconditioners based on low rank corrections*. Research Report RR-8557. 2014, p. 18.

[63] L. Grigori and O. Tissot. *Scalable Linear Solvers based on Enlarged Krylov subspaces with Dynamic Reduction of Search Directions*. Research Report RR-9190. Inria Paris, Laboratoire Jacques-Louis Lions, UPMC, Paris, 2018.

[64] A. el Guennouni, K. Jbilou, and H. Sadok. "A block version of BiCGSTAB for linear systems with multiple right-hand sides." eng. In: *ETNA. Electronic Transactions on Numerical Analysis [electronic only]* 16 (2003), pp. 129–142.

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

[66] R. Haferssas, P. Jolivet, and S. Rubino. "Efficient and scalable discretization of the Navier-Stokes equations with LPS modeling". In: *Computer Methods in Applied Mechanics and Engineering* 333 (2018), pp. 371 –394.

[67] F. Hecht. "New development in FreeFem++". In: *J. Numer. Math.* 20.3-4 (2012), pp. 251–265.

[68] M. R. Hestenes and E. Stiefel. "Methods of conjugate gradients for solving linear systems." In: *Journal of research of the National Bureau of Standards.* 49 (1952), pp. 409–436.

[69] M. Hoemmen. "Communication-avoiding Krylov subspace methods". PhD thesis. EECS Department, University of California, Berkeley, 2010.

[70] D. Imberti and J. Erhel. "Varying the s in Your *s*-step GMRES". In: *Electronic Transactions on Numerical Analysis (ETNA)* 47 (2017), pp. 206–230.

[71] H. Ji and Y. Li. "A breakdown-free block conjugate gradient method". In: *BIT Numerical Mathematics* 57.2 (2017), pp. 379–403.

[72] P. Jiránek, M. Rozložník, and M. Gutknecht. "How to Make Simpler GMRES and GCR More Stable". In: *SIAM Journal on Matrix Analysis and Applications* 30.4 (2009), pp. 1483–1499. eprint: https://doi.org/10.1137/070707373.

[73] P. Jolivet. "Domain decomposition methods. Application to high-performance computing". Theses. Université de Grenoble, Oct. 2014.

[74] P. Jolivet, F. Hecht, F. Nataf, and C. Prud'homme. "Scalable Domain Decomposition Preconditioners For Heterogeneous Elliptic Problems". In: *Proceedings of the 2013 International Conference on High Performance Computing, Networking, Storage and Analysis*. SC13. ACM, 2013, 80:1–80:11.

[75] P. Jolivet and P.-H. Tournier. "Block Iterative Methods and Recycling for Improved Scalability of Linear Solvers". In: *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. SC '16. Salt Lake City, Utah: IEEE Press, 2016, 17:1–17:14.

[76] G. Karypis and V. Kumar. "METIS – Unstructured Graph Partitioning and Sparse Matrix Ordering System, Version 2.0". In: (1995).

[77] M. E. Kilmer and E. de Sturler. "Recycling Subspace Information for Diffuse Optical Tomography". In: *SIAM Journal on Scientific Computing* 27.6 (2006), pp. 2140–2166. eprint: https://doi.org/10.1137/040610271.

[78] T. Konolige and J. Brown. "A Parallel Solver for Graph Laplacians". In: *ArXiv e-prints* (May 2017). arXiv: 1705.06266 [cs.DC].

[79] M. Kreutzer et al. "GHOST: Building Blocks for High Performance Sparse Linear Algebra on Heterogeneous Systems". In: *International Journal of Parallel Programming* 45.5 (2017), pp. 1046–1072.

[80] J. Langou. "Iterative methods for solving linear systems with multiple right-hand sides". PhD thesis. CERFACS, 2003.

[81] B. R. Lowery and J. Langou. *Stability Analysis of QR factorization in an Oblique Inner Product*. Tech. rep. 2014. arXiv: 1401.5171.

[82] J. Malek and Z. Strakos. *Preconditioning and the Conjugate Gradient Method in the Context of Solving PDEs*. SIAM Spotlights. Philadelphia: SIAM, 2014.

[83] R. B. Morgan. "A Restarted GMRES Method Augmented with Eigenvectors". In: *SIAM Journal on Matrix Analysis and Applications* 16.4 (1995), pp. 1154–1171. eprint: https://doi.org/10.1137/S0895479893253975.

