# Accélération de la convergence de méthodes numériques parallèles pour résoudre des systèmes d'équations différentielles linéaires et transitoires non linéaires

Laurent Berenguer

**UNIVERSITÉ DE LYON - UNIVERSITÉ CLAUDE BERNARD - LYON 1**

**ÉCOLE DOCTORALE INFOMATHS**

## THÈSE DE DOCTORAT

**Discipline : MATHÉMATIQUES APPLIQUÉES**

Présentée par
Laurent BERENGUER

# Accélération de la convergence de méthodes numériques parallèles pour résoudre des systèmes d'équations différentielles linéaires et transitoires non linéaires

Préparée à l'Institut Camille Jordan

Dirigée par Damien Tromeur-Dervout et Jean-François Méhaut

Soutenue le 13/10/2014

**Jury :**

| | | | |
|---|---|---|---|
| *Rapporteurs :* | LAURA GRIGORI | - | DR INRIA |
| | FRÉDÉRIC MAGOULÈS | - | PR ECP |
| *Examinateurs :* | ERIC BLAYO | - | PR UJF |
| | BRUNO LACABANNE | - | SIEMENS |
| | HASSANE SADOK | - | PR ULCO |
| *Directeurs :* | DAMIEN TROMEUR-DERVOUT | - | PR U. LYON1 |
| | JEAN-FRANÇOIS MÉHAUT | - | PR UJF |

# Remerciements

J'aimerais tout d'abord remercier mon directeur de thèse, Damien Tromeur-Dervout, pour la confiance qu'il m'a accordée en me proposant d'effectuer ce travail. Je le remercie également pour sa capacité à guider mes recherches tout en me laissant la liberté nécessaire. De même, je souhaiterais remercier mon codirecteur Jean-François Méhaut, ainsi que son équipe, avec qui les échanges ont toujours été constructifs.

Je souhaiterais également exprimer ma gratitude envers ceux qui ont accepté d'évaluer mon travail : les rapporteurs Laura Grigori et Frédéric Magoulès, ainsi que Eric Blayo et Hassane Sadok.

Je remercie Jocelyne Erhel et son équipe, pour m'avoir permis de travailler sur les écoulements en milieux poreux à travers l'ANR-MONU12-0012 H2MNO4. Je remercie en particulier Grégoire Lecourt pour son aide.

J'adresse aussi mes remerciements à la société LMS Imagine / Siemens pour avoir accompagné cette thèse. Je remercie en particulier Bruno Lacabanne pour le temps qu'il m'a consacré afin de me permettre d'appréhender les problématiques industrielles.

Enfin, je remercie Thomas Dufaud et François Pacull avec qui j'ai pu collaborer.

**Cette thèse a été effectuée à l'Institut Camille Jordan.**
**Elle a été soutenue financièrement par la Région Rhône-Alpes.**

# Résumé

La résolution des équations différentielles (EDP/EDO/EDA) est au cœur de la simulation de phénomènes physiques. L'accroissement de la taille et de la complexité des modèles nécessite la mise en œuvre de méthodes de résolution robustes et performantes en termes de temps de calcul. L'objectif de cette thèse est de proposer des méthodes pour accélérer la résolution des équations différentielles par des méthodes de décomposition de domaine.

On considère d'abord les méthodes de décomposition de domaine de Schwarz pour la résolution de grands systèmes linéaires issus de la discrétisation d'EDP. Afin d'accélérer la convergence de la méthode de Schwarz, on propose une approximation de l'opérateur de propagation d'erreur. Cette approximation respectera la structure de l'opérateur exact, ce qui conduira à une réduction très significative des temps de calcul sur le problème des écoulements dans les milieux poreux hétérogènes.

La deuxième contribution concerne la résolution de la suite de systèmes linéaires provenant de l'intégration en temps de problèmes non linéaires. On propose deux approches en utilisant le fait que la matrice jacobienne ne varie que peu d'un système à l'autre. Premièrement, on applique la mise à jour de Broyden au préconditionneur RAS (*Restricted Additive Schwarz*) au lieu de recalculer les factorisations LU. La deuxième approche consiste à dédier des processeurs à la mise à jour partielle et asynchrone du préconditionneur RAS. Des résultats numériques sur le problème de la cavité entraînée et sur un problème de réaction-diffusion montrent qu'une accélération superlinéaire peut être obtenue.

La dernière contribution a pour objet la résolution simultanée des problèmes non linéaires de pas de temps consécutifs. On étudie le cas où la méthode de Broyden est utilisée pour résoudre ces problèmes non linéaires. Dans ce cas, la mise à jour de Broyden peut être propagée d'un pas de temps à l'autre. La parallélisation à travers les pas de temps est également appliquée à la recherche d'une solution initiale consistante pour les équations différentielles algébriques.

**Mots clés :** décomposition de domaine, accélération de suites vectorielles, quasi-Newton, systèmes non linéaires, méthodes asynchrones.

# Abstract

Solving differential equations (PDEs/ODEs/DAEs) is central to the simulation of physical phenomena. The increase in size and complexity of the models requires the design of methods that are robust and efficient in terms of computational time. The aim of this thesis is to design methods that accelerate the solution of differential equations by domain decomposition methods.

We first consider Schwarz domain decomposition methods to solve large-scale linear systems arising from the discretization of PDEs. In order to accelerate the convergence of the Schwarz method, we propose an approximation of the error propagation operator. This approximation preserves the structure of the exact operator. A significant reduction of computational time is obtained for the groundwater flow problem in highly heterogeneous media.

The second contribution concerns solving the sequence of linear systems arising from the time-integration of nonlinear problems. We propose two approaches, taking advantage of the fact that the Jacobian matrix does not change dramatically from one system to another. First, we apply Broyden's update to the Restricted Additive Schwarz (RAS) preconditioner instead of recomputing the local LU factorizations. The second approach consists of dedicating processors to the asynchronous and partial update of the RAS preconditioner. Numerical results for the lid-driven cavity problem, and for a reaction-diffusion problem show that a super-linear speedup may be achieved.

The last contribution concerns the simultaneous solution of nonlinear problems associated to consecutive time steps. We study the case where the Broyden method is used to solve these nonlinear problems. In that case, Broyden's update of the Jacobian matrix may also be propagated from one time step to another. The parallelization through the time steps is also applied to the problem of finding a consistent initial guess for differential-algebraic equations.

**Keywords:** Domain decomposition, acceleration of vector sequences, quasi-Newton, nonlinear systems, asynchronous methods.

# Table des matières

# Liste des figures

# Liste des tableaux

# Liste des algorithmes

# Chapitre 1

# Introduction

La résolution des équations différentielles est au cœur de la simulation numérique de phénomènes physiques. L'accroissement de la taille et de la complexité des modèles nécessite la mise en œuvre de méthodes de résolution robustes et performantes en termes de temps de calcul. Cela signifie que ces méthodes numériques doivent être adaptées à l'évolution des architectures de calcul. Derrière l'évolution de la puissance de calcul (voir la figure 1.1) se cache une certaine complexification des calculateurs : le nombre de processeurs augmente, ainsi que le nombre de cœurs par processeur. On assiste également à une diversification des architectures de calcul, avec notamment l'utilisation des coprocesseurs tels que les cartes graphiques. L'objectif de cette thèse est de concevoir et d'implémenter des méthodes numériques adaptées aux calculs à hautes performances dans le cas de la résolution des équations différentielles par des méthodes implicites. Ces méthodes numériques seront des versions "accélérées" de méthodes existantes, c'est-à-dire des versions moins coûteuses en termes de temps de calcul. Il y a plusieurs façons de réduire les temps de calcul d'une méthode numérique itérative : on peut réduire le nombre d'itérations (techniques de préconditionnement) et le coût de chaque itération (économie de calculs, parallélisation, etc.).

Les méthodes implicites sont souvent incontournables pour des raisons de stabilité. Ainsi, la discrétisation par éléments finis ou volumes finis des équations différentielles conduit à la résolution d'un (dans le cas d'un problème stationnaire linéaire) ou de plusieurs (dans le cas d'un problème transitoire non linéaire) systèmes linéaires creux. Le chapitre 2 présentera un état de l'art de la résolution de ces systèmes linéaires. Dans la première partie de ce chapitre, on présentera rapidement les méthodes itératives de Krylov, qui sont parmi les plus couramment utilisées pour résoudre des problèmes linéaires. L'avantage des méthodes itératives est leur faible coût de calcul et de mémoire par rapport aux méthodes directes. Elles semblent donc être les plus adaptées au calcul à hautes performances, dans la mesure où elles sont essentiellement composées de produit de matrices par des vecteurs et de produits scalaires qui sont aisément parallélisables. L'inconvénient principal des méthodes de Krylov est que leur convergence peut être extrêmement lente lorsque le système est mal conditionné. C'est pourquoi, en pratique, on accélèrera toujours la convergence des méthodes de Krylov par une méthode de préconditionnement. Ces méthodes de préconditionnement sont souvent composées d'un petit nombre d'itérations d'une méthode de résolution. Dans le cas de très

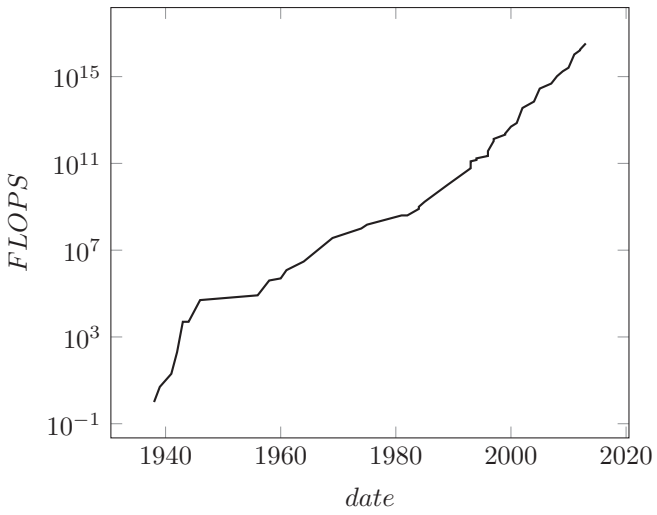


**FIGURE *1.1*** – *Évolution de la puissance (en opérations à virgule flottante par seconde) du supercalculateur le plus puissant.*

grands systèmes linéaires, les communications impliquées par les produits matrice-vecteur et les produits scalaires peuvent être pénalisantes. De plus l'extensibilité des méthodes de Krylov préconditionnées est limitée : selon les problèmes, les coûts de calcul peuvent augmenter plus rapidement que la taille des problèmes traités, et il en est de même pour l'utilisation de la mémoire et les communications. Cette augmentation des coûts de calcul s'explique évidemment par l'accroissement de la taille des données, mais aussi par le fait qu'en général, le nombre d'itérations de Krylov augmente avec la taille du problème. Dans ce cas, les méthodes de décomposition de domaine peuvent s'avérer utiles. Le domaine est ainsi découpé en sous-domaines, ce qui fait apparaître des conditions aux limites artificielles à l'interface des sous-domaines. La solution du problème global se fera uniquement à partir de la résolution indépendante des sous-problèmes associés à chaque sous-domaine, et de l'échange des conditions aux limites. Ces méthodes de décomposition de domaine peuvent donc être très utiles pour résoudre des problèmes qu'il n'aurait pas été possible de résoudre autrement. En pratique, les méthodes de décomposition de domaine sont également très utilisées en tant que préconditionneur. La deuxième partie du chapitre 2 se concentrera sur les méthodes de décomposition de domaine de Schwarz. On étudiera en particulier la convergence de la solution globale puis la convergence de la solution restreinte aux interfaces artificielles.