[84] A. Napov and Y. Notay. "An Algebraic Multigrid Method with Guaranteed Convergence Rate". In: *SIAM Journal on Scientific Computing* 34.2 (2012), A1079–A1109. eprint: https://doi.org/10.1137/100818509.

[85] R. Nicolaides. "Deflation of Conjugate Gradients with Applications to Boundary Value Problems". In: *SIAM Journal on Numerical Analysis* 24.2 (1987), pp. 355–365. eprint: https://doi.org/10.1137/0724027.

[86] A. A. Nikishin and A. Yeremin. "Variable Block CG Algorithms for solving large sparse symmetric positive definite linear systems on parallel computers, I: General Iterative Scheme". In: *SIAM Journal on Matrix Analysis and Applications* 16 (1995), pp. 1135–1153.

[87] Q. Niu, L. Grigori, P. Kumar, and F. Nataf. "Modified tangential frequency filtering decomposition and its fourier analysis". In: *Numerische Mathematik* 116 (2010), pp. 123–148.

[88] M. O'Connell, M. E. Kilmer, E. de Sturler, and S. Gugercin. "Computing Reduced Order Models via Inner-Outer Krylov Recycling in Diffuse Optical Tomography". In: *SIAM Journal on Scientific Computing* 39.2 (2017), B272–B297. eprint: https://doi.org/10.1137/16M1062880.

[89] D. P. O'Leary. "The block conjugate gradient algorithm and related methods." In: *Linear Algebra and Its Applications* 29 (1980), pp. 293–322.

[90] C. Paige. "Accuracy and effectiveness of the Lanczos algorithm for the symmetric eigenproblem". In: *Linear Algebra and its Applications* 34 (1980), pp. 235 – 258.

[91] C. C. Paige, B. N. Parlett, and H. A. van der Vorst. "Approximate solutions and eigenvalue bounds from Krylov subspaces". In: *Numerical Linear Algebra with Applications* 2 (1995), pp. 115–133.

[92] J. Papez, L. Grigori, and R. Stompor. "Solving linear equations with messenger-field and conjugate gradients techniques - an application to CMB data analysis". In: *ArXiv e-prints* (Mar. 2018). arXiv: 1803.03462.

[93] M. L. Parks, E. de Sturler, G. Mackey, D. D. Johnson, and S. Maiti. "Recycling Krylov subspaces for sequences of linear systems". In: *SIAM J. Sci. Comput.* 28.5 (2006), pp. 1651–1674.

[94] G. Puglisi, D. Poletti, G. Fabbian, C. Baccigalupi, L. Heltai, and R. Stompor. "Iterative map-making with two-level preconditioning for polarized Cosmic Microwave Background data sets". In: *ArXiv e-prints* (Jan. 2018). arXiv: 1801.08937.

[95] M. Robbé and M. Sadkane. "Exact and Inexact Breakdowns in the block GMRES Method". In: *Linear Algebra Appl.* 419 (2006), pp. 265–285.

[96] M. Rozložník, M. Tůma, A. Smoktunowicz, and J. Kopal. "Numerical stability of orthogonalization methods with a non-standard inner product". In: *BIT Numerical Mathematics* 52.4 (2012), pp. 1035–1058.

[97] J. Rudi et al. "An Extreme-scale Implicit Solver for Complex PDEs: Highly Heterogeneous Flow in Earth's Mantle". In: *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. SC '15. Austin, Texas: ACM, 2015, 5:1–5:12.

[98] Y. Saad. *Numerical Methods for Large Eigenvalue Problems*. Philadelphia: SIAM, 2011. eprint: https://epubs.siam.org/doi/pdf/10.1137/1.9781611970739.

[99] Y. Saad and H. S. M. "GMRES: a generalized minimal residual algorithm for solving non-symmetric linear systems." In: *SIAM J. Sci. Statist. Comput.* 7 (1986), pp. 856–869.

[100] Y. Saad, M. Yeung, J. Erhel, and F. Guyomarc'h. "A Deflated Version of the Conjugate Gradient Algorithm". In: *SIAM Journal on Scientific Computing* 21.5 (2000), pp. 1909–1926. eprint: https://doi.org/10.1137/S1064829598339761.

[101] Y. Saad. *Iterative methods for sparse linear systems*. Philadelphia: SIAM, 2003.

[102] V. Simoncini and D. Szyld. "Theory of Inexact Krylov Subspace Methods and Applications to Scientific Computing". In: *SIAM Journal on Scientific Computing* 25.2 (2003), pp. 454–477. eprint: https://doi.org/10.1137/S1064827502406415.

[103] G. L. G. Sleijpen and H. A. Van der Vorst. "A Jacobi-Davidson iteration method for linear eigenvalue problems". In: *SIAM Review* 42.2 (2000), pp. 267–293.

[104] K. Soodhalter. "A block MINRES algorithm based on the band Lanczos method". In: *Numerical Algorithms* 69 (Jan. 2013).

[105] D. C. Sorensen. "Implicit application of polynomial filters in a k-step Arnoldi method". In: *SIAM Journal on Matrix Analysis and Applications* 13.1 (1992), pp. 357–385.

[106] D. C. Sorensen. "Numerical methods for large eigenvalue problems". In: *Acta Numerica* 11 (2002), 519–584.

[107] N. Spillane, V. Dolean, P. Hauret, F. Nataf, C. Pechstein, and R. Scheichl. "Abstract robust coarse spaces for systems of PDEs via generalized eigenproblems in the overlaps". In: *Numerische Mathematik* 126.4 (2014), pp. 741–770.

[108] N. Spillane. "An Adaptive MultiPreconditioned Conjugate Gradient Algorithm". In: *SIAM Journal on Scientific Computing* 38 (Jan. 2016), A1896–A1918.

[109] G. W. Stewart. "A Krylov-Schur algorithm for large eigenproblems". In: *SIAM Journal on Matrix Analysis and Applications* 23.3 (2001/02), pp. 601–614.

[110] R. Stompor, S. Leach, F. Stivoli, and C. Baccigalupi. "Maximum likelihood algorithm for parametric component separation in cosmic microwave background experiments". In: *Monthly Notices of the Royal Astronomical Society* 392.1 (2009), pp. 216–232. eprint: /oup/backfile/content_public/journal/mnras/392/1/10.1111/j.1365-2966.2008.14023.x/2/mnras0392-0216.pdf.

[111] E. de Sturler. "Truncation Strategies for Optimal Krylov Subspace Methods". In: *SIAM Journal on Numerical Analysis* 36.3 (1999), pp. 864–889. eprint: https://doi.org/10.1137/S0036142997315950.

[112] H. Sundar, G. Biros, C. Burstedde, J. Rudi, O. Ghattas, and G. Stadler. "Parallel geometric-algebraic multigrid on unstructured forests of octrees". In: *High Performance Computing, Networking, Storage and Analysis (SC), 2012 International Conference for*. 2012, pp. 1–11.

[113] M. Szydlarski, L. Grigori, and R. Stompor. "Accelerating the cosmic microwave background map-making procedure through preconditioning". In: *Astronomy & Astrophysics* 572, A39 (Dec. 2014), A39. arXiv: 1408.3048.

[114] M. Szydlarski, L. Grigori, and R. Stompor. "Accelerating the cosmic microwave background map-making procedure through preconditioning". In: *A&A* 572 (2014), A39.

[115] J. M. Tang, R. Nabben, C. Vuik, and Y. A. Erlangga. "Comparison of Two-Level Preconditioners Derived from Deflation, Domain Decomposition and Multigrid Methods". In: *Journal of Scientific Computing* 39.3 (2009), pp. 340–370.

[116] R. Thakur, R. Rabenseifner, and W. Gropp. "Optimization of Collective Communication Operations in MPICH". In: *The International Journal of High Performance Computing Applications* 19.1 (2005), pp. 49–66. eprint: https://doi.org/10.1177/1094342005051521.

[117] A. Toselli and O. Widlund. *Domain Decomposition Methods - Algorithms and Theory*. Vol. 34. Springer Series in Computational Mathematics. New York: Springer Berlin Heidelberg, 2004.

[118] H. van der Vorst. "Bi-CGSTAB: A Fast and Smoothly Converging Variant of Bi-CG for the Solution of Nonsymmetric Linear Systems". In: *SIAM Journal on Scientific and Statistical Computing* 13.2 (1992), pp. 631–644. eprint: https://doi.org/10.1137/0913035.

[119] E. Wang et al. "Intel math kernel library". In: *High-Performance Computing on the Intel® Xeon Phi*. Springer, 2014, pp. 167–188.

[120] S. F. M. William L. Briggs Van Emden Henson. *A multigrid tutorial*. Philadelphia: SIAM, 2000.