Le chapitre 3 sera consacré à l'accélération de la méthode de décomposition de domaine de type Schwarz dans le cas d'un problème linéaire. En effet, on peut décrire l'évolution de la solution restreinte aux interfaces par un processus de Richardson, qui converge de manière purement linéaire. Il est donc possible d'utiliser la version vectorielle de l'accélération d'Aitken d'une suite. Cette idée n'est pas nouvelle et remonte à [56]. L'application de la

formule d'Aitken nécessite l'opérateur de propagation d'erreur noté $P$, qui décrit l'évolution de l'erreur de la solution restreinte aux interfaces artificielles : $e^{n+1} = Pe^n$. Cet opérateur peut parfois être calculé de manière analytique. Il peut également être calculé de manière algébrique, après $n_\Gamma + 1$ itérations de Schwarz, où $n_\Gamma$ est le nombre de points sur des interfaces artificielles. Cette méthode est donc très efficace pour résoudre des problèmes 1D. En dimension supérieure, le nombre de points des interfaces artificielles devient trop grand pour envisager l'accélération d'Aitken exacte. Dans ce cas, il est toujours possible d'effectuer l'accélération dans un espace de petite dimension qui pourra être construit à partir de quelques itérations de Schwarz. Plus précisément, l'accélération sera construite à partir des traces du processus de Schwarz, c'est-à-dire de l'ensemble des solutions restreintes aux interfaces. C'est ce qui a été proposé dans [52] où l'accélération est effectuée dans l'espace de Fourier. Une version purement algébrique de la méthode d'Aitken-Schwarz ($A-S$) a été proposée dans [118]. L'idée consistait à effectuer l'accélération dans un espace représentatif des traces. Une base de cet espace est obtenue par la décomposition en valeurs singulières des traces. Nous avons proposé une première implémentation (Fortran90/MPI) massivement parallèle de cette méthode dans [11]. Une modification de cet algorithme a été proposée dans [8] permettant de calculer la décomposition en valeurs singulières au fur et à mesure des itérations de Schwarz. Cette méthode a pour avantage de réduire significativement les besoins en mémoire de la méthode. L'accélération d'Aitken nécessite une approximation de l'opérateur de transfert d'erreur. On utilisera cette approximation pour préconditionner le processus de Richardson après la première accélération. On montrera qu'utiliser ce préconditionneur qui est très similaire à un préconditionneur basé sur la déflation, est équivalent à effectuer une accélération d'Aitken entre chaque itération de Schwarz. Afin d'améliorer la qualité de l'accélération, nous proposerons également d'exploiter la structure creuse de la matrice $P$. Cette nouvelle méthode, appelée $SA-S$ pour *Sparse Aitken-Schwarz*, s'avère bien plus extensible que $A-S$ car elle est plus robuste à l'augmentation du nombre de sous-domaines. Nous montrerons également que l'utilisation de $SA-S$ revient à effectuer l'accélération dans un espace plus grand que l'espace de Krylov engendré jusqu'ici par les itérations de Schwarz. La qualité de l'accélération sera grandement améliorée, et son coût sera diminué puisque les décompositions en valeurs singulières deviendront locales aux interfaces. Cette méthode sera testée dans la dernière partie de ce chapitre sur un problème d'écoulement dans un milieu poreux fortement hétérogène. On présentera une implémentation PETSc de cette méthode qui utilise deux niveaux de parallélisme.

Lorsque l'on résout un problème non linéaire transitoire par une méthode implicite, on utilise en général une méthode de type Newton pour résoudre les problèmes linéaires de chaque pas de temps. Chaque itération de la méthode de Newton est elle-même constituée de la résolution du problème linéarisé. Une partie très importante des temps de calcul sera donc composée de la résolution des systèmes linéaires de la forme $J(x)\Delta x = -F(x)$, où $J$ est la matrice jacobienne. Le chapitre 4 sera consacré à la résolution de cette suite de sys-

tèmes linéaires. En général, la matrice jacobienne ne changera pas de manière radicale d'un système linéaire à l'autre. Afin d'économiser des calculs, il est donc envisageable d'utiliser la même matrice de préconditionnement d'un système linéaire à l'autre, plutôt que de la recalculer. On pourra ainsi garder le même préconditionneur jusqu'à ce qu'il perde sa capacité à réduire le nombre d'itérations de Krylov. Dans ce chapitre 4, on étudiera la mise à jour du préconditionneur. L'idée consiste à améliorer le préconditionneur existant plutôt que de le recalculer. On se concentrera sur le préconditionneur *Restricted Additive Schwarz* (RAS), qui comme son nom l'indique, est basé sur une méthode de décomposition de domaine. Deux méthodes seront proposées dans ce chapitre. On appliquera d'abord la mise à jour de quasi-Newton au préconditionneur RAS [14]. L'implémentation de cette méthode est simple, et peut permettre de réduire modérément les temps de calcul. Nous proposerons également de recalculer partiellement ce préconditionneur RAS. Ce recalcul partiel consiste à mettre à jour uniquement les parties du préconditionneur associées à certains sous-domaines, tout en gardant les autres parties constantes. Ce nouveau préconditionneur sera nommé *Asynchronous Restricted Additive Schwarz* (AsRAS). On proposera une implémentation client-serveur de ce préconditionneur. C'est-à-dire que certains processeurs, les clients, auront en charge un sous-domaine. D'autres processeurs, les serveurs, calculeront uniquement les mises à jour du préconditionneur. Résoudre un problème de taille fixe avec le préconditionneur RAS en utilisant de plus en plus de processeurs, revient à multiplier le nombre de sous-domaines, et donc à diminuer l'efficacité numérique du préconditionneur. Il peut donc être avantageux d'utiliser les processeurs additionnels différemment. Finalement, c'est ce que permet le préconditionneur AsRAS. Cette méthode a été testée sur un problème de réaction-diffusion dans [15] et sur le problème de la cavité entraînée dans [12]. Ces tests mettront en évidence l'efficacité de la méthode, qui permet d'obtenir des accélérations superlinéaires, c'est-à-dire que l'ajout de $20\%$ de processeurs de type serveur peut permettre de réduire les temps de calcul d'un facteur supérieur à $1.2$. Le préconditionneur AsRAS permet donc d'exploiter efficacement plus de processeurs pour résoudre un problème.

Lorsque la décomposition du domaine spatial n'est pas envisageable, ou qu'elle n'est plus efficace, il est possible de décomposer le domaine temporel. Dans le dernier chapitre, nous proposons deux applications de la parallélisation en temps. Nous considérerons ici des problèmes non linéaires, et un domaine temporel sera réduit à un pas de temps. Le but est de commencer la résolution du problème non linéaire au temps $t^{i+1}$ alors que la résolution du problème au temps $t^i$ n'est pas terminée. Cette méthode, dite du pipelinage des pas de temps, sera d'abord étudiée dans le cas où la résolution du problème non linéaire s'effectue par une méthode de quasi-Newton [13]. Dans ce cas, il sera possible de propager les mises à jour de quasi-Newton d'un pas de temps à l'autre. Dans la deuxième partie de ce dernier chapitre, on appliquera l'idée du pipelinage des pas de temps à une méthode d'initialisation des équations différentielles algébriques. En effet, le calcul d'une solution initiale consistante satisfaisant les contraintes algébriques n'est pas naturel dans le cas des EDA. Il existe des mé-

thodes numériques itératives pour la recherche d'une condition initiale consistante où chaque itération consiste à effectuer un petit nombre de pas de temps. Après avoir présenté le problème particulier de l'initialisation des EDA, l'algorithme du pipelinage des pas de temps y sera adapté. On s'attend à ce que l'efficacité de la parallélisation en temps soit plus faible que celle de la parallélisation en espace. Intuitivement cela s'explique par la nature séquentielle des EDO/EDA. Des résultats numériques sur le problème de Fekete viendront montrer qu'il est tout de même possible de diminuer les temps de calcul de manière significative. On pourra par exemple diminuer le temps de calcul d'environ $35\%$ en utilisant deux processeurs.

# Chapitre 2

# Résolution des systèmes linéaires creux

## Sommaire

Ce chapitre présente un état de l'art de la résolution des systèmes linéaires creux, en se focalisant sur les notions et méthodes qui seront utilisées dans les chapitres suivants, et en particulier dans les expérimentations numériques.

Il existe deux grandes familles de méthodes pour la résolution des systèmes linéaires creux : les méthodes directes, basées sur la factorisation de l'opérateur linéaire, et les méthodes itératives. Les méthodes directes ont l'avantage d'être très robustes, et l'inconvénient d'être, en général, plus coûteuses et moins extensibles que les méthodes itératives. Le coût en mémoire des factorisations LU peut cependant être réduit en renumérotant les inconnues pour améliorer le profil de la matrice [117]. Il est également possible d'optimiser les communications [38].

Dans la première section de ce chapitre, on rappellera les principales caractéristiques des méthodes de Krylov, qui sont les méthodes itératives les plus largement utilisées. On introduira les espaces de Krylov à partir de la méthode du gradient conjugué et on présentera les méthodes BiCGSTAB et GMRES qui seront utilisées dans les tests numériques des chapitres suivants.

La deuxième section sera consacrée aux méthodes de décomposition de domaine de type Schwarz, qui permettent de décomposer un problème en sous-problèmes. La solution du problème global sera obtenue de manière itérative, où chaque itération sera composée de la résolution de sous-problèmes et de l'échange des conditions aux limites. La méthode de décomposition de domaine sera présentée d'abord d'un point de vue analytique, puis vue comme des itérations de Richardson. Dans la deuxième partie, on discutera de l'accélération d'Aitken appliquée à ce processus de Richardson.

## 2.1 Méthodes de Krylov

### 2.1.1 Méthode du gradient conjugué

La construction de la méthode du gradient conjugué est largement détaillée dans [69, chapitre 11] et [111]. Considérons la résolution du système linéaire $Ax = b$ avec $x, b \in \mathbb{R}^n$ et $A \in \mathbb{R}^{n\times n}$. La fonction $\phi(x)$ de l'équation (2.1) sera minimale pour $x = A^{-1}b$.

$$\phi(x) = \frac{1}{2}x^T Ax - x^T b, \quad x \in \mathbb{R}^n. \tag{2.1}$$

Notons $x_0$ la première approximation de $x$, dans ce cas la fonction $\phi$ décroît le plus rapidement dans la direction $-\nabla\phi(x_0) = r_0$ avec $r_0 = b - Ax_0$ le résidu. La méthode de la plus rapide descente consiste à effectuer les itérations $x_{k+1} = x_k + \alpha_k r_{k-1}$ avec $\alpha_k$ qui minimise $\phi(x + \alpha r_0)$. La solution du système linéaire est donc obtenue en minimisant $\phi$ le long des directions $\{r_0, r_1, \cdots\}$.

Dans la méthode du gradient conjugué, les directions $\{p_1, p_2 \cdots\}$ sont choisies différemment. En effet, lorsque les directions sont linéairement indépendantes, la convergence est

assurée en au plus $n$ itérations. Dans la méthode du gradient conjugué, on va donc chercher à construire la direction $p_k$ qui sera la direction la plus proche de $r_{k-1}$ tout en étant A-conjuguée à la $k-1$ème direction (c'est-à-dire que $p_k^T A p_{k-1} = 0$). L'algorithme du gradient conjugué est donné en algorithme 1. On peut montrer que si $x_0 = 0$, alors la solution $x_k$ est

**Algorithme 1** Gradient conjugué

**Require:** an initial guess $x_0$
**Require:** a tolerance $\epsilon$
$r_0 \leftarrow b - Ax_0$
$p_0 \leftarrow r_0$
**for** $k = 0 \ldots$ **do**
  $\alpha_k \leftarrow \dfrac{r_k^T r_k}{p_k^T A p_k}$
  $x_{k+1} \leftarrow x_k + \alpha_k p_k$
  $r_{k+1} \leftarrow r_k - \alpha_k A p_k$
  $\beta_k \leftarrow \dfrac{r_{k+1}^T r_{k+1}}{r_k^T r_k}$
  $p_{k+1} \leftarrow r_{k-1} + \beta_k p_k$
  **if** $\|r_{k+1}\| < \epsilon$ **then** exit the loop **end if**
**end for**

dans l'espace de Krylov noté $\mathcal{K}^k(r_0, A)$ de l'équation (2.2) [69, chapitre 10].

$$\mathcal{K}^k(r_0, A) = span\{r_0, Ar_0, \cdots, A^{k-1} r_0\} \tag{2.2}$$

La convergence du gradient conjugué n'est assurée que pour une matrice $A$ symétrique définie positive. Ainsi, si l'on note $\kappa(A) = \|A\|_2 \|A^{-1}\|_2$ le conditionnement de la matrice $A$, et $e_k = x^* - x_k$ l'erreur (c'est-à-dire la différence entre la solution exacte et la solution à l'itération $k$ $x_k$), on a [111, chapitre 6.11.3] :

$$\|e_k\|_A < 2\|e_0\|_A \left( \frac{\sqrt{\kappa(A)} - 1}{\sqrt{\kappa(A)} + 1} \right)^k, \tag{2.3}$$

où $\|e_k\|_A^2 = e_k^T A e_k$. Notons que le conditionnement de la matrice $A$ s'écrit également en fonction des valeurs singulières : $\kappa(A) = \sigma_{max}/\sigma_{min}$. Dans le cas où la matrice $A$ est symétrique, comme ici, les valeurs singulières sont les valeurs absolues des valeurs propres.

### 2.1.2 Le gradient bi-conjugué stabilisé

Dans le cas où l'opérateur n'est pas symétrique, il est possible d'utiliser la méthode du gradient bi-conjugué (BiCG) qui consiste à résoudre le problème augmenté suivant :

$$\begin{bmatrix} A & 0 \\ 0 & A^T \end{bmatrix} \begin{bmatrix} x \\ \widetilde{x} \end{bmatrix} = \begin{bmatrix} b \\ \widetilde{b} \end{bmatrix}. \tag{2.4}$$

Dans le cas où la matrice $A$ est symétrique, le BiCG fournira la même suite d'itérés $x_k$ que le gradient conjugué (CG) mais demandera deux fois plus de calculs. La méthode du BiCG est peu utilisée en pratique car la convergence peut être très irrégulière : il peut y avoir de nombreux pics sur la courbe du résidu. Ceci vient du fait que le BiCG ne minimise pas le résidu. Une version stabilisée, mise au point par VAN DER VORST dans [122], est donnée en algorithme 2. Cette méthode du gradient bi-conjugué stabilisé (BiCGStab) sera utilisée dans certaines des expériences numériques de ce manuscrit. L'étape de la stabilisation peut être vue comme une itération de la méthode GMRES.

**Algorithme 2** BiCGStab

**Require:** an initial guess $x_0$
**Require:** a tolerance $\epsilon$
$r_0 \leftarrow b - Ax_0$
$r_0^* \leftarrow r_0$
$p_0 \leftarrow r_0$
**for** $k = 0 \ldots$ **do**
  $\alpha_k \leftarrow \dfrac{(r_k, r_0^*)}{(Ap_k, r_0^*)}$
  $s_k \leftarrow r_k - \alpha_k A p_k$
  $\omega_k \leftarrow \dfrac{(As_k, s_k)}{(As_k, As_k)}$
  $x_{k+1} \leftarrow x_k + \alpha_k p_k + \omega_k s_k$
  $r_{k+1} \leftarrow s_k - \omega_k A s_k$
  $\beta_k \leftarrow \dfrac{(r_{k+1}, r_0^*)}{(r_k, r_0^*)} \times \dfrac{\alpha_k}{\omega_k}$
  $p_{k+1} \leftarrow r_{k+1} + \beta_k (p_k - \omega_k A p_k)$
  **if** $\|r_{k+1}\| < \epsilon$ **then** exit the loop **end if**
**end for**

### 2.1.3 GMRES

La méthode GMRES [110] permet de minimiser le résidu à chaque itération. Ainsi, le vecteur $x_k$ de $x_0 + \mathcal{K}^k(r_0, A)$ est celui qui minimise le résidu $\|b - Ax_k\|_2$. Le vecteur $x_k$ de $x_0 + \mathcal{K}^k(r_0, A)$ peut s'écrire comme

$$x_k = x_0 + V_k y \tag{2.5}$$

où $V_k$ est la matrice formée des vecteurs orthonormés de la base de $\mathcal{K}^k(r_0, A)$. Ces vecteurs sont obtenus par un processus d'Arnoldi. Une implémentation possible de GMRES est donnée en algorithme 3. GMRES demande de stocker $V_k$, ce qui peut être coûteux. En pratique, une taille maximale pour $V_k$ sera fixée. Lorsque cette taille sera atteinte, $V_k$ sera supprimée et l'algorithme de GMRES sera relancé en utilisant $x_k$ comme nouvelle solution initiale. Cette façon de limiter l'augmentation de la mémoire s'appelle *restarted GMRES*, et il s'agit de la

**Algorithme 3** GMRES

**Require:** an initial guess : $x_0$, a maxium number of iterations $m$

$r_0 \leftarrow b - Ax_0$, $\beta \leftarrow \|r_0\|_2$, $v_1 \leftarrow r_0/\beta$
$\bar{H} \leftarrow \mathbb{R}^{(m+1)\times m}$, $h_{ij} = 0\ \forall (i,j) \in [\![1, m+1]\!] \times [\![1, m]\!]$
**for** $i = 1, 2, \ldots, m$ **do**
  $\omega_i \leftarrow Av_i$
  **for** $i = 1, 2, \ldots, j$ **do**
    $h_{ij} \leftarrow (\omega_j, v_i)$
    $\omega_j \leftarrow \omega_j - h_{ij}v_i$
  **end for**
  $h_{j+1,j} = \|\omega_j\|_2$
  **if** $h_{j+1,j} = 0$ **then** $m \leftarrow j$, goto 13 **end if**
  $v_{j+1} = \omega_j / h_{j+1,j}$
**end for**
$y \leftarrow argmin_y \|\beta e_1 - \bar{H}_m y\|_2$
$x_m \leftarrow x_0 + V_m y_m$

méthode de Krylov par défaut de la bibliothèque PETSc [5]. Il est possible d'améliorer la convergence de la méthode *restarted GMRES* en exploitant une partie de l'information contenue dans $V_k$ après son écrasement en mémoire. C'est le principe des méthodes de déflation : l'espace de Krylov après le *restart* peut être augmenté [31], ou bien un préconditionneur peut être construit [49] à partir des informations obtenues avant le *restart*.

### 2.1.4 Le préconditionnement

#### 2.1.4.1 Principe

Afin d'accélérer la convergence des méthodes de Krylov, il est possible de préconditionner le système linéaire. La résolution du système linéaire $Ax = b$ est donc remplacée soit par

$$M^{-1}Ax = M^{-1}b \tag{2.6}$$

dans le cas un système préconditionné à gauche, soit par

$$\begin{cases} AM^{-1}y = b \\ Mx = y \end{cases} \tag{2.7}$$

si le système est préconditionné à droite. Le préconditionneur $M$ doit approcher la matrice $A$, mais son inverse doit être facile à calculer. Un préconditionneur doit réduire le conditionnement du système : $\kappa(M^{-1}A) << \kappa(A)$ pour un préconditionnement gauche. En pratique, il n'y a en général pas besoin de construire la matrice $M$, ni son inverse, car seul le produit matrice-vecteur est nécessaire aux méthodes de Krylov. En général, le produit matrice-vecteur

$w = M^{-1}v$ sera obtenu par la résolution (éventuellement grossière) du système $Mw = v$. Il est donc possible de préconditionner un système linéaire en utilisant une autre méthode de résolution que la méthode de Krylov. Ainsi, les méthodes de décomposition de domaine, les méthodes multigrilles [73], ou de factorisations incomplètes peuvent être utilisées pour préconditionner les itérations d'une méthode de Krylov.

#### 2.1.4.2 Méthodes de factorisation incomplète

La résolution d'un problème par une méthode de factorisation LU de l'opérateur $A$ peut être coûteuse en mémoire dans la mesure où le nombre d'éléments non nuls dans les facteurs $L$ et $U$ peut être très supérieur au nombre d'éléments non nuls de $A$. Les méthodes de factorisation LU incomplète (ILU) permettent de contrôler le niveau de remplissage. Ainsi, la méthode $ILU(0)$ consiste à calculer la factorisation $A \approx L_0U_0$ avec $L_0$ et $U_0$ ayant des coefficients non nuls uniquement là où $A$ a des coefficients non nuls. On pourra autoriser un certain remplissage, ainsi on peut définir la factorisation $ILU(p)$ $A \approx L_pU_p$ où $L_p$ et $U_p$ ont des coefficients non nuls uniquement là où $L_{p-1} \times U_{p-1}$ en possède. En pratique, il n'est pas nécessaire de calculer la factorisation $ILU(p-1)$ pour obtenir la factorisation $ILU(p)$.

#### 2.1.4.3 Méthodes multigrilles

Lorsque la matrice $A$ est issue de la discrétisation d'équations aux dérivées partielles, il est possible de construire un maillage grossier. Dans ce cas, l'application du préconditionneur sera composée d'une projection de la solution sur la grille grossière, de la résolution du problème grossier, et d'un lissage pour revenir à la grille fine. En général, il y aura plusieurs niveaux de grilles. Les méthodes multigrilles peuvent également être considérées d'un point de vue purement algébrique. Dans ce cas, la construction d'une grille grossière pourra consister à agréger des inconnues. Les logiciels AGMG [100] et Hypre [50] sont utilisés dans le chapitre 3 de ce manuscrit.