[121] G. Wu. "The convergence of harmonic Ritz vectors and harmonic Ritz values, revisited". In: *SIAM Journal on Matrix Analysis and Applications* 38.1 (2017), pp. 118–133.

[122] K. Wu and H. Simon. "Thick-restart Lanczos method for large symmetric eigenvalue problems". In: *SIAM J. Matrix Anal. Appl.* 22.2 (2000), pp. 602–616.

[123] I. Yamazaki, S. Tomov, and J. Dongarra. "Mixed-Precision Cholesky QR Factorization and Its Case Studies on Multicore CPU with Multiple GPUs". In: *SIAM Journal on Scientific Computing* 37.3 (2015), pp. C307–C330. eprint: https://doi.org/10.1137/14M0973773.

[124] C. Yang et al. "10M-Core Scalable Fully-Implicit Solver for Nonhydrostatic Atmospheric Dynamics". In: *SC16: International Conference for High Performance Computing, Networking, Storage and Analysis*. 2016, pp. 57–68.

Iterative methods for solving linear systems on massively parallel architectures

Abstract

Krylov methods are widely used for solving large sparse linear systems of equations. On distributed architectures, their performance is limited by the communication needed at each iteration of the algorithm. In this thesis, we first study the use of so-called Enlarged Krylov subspaces for reducing the number of iterations, and therefore the overall communication, of Krylov methods. We consider a reformulation of the Conjugate Gradient (CG) method using these enlarged Krylov subspaces: the Enlarged Conjugate Gradient (ECG) method. This method is first studied from a theoretical point of view. In particular, we show that its convergence speed is close to that of the so-called Deflated Conjugate Gradient method. In order to mitigate the effect of the extra arithmetic operations induced by the method, we explain how to dynamically reduce the number of search directions during the iterations. We then present the parallel design of two variants of the ECG method as well as their corresponding dynamic versions. Using a block Jacobi preconditioner, we show that our implementation scales up to several thousands of cores, and it can be significantly faster than the PETSc implementation of the CG method. We then focus on the Cosmic Microwave Background (CMB) analysis. We investigate the usage of so–called recycling strategies in this context. As a result of the multiplicity of the smallest eigenvalue, these techniques may not improve the convergence in some cases. Hence, we propose a cheap procedure to adapt the initial guess that permits to reduce the overall number of iterations in such situations.

Keywords: linear solvers, parallel computing, Krylov methods, recycling techniques

---

Méthodes itératives de résolution de systèmes linéaires sur des architectures parallèles

Résumé

Les méthodes de Krylov sont largement utilisées pour résoudre des systèmes linéaires creux de grande taille. Sur une architecture distribuée, leur performance est souvent limitée par les communications requises à chaque itération de l'algorithme. Dans cette thèse, nous commençons par étudier l'utilisation des sous–espaces dits de Krylov élargis pour réduire le nombre d'itérations, et ainsi le nombre de communications, des méthodes de Krylov. Nous nous intéressons à une reformulation de la méthode du Gradient Conjugué (CG) qui utilise ces sous–espaces de Krylov élargis : la méthode du Gradient Conjugué Élargi (ECG). Cette méthode est d'abord étudiée d'un point de vue théorique. En particulier, nous montrons que sa vitesse de convergence est proche de celle de la méthode dite du Gradient Conjugué Déflaté. Afin d'atténuer l'effet des opérations arithmétiques supplémentaires requises par la méthode, nous expliquons comment réduire dynamiquement le nombre de directions de recherche pendant les itérations. Nous présentons ensuite le design parallèle des deux variantes de la méthode ECG ainsi que les versions dynamiques qui correspondent. En utilisant un préconditionneur de type bloc Jacobi, nous montrons que notre implémentation est scalable jusqu'à plusieurs milliers de processeurs, et qu'elle peut être significativement plus rapide que l'implémentation de la méthode CG présente dans la librairie PETSc. Nous nous concentrons ensuite sur l'analyse des observations du fond diffus cosmologique. Nous évaluons l'usage des techniques dites de recyclage dans ce contexte. En raison de la multiplicité de la plus petite valeur propre, ces techniques ne permettent pas d'améliorer la convergence dans certains cas. Par conséquent, nous proposons une procédure peu coûteuse pour adapter la solution initiale qui permet de réduire le nombre total d'itérations dans ces situations.

Mots clés : solveurs linéaires, calcul parallèle, méthodes de Krylov, techniques de recyclage

---

Laboratoire Jacques-Louis Lions
4 place Jussieu – 75005 Paris – France