## 2.2 Décomposition de domaine de Schwarz

La décomposition de domaine de Schwarz a été initialement introduite dans [76] au niveau continu. Sa version algébrique est aujourd'hui très utilisée pour résoudre des problèmes linéaires de la forme $Ax = b$, car elle est très adaptée au calcul parallèle. En effet, elle consiste à découper le problème global à tout le domaine en sous-problèmes locaux aux sous-domaines. Cette décomposition de domaine fait apparaître des limites artificielles à l'interface des sous-domaines. La méthode de Schwarz sera donc composée de la résolution des problèmes locaux, et de la mise à jour des conditions aux limites artificielles. D'un point de vue pratique, les communications seront donc locales, c'est-à-dire entre sous-domaines voisins. Le principal inconvénient de la méthode de décomposition de domaine de Schwarz est

que la convergence peut être extrêmement lente. La convergence dépend du problème, de la géométrie des sous-domaines et de la taille du recouvrement éventuel. Intuitivement, ce faible taux de convergence peut justement être expliqué par le fait que la méthode de Schwarz est composée de résolutions locales et d'échanges locaux : d'un point de vue spectral, la méthode de Schwarz influera surtout sur les valeurs propres associées aux modes rapides, et peu sur celles associées aux modes lents. Il existe de nombreuses méthodes d'accélération de la convergence, elles sont en général basées sur la résolution d'un problème commun à tous les sous-domaines. Il est donc naturel de considérer la résolution du problème restreint aux interfaces artificielles. En effet, la solution exacte sur les interfaces artificielles s'étend à tout le domaine à l'aide d'une résolution locale sur chaque sous-domaine. Il est cependant possible de relier les sous-domaines en utilisant une méthode multigrille classique. Dans ce manuscrit, on ne considèrera que le problème réduit aux interfaces artificielles.

On introduira d'abord la méthode de Schwarz dans le cas continu pour des opérateurs elliptiques. Cette méthode sera étendue par analogie à la résolution d'un problème algébrique $Ax = b$, sans supposer que l'opérateur $A$ est issu de la discrétisation d'opérateurs elliptiques.

### 2.2.1 Méthode de Schwarz au niveau continu

Considérons d'abord un problème elliptique sur un seul domaine de la forme suivante :

$$\left\{ \begin{array}{lcl} L(x)u(x) & = & f(x), \text{ pour } x \in \Omega, \\ \gamma_0 u(x) & = & g(x), \text{ pour } x \in \partial\Omega, \end{array} \right. \tag{2.8}$$

où $\Omega \subset \mathbb{R}^n$ est un domaine borné de frontière lipschitzienne $\partial\Omega$ et $L(x)$ est un opérateur uniformément elliptique de la forme

$$L(x)u(x) \quad = \quad -\sum_{i=1}^{n}\sum_{j=1}^{n} \frac{\partial}{\partial x_j}\left(a_{ji}(x)\frac{\partial}{\partial x_i}u(x)\right). \tag{2.9}$$

L'opérateur $\gamma_0$ est l'opérateur de trace. Le théorème 1 de trace issu de [115, p9] établit l'existence d'un tel opérateur de restriction aux conditions aux limites, mais aussi celle d'un opérateur qui étend la solution sur la trace à tout le domaine.

**Théorème 1.** *Soit $\Omega \subset \mathbb{R}^n$ un domaine borné de frontière lipschitzienne $\partial\Omega$. Pour tout $u$ de l'espace de Hilbert $H^1(\Omega)$ la trace $\gamma_0 u \in H^{1/2}(\partial\Omega)$ existe et satisfait :*

$$\|\gamma_0 u\|_{H^{1/2}(\partial\Omega)} \leq c.\|u\|_{H^1(\Omega)}$$

*Réciproquement, pour tout $u \in H^{1/2}(\partial\Omega)$ il existe une extension bornée $\mathcal{E}u \in H^1(\Omega)$ satisfaisant $\gamma_0\mathcal{E}u = u$ et*

$$\|\mathcal{E}u\|_{H^1(\Omega)} \leq c'\|u\|_{H^{1/2(\partial\Omega)}}.$$

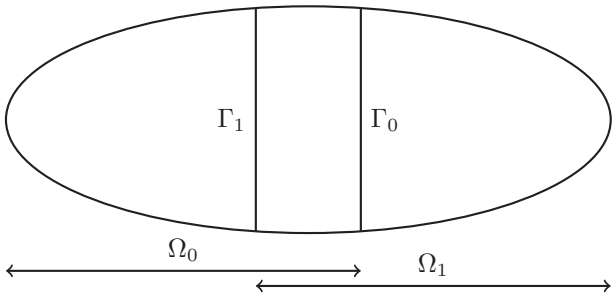


**FIGURE *2.1*** – *Découpage d'un domaine 2D avec recouvrement,* $\Gamma_0$ *et* $\Gamma_1$ *sont les frontières artificielles.*

L'existence et l'unicité de la solution $u \in H^1(\Omega)$ du problème (2.8) sont démontrées dans [115] en utilisant le théorème de Lax-Milgram généralisé [97].

Dans la suite, lorsqu'on considèrera plusieurs sous-domaines, le terme *trace* désignera uniquement la partie de la solution restreinte aux interfaces artificielles entre les sous-domaines. Comme les conditions aux limites physiques sont constantes d'une itération à l'autre de la méthode de décomposition de domaine, elles peuvent être incluses dans l'opération qui étend la solution des interfaces artificielles à tout le domaine.

#### 2.2.1.1 Méthode de Schwarz généralisée

On considère maintenant la méthode *Generalized Schwarz Alternating Method* (GSAM) introduite par ENGQUIST et ZHAO dans [48] qui permet de rassembler plusieurs techniques de Schwarz [106]. On considèrera uniquement le cas où le domaine $\Omega$ est découpé en deux sous-domaines (voir la figure 2.1) $\Omega_0$ et $\Omega_1$ qui peuvent se recouvrir. On note $\Gamma_0$ la frontière artificielle (c'est-à-dire celle qui n'est pas dans $\partial\Omega$) de $\Omega_0$, et $\Gamma_1$ celle de $\Omega_1$. On peut résumer le GSAM avec les deux algorithmes 4 et 5. La solution du problème global de la forme (2.8) s'obtient donc à partir de la résolution des problèmes restreints aux sous-domaines $\Omega_0$ et $\Omega_1$. Les conditions aux interfaces artificielles sont de type Robin : $\Lambda_0 u_0 + \lambda_0 \dfrac{\partial u_0}{n_0} = \mu_1$ sur $\Gamma_0$, et $\Lambda_1 u_1 + \lambda_1 \dfrac{\partial u_1}{n_1} = \mu_0$ sur $\Gamma_1$, avec $\Lambda_i$ des opérateurs et $\lambda_i$ des constantes. Ces conditions aux limites sont des conditions de transmission de la solution dans la mesure où $\mu_1$ est défini par la solution $u_0$ dans $\Omega_0$ et réciproquement pour $\mu_0$. La différence entre les versions additive et multiplicative est dans le moment de la mise à jour des conditions aux limites. Dans la version additive, les deux sous-problèmes sont résolus simultanément puis les conditions aux limites sont échangées. La version multiplicative consiste à résoudre d'abord le problème sur $\Omega_0$, puis la condition limite sur $\Gamma_1$ est mise à jour avant la résolution du problème sur $\Omega_1$.

La table 2.1 présente les principales familles de méthodes de Schwarz obtenues selon

**Algorithme 4** GSAM : version multiplicative

**Require:** $\Lambda_0$ and $\Lambda_1$ : two linear operators

**Require:** $\lambda_0, \lambda_1 \in \mathbb{R}$

1: $n \leftarrow 0$

2: **repeat**

3: Solve

$$\left\{ \begin{array}{rcl} L(x)u_0^{2n+1}(x) & = & f(x),\ \forall x \in \Omega_0, \\ u_0^{2n+1}(x) & = & g(x),\ \forall x \in \partial\Omega_0 \backslash \Gamma_0, \\ \Lambda_0 u_0^{2n+1} & + & \lambda_0 \dfrac{\partial u_0^{2n+1}(x)}{\partial n_0} = \Lambda_0 u_1^{2n} + \lambda_0 \dfrac{\partial u_1^{2n}(x)}{\partial n_0},\ \forall x \in \Gamma_0 \end{array} \right. \tag{2.10}$$

4: Solve

$$\left\{ \begin{array}{rcl} L(x)u_1^{2n+2}(x) & = & f(x),\ \forall x \in \Omega_1, \\ u_1^{2n+2}(x) & = & g(x),\ \forall x \in \partial\Omega_1 \backslash \Gamma_1, \\ \Lambda_1 u_1^{2n+2} & + & \lambda_1 \dfrac{\partial u_1^{2n+2}(x)}{\partial n_1} = \Lambda_1 u_0^{2n+1} + \lambda_1 \dfrac{\partial u_0^{2n+1}(x)}{\partial n_1},\ \forall x \in \Gamma_1. \end{array} \right. \tag{2.11}$$

5: $n \leftarrow n + 2$

6: **until** convergence

**Algorithme 5** GSAM : version additive

**Require:** $\Lambda_0$ and $\Lambda_1$ : two linear operators

**Require:** $\lambda_0, \lambda_1 \in \mathbb{R}$

1: $n \leftarrow 0$

2: **repeat**

3: Solve the two problems

$$\left\{ \begin{array}{rcl} L(x)u_0^n(x) & = & f(x),\ \forall x \in \Omega_0, \\ u_0^n(x) & = & g(x),\ \forall x\ \in \partial\Omega_0 \backslash \Gamma_0, \\ \Lambda_0 u_0^n & + & \lambda_0 \dfrac{\partial u_0^n(x)}{\partial n_0} = \Lambda_0 u_1^{n-1} + \lambda_0 \dfrac{\partial u_1^{n-1}(x)}{\partial n_0},\ \forall x \in \Gamma_0 \end{array} \right. \tag{2.12}$$

and

$$\left\{ \begin{array}{rcl} L(x)u_1^n(x) & = & f(x),\ \forall x \in \Omega_1, \\ u_1^n(x) & = & g(x),\ \forall x \in \partial\Omega_1 \backslash \Gamma_1, \\ \Lambda_1 u_1^n & + & \lambda_1 \dfrac{\partial u_1^n(x)}{\partial n_1} = \Lambda_1 u_0^{n-1} + \lambda_1 \dfrac{\partial u_0^{n-1}(x)}{\partial n_1},\ \forall\, x \in \Gamma_1. \end{array} \right. \tag{2.13}$$

4: $n \leftarrow n + 1$

5: **until** convergence

les choix des paramètres $\lambda_i$ et $\Lambda_i$. Ainsi, la méthode de Schwarz originelle est donnée par $\Lambda_0 = \Lambda_1 = I$ et $\lambda_0 = \lambda_1 = 0$ : des conditions de Dirichlet sont imposées aux interfaces artificielles. La version modifiée de la méthode de Schwarz proposée par LIONS dans [89] est obtenue en choisissant $\Lambda_0 = \Lambda_1$ une constante positive et $\lambda_0 = \lambda_1 = 1$.

**TABLE *2.1 –*** *Principales familles de méthodes de Schwarz obtenues à partir de GSAM selon les valeurs de $\Lambda_i$ et $\lambda_i$*

| Recouvrement | $\Lambda_0$ | $\Lambda_1$ | $\lambda_0$ | $\lambda_1$ | Méthode |
|---|---|---|---|---|---|
| avec | $Id$ | $Id$ | 0 | 0 | Schwarz |
| avec | $Id$ | $Id$ | $\alpha$ | $\alpha$ | ORAS [37] |
| sans | $Id$ | 0 | 0 | 1 | Neumann-Dirichlet [93] |
| sans | $Id$ | $Id$ | 1 | 1 | Schwarz modifié [88] |

Afin d'illustrer les différences entre les méthodes additive et multiplicative, on considère un problème modèle 1D, découpé en deux sous-domaines, pour lequel la solution est l'interpolation linéaire entre les conditions aux limites de Dirichlet. Les deux premières itérations sont présentées en figure 2.2.

Les deux méthodes sont différentes du point de vue de leur implémentation parallèle :

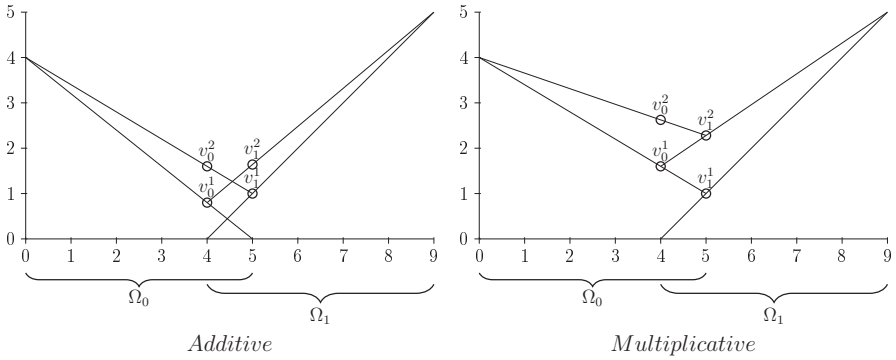


**FIGURE 2.2 –** *Deux premiers itérés des méthodes de Schwarz additive et multiplicative,* $\Lambda_0 = \Lambda_1 = I$ *et* $\lambda_0 = \lambda_1 = 0$*, c'est-à-dire la version Dirichlet-Dirichlet.*

- Lors d'une itération de la méthode de Schwarz additive, tous les sous-problèmes sont résolus simultanément avant l'échange des conditions aux limites. Ainsi, il est possible d'occuper tous les processeurs en même temps.
- Dans le cas de deux sous-domaines, une itération de Schwarz multiplicatif est composée de deux résolutions successives (et donc séquentielles). En pratique, la version multiplicative sera également parallèle dans la mesure où il est possible de résoudre simultanément les sous-problèmes associés à des sous-domaines disjoints. Par exemple, il est possible d'utiliser un coloriage rouge-noir pour un problème 1D. Dans ce cas, une itération sera composée d'une résolution parallèle sur les domaines rouges, puis d'une résolution parallèle sur les domaines noirs. En général (dans le cas où il n'y a qu'un seul second membre à traiter, et où un processeur ne gère qu'un seul sous-domaine) seule la moitié des processeurs est occupée à un moment donné.

Remarquons également que la méthode de Schwarz multiplicative converge exactement deux fois plus rapidement que la méthode additive sur les illustrations.

ENGQUIST et ZHAO [48] ont montré qu'avec un choix approprié des opérateurs $\Lambda_i$, la méthode converge en deux itérations. Ils ont établi la proposition qui suit :

**Proposition 1.** *Si* $\Lambda_0$ *(ou* $\Lambda_1$*) est l'opérateur de correspondance Dirichlet-Neumann sur l'interface artificielle* $\Gamma_0$ *(ou* $\Gamma_1$*) correspondant à l'équation aux dérivées partielles homogène sur* $\Omega_1$ *(ou* $\Omega_0$*) avec des conditions aux limites homogènes sur* $\partial\Omega_1 \cap \partial\Omega$ *(ou* $\partial\Omega_0 \cap \partial\Omega$*), alors GSAM converge en deux itérations.*

La méthode GSAM converge donc en deux itérations si les opérateurs de correspondance Dirichlet-Neumann $\Lambda_i$, $i = 0, 1$, sont disponibles. Afin d'accélérer la convergence de la méthode de Schwarz, on peut chercher à estimer l'opérateur de correspondance Dirichlet-Neumann : pour certains problèmes, une approximation de cet opérateur peut être obte-

nue de manière analytique ou algébrique [92]. La convergence peut également être accélérée en calculant uniquement les plus petites valeurs propres et les vecteurs associés [95]. La méthodologie Aitken-Schwarz, introduite par [57, 56] et qui sera reprise dans le chapitre suivant, ne consistera pas à approcher directement cet opérateur de correspondance Dirichlet-Neumann, mais à approcher l'opérateur de propagation de l'erreur lié à cet opérateur Dirichlet-Neumann.

On considère toujours un domaine divisé en deux sous-domaines. Pour montrer que la convergence est linéaire à l'interface, il faut d'abord introduire les notations suivantes :

- Soit $R^j_{\Gamma_i} : H^1(\Omega_j) \rightarrow H^{1/2}(\Gamma_i)$ l'opérateur qui restreint la solution du domaine $\Omega_j$ à l'interface $\Gamma_i$.
- Soit $R_i^* : H^{1/2}(\Gamma_i) \rightarrow H^1(\Omega_i)$ l'opérateur de prolongation tel que $R_i^* g$ est la solution du problème :

$$\left\{ \begin{array}{rcl} L(x)R_i^* g & = & 0,\ \forall x \in \Omega_i, \\ R_i^* g & = & 0,\ \forall x \in \partial\Omega_i \backslash \Gamma_i,\ . \\ R_i^* g & = & g,\ \forall x \in \Gamma_i \end{array} \right. \tag{2.14}$$

  L'opérateur $R_i^*$ est donc l'inverse droit de $R^i_{\Gamma_i}$ puisque $R^i_{\Gamma_i} R_i^* = I$.
- Soit $S_{ij}$ l'opérateur de correspondance Dirichlet-Neumann associé au domaine $\Omega_i$ sur $\Gamma_j$.

Dans le cas de la méthode multiplicative, les erreurs $e_0^{2k+1} = u_0^{2k+1} - u_0$ et $e_1^{2k+2} = u_1^{2k+2} - u_1$ satisfont :

$$\left\{ \begin{array}{rcl} L(x)e_0^{2k+1} & = & 0, \forall x \in \Omega_0, \\ e_0^{2k+1} & = & 0, \forall x \in \partial\Omega_0 \backslash \Gamma_0 \\ \Lambda_0 e_0^{2k+1} & + & \lambda_0 \dfrac{\partial e_0^{2k+1}}{\partial n_0} = \Lambda_0 e_1^{2k} + \lambda_0 \dfrac{\partial e_1^{2k}}{\partial n_0}, \forall x \in \Gamma_0 \end{array} \right.$$

et

$$\left\{ \begin{array}{rcl} L(x)e_1^{2k+2} & = & 0, \forall x \in \Omega_1, \\ e_1^{2k+2} & = & 0, \forall x \in \partial\Omega_1 \backslash \Gamma_1 \\ \Lambda_1 e_1^{2k+2} & + & \lambda_1 \dfrac{\partial e_1^{2k+2}}{\partial n_1} = \Lambda_1 e_0^{2k+1} + \lambda_1 \dfrac{\partial e_0^{2k+1}}{\partial n_1}, \forall x \in \Gamma_1. \end{array} \right.$$

Les erreurs aux interfaces s'écrivent donc :

$$\begin{array}{rcl} (\Lambda_0 + \lambda_0 S_{00}) R^0_{\Gamma_0} e_0^{2k+1} & = & (\Lambda_0 + \lambda_0 S_{01}) R^1_{\Gamma_0} e_1^{2k} \\ (\Lambda_1 + \lambda_1 S_{11}) R^1_{\Gamma_1} e_1^{2k+2} & = & (\Lambda_1 + \lambda_1 S_{10}) R^0_{\Gamma_1} e_0^{2k+1}. \end{array}$$

Par conséquent :

$$\begin{array}{rcl} e_1^{2k+2} & = & R_1^*(\Lambda_1 + \lambda_1 S_1)^{-1}(\Lambda_1 + \lambda_1 S_{11}) R^0_{\Gamma_1} R_0^* (\Lambda_0 + \lambda_0 S_{00})^{-1}(\Lambda_0 + \lambda_0 S_{11}) R^0_{\Gamma_1} e_1^{2k} \\ e_0^{2k+1} & = & R_0^*(\Lambda_0 + \lambda_0 S_{00})^{-1}(\Lambda_0 + \lambda_0 S_{11}) R^1_{\Gamma_0} R_1^* (\Lambda_1 + \lambda_1 S_{11})^{-1}(\Lambda_1 + \lambda_1 S_{00}) R^1_{\Gamma_0} e_0^{2k-1}. \end{array}$$

En particulier, pour la méthode de Schwarz originelle, on a :

$$e_i^k = \underbrace{R_i^* R_{\Gamma_i}^j}_{p_j} \underbrace{R_j^* R_{\Gamma_j}^i}_{p_i} (e_i^{k-1}), (i,j) \in \{(0,1),(1,0)\}.$$

La convergence sera donc dite purement linéaire dans la mesure où l'erreur à l'itération $k$ s'écrit comme le produit d'un opérateur linéaire constant avec l'erreur à l'itération $k-1$. Il en est de même pour la méthode additive, où l'on peut écrire :

$$\begin{bmatrix} e_0^{k+1} \\ e_1^{k+1} \end{bmatrix} = \begin{bmatrix} 0 & p_1 \\ p_0 & 0 \end{bmatrix} \begin{bmatrix} e_0^k \\ e_1^k \end{bmatrix}.$$

#### 2.2.1.2 Lien entre l'opérateur Dirichlet-Neumann au niveau continu et le complément de Schur au niveau discret

Si l'on considère le cas d'un découpage en deux sous-domaines sans recouvrement, alors la résolution du problème (2.8) discrétisé conduit à la résolution du système linéaire suivant :

$$Au = \begin{bmatrix} A_{00} & 0 & A_{0\Gamma} \\ 0 & A_{11} & A_{1\Gamma} \\ A_{\Gamma 0} & A_{\Gamma 1} & A_{\Gamma\Gamma}^{(0)} + A_{\Gamma\Gamma}^{(1)} \end{bmatrix} \begin{bmatrix} u_0 \\ u_1 \\ u_\Gamma \end{bmatrix} = \begin{bmatrix} f_0 \\ f_1 \\ f_\Gamma^{(0)} + f_\Gamma^{(1)} \end{bmatrix} \tag{2.15}$$

où $u_0$ et $u_1$ représentent les vecteurs des inconnues internes aux sous-domaines $\Omega_0$ et $\Omega_1$, et $u_\Gamma$ représente le vecteur des inconnues sur $\Gamma$. Les inconnues internes peuvent être formellement éliminées de l'équation (2.15) en écrivant :

$$\begin{aligned} u_0 &= A_{00}^{-1}(f_0 - A_{0\Gamma} u_\Gamma), \\ u_1 &= A_{11}^{-1}(f_1 - A_{1\Gamma} u_\Gamma), \end{aligned} \tag{2.16}$$

et en notant

$$\begin{aligned} S_0 &= A_{\Gamma\Gamma}^{(0)} - A_{\Gamma 0} A_{00}^{-1} A_{0\Gamma},\ g_0 = f_\Gamma^{(0)} - A_{\Gamma 0} A_{00}^{-1} f_0, \\ S_1 &= A_{\Gamma\Gamma}^{(1)} - A_{\Gamma 1} A_{11}^{-1} A_{1\Gamma}, g_1 = f_\Gamma^{(1)} - A_{\Gamma 1} A_{11}^{-1} f_1, \end{aligned} \tag{2.17}$$

on obtient le système du complément de Schur défini sur l'interface :

$$S u_\Gamma = (S_0 + S_1) u_\Gamma = g_0 + g_1 = g. \tag{2.18}$$

Si on considère le problème sur $\Omega_0$, alors on a l'identité suivante (voir [96]) :

$$\begin{bmatrix} A_{00} & A_{0\Gamma} \\ A_{\Gamma 0} & A_{\Gamma\Gamma}^{(0)} \end{bmatrix}^{-1} \begin{bmatrix} 0 \\ S_0 u_\Gamma \end{bmatrix} = \begin{bmatrix} A_{00} & A_{0\Gamma} \\ 0 & I \end{bmatrix}^{-1} \begin{bmatrix} 0 \\ u_\Gamma \end{bmatrix} \tag{2.19}$$

où le membre de gauche correspond à la résolution du problème sur $\Omega_0$ avec des conditions aux limites de type Neumann sur $\Gamma$, et le membre de droite correspond à la résolution du problème avec des conditions de Dirichlet sur $\Gamma$. Cela permet donc de montrer le lien entre le complément de Schur et l'opérateur de correspondance Dirichlet-Neumann. Il est donc possible d'utiliser une approximation algébrique de ces compléments de Schur locaux aux sous-domaines pour accélérer la convergence de la méthode de Schwarz (voir [32, 55, 66]).

### 2.2.2 Convergence strictement linéaire au niveau discret

On montrera le résultat classique de la convergence purement linéaire de la méthode de Schwarz additive algébrique. Cela consistera à réécrire les itérations de Schwarz comme un processus de Richardson. On montrera également que la solution restreinte aux interfaces artificielles converge de manière purement linéaire.

#### 2.2.2.1 Notations

Un domaine comptant $n$ inconnues est divisé en $N$ sous-domaines qui se recouvrent. Cette partition des inconnues peut être faite à partir de la connaissance des équations de la physique et de leur discrétisation, en séparant les sous-problèmes qui dépendent peu les uns des autres. Cependant, en général on partitionnera les inconnues de manière algébrique, c'est-à-dire en utilisant uniquement la matrice, avec comme objectif de limiter les dépendances de données entre les sous-domaines et d'équilibrer la charge de travail entre les sous-domaines. Le $i$ème sous-domaine est donc composé de $n_i$ inconnues si on compte le recouvrement, ou de $\widetilde{n}_i$ inconnues si on exclut le recouvrement. Dans ce cas, on a bien $n = \Sigma_{i=0}^{N-1}\widetilde{n}_i$. Notons $R_i \in \mathbb{R}^{n_i \times n}$ l'opérateur qui restreint un vecteur global au $i$ème sous-domaine, recouvrement compris. Dans le cas de l'exemple donné en figures 2.3 et 2.5, $R_1$ est l'opérateur qui restreint aux points de $\Omega_1$. De la même manière, on note $\widetilde{R}_i \in \mathbb{R}^{n_i \times n}$ l'opérateur qui restreint les inconnues au $i$ème sous-domaine, en mettant à $0$ les composantes du vecteur qui correspondent au recouvrement.

Les algorithmes 5 et 4 sont donnés pour des conditions aux limites de Robin générales sur les interfaces artificielles. Dans le reste de ce manuscrit, on se limitera au cas où $\Lambda_0 = \Lambda_1 = I$ et $\lambda_0 = \lambda_1 = 0$, c'est-à-dire à la version Dirichlet-Dirichlet. On montre maintenant que cette méthode de Schwarz appliquée au système linéaire $Ax = b$ est équivalente à un processus de Richardson. Pour cela, on introduit les notations suivantes :

- $W_i$ est l'ensemble des numéros globaux des inconnues du sous-domaine $i$. Dans les exemples donnés en figures 2.3 et 2.5, il s'agit des points de $\Omega_i$.
- $A_i$ est la partie de l'opérateur $A$ associée au sous-domaine $i$ : $A_i = R_i A R_i^T$. Dans le cas du problème associé au maillage présenté en figure 2.5, si on numérote les inconnues de manière naturelle, alors les matrices $A_0$ et $A_1$ peuvent être vues comme des blocs de la matrice $A$ (voir la figure 2.6).
- $x_i = R_i x$ et $b_i = R_i b$ sont les restrictions au sous-domaine $i$ de la solution et du second membre.
- $x_{i,e}$ représente les dépendances de données extérieures au sous-domaine $i$ : $x_{i,e}$ contient l'ensemble des $x_j$ tels que $A_{kj} \neq 0$ avec $k \in W_i$ et $j \notin W_i$. Sur l'exemple présenté en figure 2.3, les nœuds de $x_{1,e}$ sont ceux de l'ensemble $\Gamma_1$.
- $R_{i,e}$ est l'opérateur de restriction tel que $R_{i,e}x = x_{i,e}$.

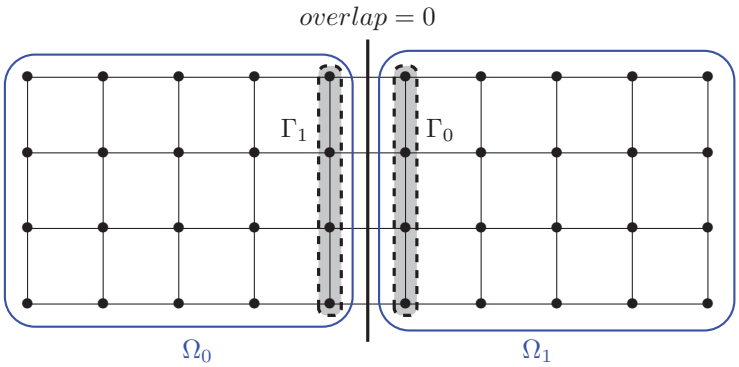


**FIGURE 2.3 –** *Découpage d'un domaine 2D sans recouvrement dans le cas d'une grille régulière.*

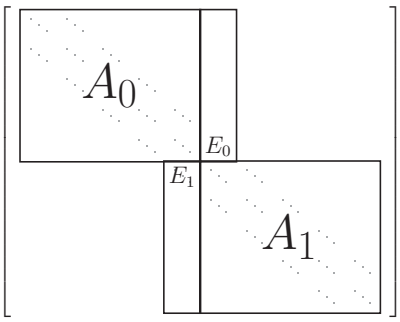


**FIGURE 2.4 –** *Matrices locales correspondant au domaine donné en figure 2.3.*

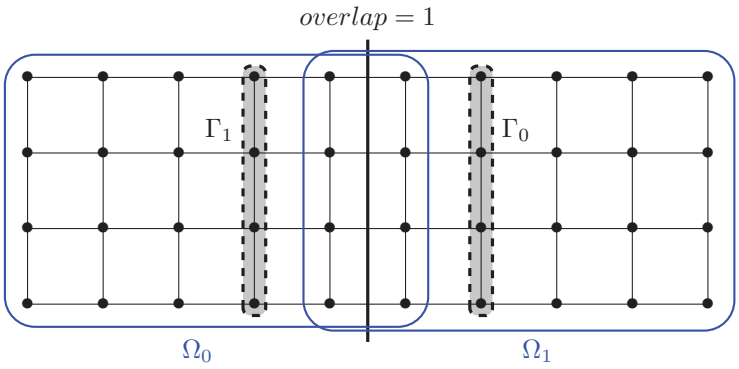


**FIGURE 2.5 –** *Découpage d'un domaine 2D avec un recouvrement de un dans le cas d'une grille régulière.*

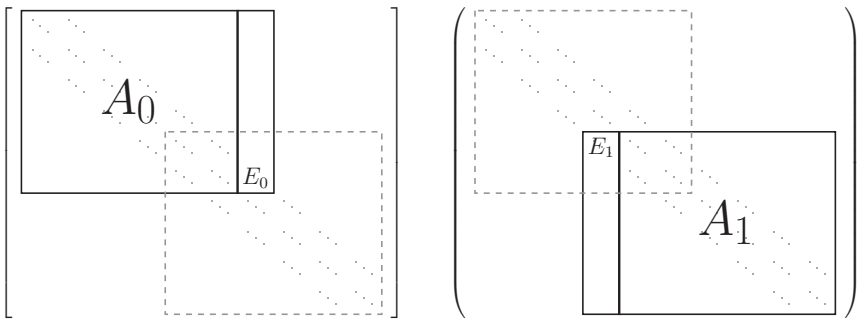


**FIGURE 2.6 –** *Matrices locales correspondant au domaine donné en figure 2.5.*

- $E_i$ est la matrice qui représente l'influence des inconnues extérieures au sous-domaine $i$ sur les inconnues du sous-domaine $i$ : $E_i = R_i A R_{i,e}^T$ (voir les figures 2.6 et 2.6).

Ainsi, une itération de Schwarz additive restreinte au sous-domaine $i$ peut s'écrire comme :

$$x_i^{k+1} = A_i^{-1}\left(b_i - E_i x_{i,e}^k\right). \tag{2.20}$$

#### 2.2.2.2 Méthode de Schwarz additive mise sous la forme d'un processus de Richardson

On peut ensuite montrer (voir [53] et les références qu'il contient) que la méthode de Schwarz peut s'écrire comme le processus de Richardson suivant :

$$x^{k+1} = x^k + M_{RAS}^{-1}\left(b - Ax^k\right) \tag{2.21}$$

avec

$$M_{RAS}^{-1} = \sum_{i=0}^{N-1} \widetilde{R}_i^T \left(R_i A R_i^T\right)^{-1} R_i = \sum_{i=0}^{N-1} \widetilde{R}_i^T A_i^{-1} R_i. \tag{2.22}$$

La matrice $M_{RAS}^{-1}$ est le préconditionneur *Restricted Additive Schwarz* [23]. Son application à des vecteurs requiert la résolution des problèmes locaux.

**Théorème 2.** *Si la solution initiale est la même, alors la suite des itérés produits par la méthode additive de Schwarz de l'équation* (2.20) *est la même que celle produite par le processus de Richardson de l'équation* (2.21).

*Démonstration.* Remarquons d'abord que $R_i A = R_i A\left(R_{i,e}^T R_{i,e} + R_i^T R_i\right)$, car la multiplication par $R_{i,e}^T R_{i,e} + R_i^T R_i$ préserve toutes les colonnes de $R_i A$ ayant des éléments non nuls. Montrons ensuite que :

$$\sum_{i=0}^{N-1} \widetilde{R}_i^T A_i^{-1} R_i A R_{i,e}^T R_{i,e} = M_{RAS}^{-1} A - I. \tag{2.23}$$

En effet :

$$\begin{aligned}\sum_{i=0}^{N-1} \widetilde{R}_i^T A_i^{-1} R_i A R_{i,e}^T R_{i,e} + I &= \sum_{i=0}^{N-1} \widetilde{R}_i^T A_i^{-1} R_i A R_{i,e}^T R_{i,e} + \sum_{i=0}^{N-1} \widetilde{R}_i^T A_i^{-1} R_i A R_i^T R_i \\ &= \sum_{i=0}^{N-1} \left(\widetilde{R}_i^T A_i^{-1} R_i\right) A \left(R_{i,e}^T R_{i,e} + R_i^T R_i\right) = M_{RAS}^{-1} A.\end{aligned} \tag{2.24}$$

On peut maintenant réécrire les itérations de Schwarz sous la forme d'un processus de Richardson :

$$\begin{aligned}x^{k+1} &= \sum_{i=0}^{N-1} \widetilde{R}_i^T x_i^{k+1} = \sum_{i=0}^{N-1} \widetilde{R}_i^T A_i^{-1} \left(b_i - E x_{i,e}^k\right) \\ &= M_{RAS}^{-1} b - \sum_{i=0}^{N-1} \widetilde{R}_i^T A_i^{-1} R_i \left(A R_{i,e}^T R_{i,e} x^k\right) \\ &= M_{RAS}^{-1} b - \left(M_{RAS}^{-1} A - I\right) x^k = x^k + M_{RAS}^{-1} \left(b - A x^k\right).\end{aligned} \tag{2.25}$$

Le passage de l'avant-dernière ligne à la dernière ligne s'effectue à l'aide de l'égalité (2.23). □

**Théorème 3.** *Si on note $x^\infty$ la solution exacte du problème $Ax = b$, et $x^k$ la solution après $k$ itérations de la méthode de Schwarz additive, alors on a l'égalité suivante :*

$$x^k - x^\infty = \left(I - M_{RAS}^{-1} A\right) \left(x^{k-1} - x^\infty\right) \tag{2.26}$$

*qui définit la convergence purement linéaire.*

*Démonstration.* Cette propriété de convergence vient de l'écriture de la méthode de Schwarz comme un processus de Richardson :

$$x^{k+1} = \left(I - M_{RAS}^{-1} A\right) x^k + M_{RAS}^{-1} b. \tag{2.27}$$

En notant $x^\infty$ la solution exacte du problème, on a :

$$x^\infty = \left(I - M_{RAS}^{-1} A\right) x^\infty + M_{RAS}^{-1} b. \tag{2.28}$$

Ainsi, en soustrayant l'équation (2.28) de l'équation (2.27), on obtient l'équation (2.26). L'équation (2.26) montre que la convergence est purement linéaire, c'est-à-dire que l'erreur à l'itération $k$ : $e^k = x^k - x^\infty$ s'écrit comme le produit d'une matrice indépendante de $k$ et de l'erreur à l'itération $k - 1$. □

De la même manière, il est possible de montrer que la méthode de Schwarz multiplicative converge également de manière purement linéaire.

#### 2.2.2.3 Convergence de la solution restreinte à l'interface

On peut écrire un processus itératif similaire à l'équation (2.27) mais restreint aux inconnues des interfaces (voir par exemple [123]), ce qui préserve la propriété de convergence purement linéaire. On définit ici l'opérateur $R_\Gamma$ qui restreint les inconnues à l'interface artificielle. Si l'on considère les exemples en figures 2.3 et 2.5 p. 41 , alors $R_\Gamma$ est l'opérateur qui restreint aux points de $\Gamma_0 \cup \Gamma_1$. La restriction du processus de Richardson (2.27) à l'interface s'écrit donc comme :

$$\begin{aligned} R_\Gamma x^{k+1} &= R_\Gamma \left(I - M_{RAS}^{-1} A\right) x^k + R_\Gamma M_{RAS}^{-1} b \\ &= R_\Gamma \left(I - M_{RAS}^{-1} A\right) R_\Gamma^T R_\Gamma x^k + R_\Gamma M_{RAS}^{-1} b. \end{aligned} \tag{2.29}$$

Le passage de la première ligne à la seconde s'effectue en utilisant l'égalité (2.23). En effet, $R_{i,e}^T R_{i,e} R_\Gamma^T R_\Gamma = R_{i,e}^T R_{i,e}$ car par définition, les dépendances de données extérieures à un sous-domaine font partie de l'interface. Le système réduit à l'interface s'écrit donc comme :

$$\underbrace{R_\Gamma M_{RAS}^{-1} A R_\Gamma^T}_{I-P} \underbrace{R_\Gamma x}_{y} = \underbrace{R_\Gamma M_{RAS}^{-1} b}_{c}. \tag{2.30}$$

Le processus de Richardson restreint à l'interface peut donc s'écrire comme :

$$y^{k+1} = P y^k + c \tag{2.31}$$

où la matrice $P$ définie par l'équation (2.32) est appelée opérateur de propagation d'erreur puisqu'on a $e^{k+1} = P e^k$ où $e^k = y^k - y^\infty$ est l'erreur sur l'interface.

$$P := R_\Gamma \left(I - M_{RAS}^{-1} A\right) R_\Gamma^T. \tag{2.32}$$

On montre maintenant que la solution restreinte aux interfaces artificielles converge de manière purement linéaire, c'est-à-dire qu'on obtient le théorème suivant :

**Théorème 4.** *Notons $R_\Gamma$ l'opérateur de restriction aux interfaces artificielles. Notons également $x^k$ le $k$-ème itéré de la méthode de Schwarz additive, et $x^\infty$ la solution exacte. Si la méthode de Schwarz converge alors la convergence de $y^k = R_\Gamma x^k$ vers $R_\Gamma x^\infty$ est purement linéaire, c'est-à-dire qu'il existe un opérateur linéaire $P$ tel que :*

$$y^{k+1} - y^\infty = P\left(y^k - y^\infty\right). \tag{2.33}$$

*Démonstration.* Remarquons que la condition sur la convergence de la méthode de Schwarz implique que le rayon spectral $\rho(P)$ soit inférieur à 1. Ensuite la démonstration est immédiate en utilisant l'équation (2.31) et en soustrayant deux itérations. On a :

$$y^{k+1} - y^{l+1} = P\left(y^k - y^l\right),\ \forall k, l \in \mathbb{N} \tag{2.34}$$

Si on note $y^\infty = R_\Gamma x^\infty$ la solution exacte à l'interface, alors $y^\infty = P y^\infty + c$. On peut donc remplacer $y^{l+1}$ et $y^l$ par $y^\infty$ dans l'équation précédente, et ainsi retrouver l'équation (2.33). □

Puisque la suite des $R_\Gamma x^n$ converge de manière purement linéaire vers $R_\Gamma x^\infty$, il est possible d'appliquer la formule d'Aitken de l'accélération de suites vectorielles.

### 2.2.3 Accélération d'Aitken

Si l'on considère que la méthode de Schwarz converge strictement quels que soient le second membre et la solution initiale, alors il existe une norme $\|.\|$ telle que $\|P\| < 1$. Ainsi la matrice $P$ et la restriction de la solution exacte à l'interface $R_\Gamma x^\infty = y^\infty$ peuvent être obtenues après $n_\Gamma + 1$ itérations en utilisant les équations suivantes :

$$\left[y^{n_\Gamma+1} - y^{n_\Gamma}, \ldots, y^2 - y^1\right] = P\left[y^{n_\Gamma} - y^{n_\Gamma-1}, \ldots, y^1 - y^0\right]. \tag{2.35}$$

Donc, si la matrice $\left[y^{n_\Gamma} - y^{n_\Gamma-1}, \cdots, y^1 - y^0\right]$ n'est pas singulière, alors la matrice $P$ peut être calculée comme :

$$P = \left[y^{n_\Gamma+1} - y^{n_\Gamma}, \ldots, y^2 - y^1\right]\left[y^{n_\Gamma} - y^{n_\Gamma-1}, \ldots, y^1 - y^0\right]^{-1}. \tag{2.36}$$

Et donc $y^\infty$ peut être calculé comme :

$$y^\infty = (I - P)^{-1}\left(y^{n_\Gamma+1} - Py^{n_\Gamma}\right). \tag{2.37}$$

Notons que la matrice $(I - P)$ de l'équation (2.37) est inversible si et seulement si toutes les valeurs propres de $P$ sont différentes de 1. En particulier, dans le cas où la méthode de Schwarz converge strictement, le rayon spectral $\rho(P)$ est inférieur à 1, et donc $(I - P)$ est inversible. Notons également que l'accélération d'Aitken peut être appliquée même lorsque la méthode de Schwarz diverge, tant que 1 n'est pas une valeur propre de $P$. L'algorithme 6 est celui de l'accélération d'Aitken d'une suite vectorielle à convergence linéaire, dont l'opérateur $P$ n'est pas accessible. Dans cet algorithme on a supposé que la matrice $\left[e^{n_\Gamma-1}, \ldots, e^0\right]$ est inversible, ce qui peut ne pas être le cas, par exemple si $e^0$ est une valeur propre de $P$. Remarquons que dans le cas d'un problème à une seule dimension, discrétisé avec un schéma

**Algorithme 6** Accélération d'Aitken exacte

**Require:** $\mathcal{G} : \mathbb{R}^{n_\Gamma} \to \mathbb{R}^{n_\Gamma}$ a linear function $\mathcal{G}(u) = Pu + c$
**Require:** an initial guess : $y^0$
1: **for** $k = 1 \ldots n_\Gamma + 1$ **do**
2: $\quad y^k \leftarrow \mathcal{G}(y^{k-1})$
3: **end for**
4: $e^k \leftarrow y^{k+1} - y^k,\ 1 \le k \le n_\Gamma$
5: $P \leftarrow \left[e^{n_\Gamma}, \ldots, e^1\right]\left[e^{n_\Gamma-1}, \ldots, e^0\right]^{-1}$
6: $y^\infty \leftarrow (I - P)^{-1}(y^{n_\Gamma+1} - Py^{n_\Gamma})$

à trois points, la solution exacte est obtenue après autant d'itérations de Schwarz qu'il y a de

sous-domaines. En pratique, dans le cas d'un problème à trois dimensions, il n'est pas possible d'utiliser la forme exacte de l'accélération d'Aitken. La raison est qu'il faudrait faire $n_\Gamma + 1$ itérations de Schwarz, où $n_\Gamma$ est le nombre total d'inconnues des interfaces artificielles. Par exemple, dans le cas d'une interface composée de $256 \times 256$ inconnues, il faudrait effectuer $256 \times 256 + 1 = 65537$ itérations de Schwarz pour utiliser la formule d'Aitken. Dans ce qui suit, l'accélération d'Aitken sera effectuée dans un sous-espace de petite dimension. Comme l'accélération ne sera plus exacte, il sera nécessaire de réitérer ces accélérations.

# Chapitre 3

# Accélération d'Aitken de la convergence de la méthode de Schwarz

## Sommaire

L'accélération d'Aitken exacte a été présentée dans le chapitre précédent. Cette accélération peut être construite après $n_\Gamma + 1$ itérations, où $n_\Gamma$ est le nombre total d'inconnues sur les interfaces artificielles. Ce nombre peut être diminué grâce à des méthodes de coloriage : dans le cas d'un domaine 1D, un coloriage rouge-noir permet de construire l'accélération d'Aitken exacte à partir de trois itérations de Schwarz, quel que soit le nombre de sous-domaines. Cependant, dans le cas de problèmes 3D, le coloriage ne permet pas de se ramener à un nombre d'itérations raisonnable. Ce chapitre présente une méthode d'approximation de l'accélération d'Aitken dans un sous-espace de $\mathbb{R}^{n_\Gamma}$. On étudiera d'abord l'accélération dans un sous-espace général, on présentera ensuite l'accélération d'Aitken dans un espace permettant d'approcher les traces. Cet espace sera calculé à partir de la décomposition en valeurs singulières (SVD) des traces. Cette idée remonte à [118] et nous en avons réalisé la première version massivement parallèle (*Computer & Fluids* [11]). La méthode d'Aitken-Schwarz ($A-S$ : le tiret permettant d'éviter la confusion avec *Additive Schwarz*) utilisant la SVD sera dite *classique* même si d'autres espaces d'approximation peuvent être considérés (voir [9] et ses références). On discutera également de la mise à jour à la volée de la SVD [8] afin d'économiser de la mémoire, surtout lorsque le nombre de points sur les interfaces artificielles est important. Enfin, on présentera une nouvelle méthode [10] (appelée $SA-S$ pour *Sparse Aitken-Schwarz*) qui permet d'exploiter la structure creuse de l'opérateur de propagation d'erreur de la méthode de Schwarz. Des résultats numériques sur le problème des écoulements en milieux poreux hétérogènes sont présentés. Ils montrent que cette nouvelle méthode est bien plus robuste et extensible que la méthode classique.

## 3.1 Approximation de l'accélération

Dans ce chapitre, on reprend les notations introduites au chapitre précédent (voir 2.2.2, p.40).

### 3.1.1 Accélération d'Aitken dans un sous-espace général

Afin d'approcher la solution exacte $y^\infty \in \mathbb{R}^{n_\Gamma}$ sur l'interface $\Gamma$, on remplace l'opérateur de propagation d'erreur $P \in \mathbb{R}^{n_\Gamma \times n_\Gamma}$ de l'équation (2.37) par une approximation de faible rang. Cette approximation s'écrit $\widetilde{P} = UU^T PUU^T$ où $U \in \mathbb{R}^{n_\Gamma \times l}$ est une matrice dont les colonnes sont orthonormées. Le système à l'interface résolu par la formule d'Aitken s'écrit $(I-P)y^\infty = c$ avec $c = R_\Gamma M_{RAS}^{-1} b$. L'accélération d'Aitken s'effectue dans l'espace engendré par les colonnes de $U$, noté $span(U)$. Dans l'idéal, c'est-à-dire si l'on a $y^\infty \in span(U)$, on a :

$$\begin{aligned} \left(I - UU^T PUU^T\right) y^\infty &= y^\infty - UU^T P y^\infty \\ &= UU^T c. \end{aligned} \tag{3.1}$$

Le passage de la première à la deuxième ligne s'effectue en remarquant que la solution exacte $y^\infty$ est telle que $y^\infty = Py^\infty + c$. On peut éviter de calculer $UU^T c$ s'il existe un entier $q$ tel que

$y^q$ et $y^{q-1} \in span(U)$. Dans ce cas,

$$\begin{aligned} UU^T c &= UU^T \left( y^q - P y^{q-1} \right) \\ &= UU^T y^q - UU^T P U U^T y^{q-1}. \end{aligned} \tag{3.2}$$

En pratique, il n'y aura pas $y^\infty \in span(U)$, mais on cherchera à construire la matrice $U$ telle que la norme $\|y^\infty - UU^T y^\infty\|$ soit petite. De plus, la matrice $\widetilde{P}$ ne sera pas exactement égale à $UU^T P U U^T$ mais en sera une approximation. Enfin, les $y^i$ ne sont pas non plus exacts dans la mesure où les problèmes locaux sont résolus de manière inexacte par une méthode de Krylov. Pour toutes ces raisons, le système à l'interface devient :

$$\left( I - \widetilde{P} \right) \widetilde{y^\infty} = y^k - \widetilde{P} y^{k-1} \tag{3.3}$$

où $\widetilde{y^\infty}$ est une approximation de $y^\infty$. Il est parfois possible de construire $U$ a priori, par exemple lorsqu'on peut décomposer $y^\infty$ dans l'espace de Fourier avec peu de modes. On considèrera cependant uniquement le cas où la matrice $U$ est calculée de manière algébrique, a posteriori, c'est-à-dire à partir des $q+1$ traces $\left[ y^q, \ldots, y^0 \right]$ issues de $q$ itérations de Schwarz. La construction de $U$ et de $\widetilde{P}$ sera détaillée en section 3.1.3. L'équation (3.3) requiert l'inversion d'une matrice de taille $n_\Gamma \times n_\Gamma$, ce qui n'est pas envisageable pour des problèmes 3D. On peut cependant utiliser la proposition suivante :

**Proposition 2.** *L'équation* (3.3) *peut se réécrire comme l'équation* (3.4) *qui requiert l'inversion d'une matrice de dimension* $l << n_\Gamma$.

$$y^\infty \approx \widetilde{y^\infty} = U \left( I - \widehat{P} \right)^{-1} \left( U^T y^q - \widehat{P} U^T y^{q-1} \right) \tag{3.4}$$

*Démonstration.* D'après les identités matricielles de Woodbury, on a :

$$\begin{aligned} \left( I - U \widehat{P} U^T \right)^{-1} &= I - U \left( I - \widehat{P}^{-1} \right)^{-1} U^T \\ &= I - U \left( I - \left( I - \widehat{P} \right)^{-1} \right) U^T. \end{aligned} \tag{3.5}$$

Ainsi, en utilisant le fait que $y^q$ et $\widetilde{P} y^{q-1}$ sont dans $span(U)$, on a :

$$\begin{aligned} \widetilde{y^\infty} &:= \left( I - \widetilde{P} \right)^{-1} \left( y^q - \widetilde{P} y^{q-1} \right) \\ &= \left( I - U \left( I - \left( I - \widehat{P} \right)^{-1} \right) U^T \right) \left( y^q - \widetilde{P} y^{q-1} \right) \\ &= \underbrace{\left( I - UU^T \right) \left( y^q - \widetilde{P} y^{q-1} \right)}_{=0} + U \left( I - \widehat{P} \right)^{-1} U^T \left( y^q - \widetilde{P} y^{q-1} \right) \\ &= U \left( I - \widehat{P} \right)^{-1} \left( U^T y^q - U^T \widetilde{P} U U^T y^{q-1} \right) \\ &= U \left( I - \widehat{P} \right)^{-1} \left( U^T y^q - \widehat{P} U^T y^{q-1} \right). \end{aligned}$$

□

L'algorithme général de l'accélération d'Aitken dans un sous-espace est l'algorithme 7. L'étape 3 de cet algorithme est la restriction à l'interface d'une itération de Schwarz, qui sera donc implémentée comme dans l'équation (2.29). Chaque itération de l'étape 3 demandera d'effectuer les résolutions locales et l'échange des conditions limites artificielles. Il a été consi-

**Algorithme 7** Accélération d'Aitken approchée

**Require:** $y^0$ an initial guess
1: **repeat**
2: **for** $i = 1 \dots q$ **do**
3: $y^i \leftarrow Py^{i-1} + c$ //Schwarz iterations
4: **end for**
5: Compute a matrix $U$ with orthonormal columns such that $y^q$ and $y^{q-1} \in span(U)$
6: Compute $\widehat{P}$ an approximation of $U^T PU$
7: $y^0 \leftarrow U\left(I - \widehat{P}\right)^{-1}\left(U^T y^q - \widehat{P}U^T y^{q-1}\right)$
8: **until** convergence

déré que l'opérateur $P$ était estimé à partir de $q+1$ itérations successives du processus de Richardson. En réalité, l'opérateur $P$ peut être estimé à partir de $q$ vecteurs arbitraires et de leur image après une itération de Schwarz.

### 3.1.2 Espace représentatif des dernières traces

On construit ici un sous-espace de petite dimension, dans lequel sera calculé $\widehat{P}$, l'approximation de $P$ à partir de la matrice $Y = \left[y^0, \cdots, y^q\right] \in \mathbb{R}^{n_\Gamma \times (q+1)}$, contenant $q+1$ traces. La décomposition en valeurs singulières (SVD) [69] de la matrice $Y$ donne l'équation (3.6), où $\mathbb{U}$ est une matrice $n_\Gamma \times n_\Gamma$ dont les colonnes sont orthonormées, et où $\mathbb{S}$ est une matrice diagonale de dimension $n_\Gamma \times (q+1)$ avec $\mathbb{S}_{ii} = \sigma_i$, $1 \leq i \leq (q+1)$, et la matrice $\mathbb{V} \in \mathbb{R}^{(q+1)\times(q+1)}$ est orthogonale. Les vecteurs singuliers de gauche $\mathbb{U}$ et de droite $\mathbb{V}$ sont respectivement les vecteurs propres de $YY^*$ et de $Y^*Y$.

$$Y = \mathbb{U}\,\mathbb{S}\,\mathbb{V}^T \tag{3.6}$$

En pratique, il existe une version dite "économique" de la SVD, qui calcule seulement les $q+1$ premières colonnes de $\mathbb{U}$ nécessaires pour écrire l'équation (3.7), où $U_i$ et $V_i$ sont les $i$èmes colonnes de $\mathbb{U}$ et $\mathbb{V}$, et $\sigma_i$ est la $i$ème plus grande valeur singulière.

$$Y = \sum_{i=1}^{q+1} \sigma_i\, U_i\, V_i^T \tag{3.7}$$

La troncature de la SVD de $Y$ à ses $l$ plus grandes valeurs singulières donne la matrice $\widetilde{Y}$ de rang $l$ telle que la norme de Frobenius $||Y - \widetilde{Y}||_F$ est minimale. Cette SVD tronquée est

donnée en équation (3.8).

$$\widetilde{Y} = \sum_{i=1}^{l} \sigma_i \, U_i \, V_i^T \tag{3.8}$$

Dans la suite, on notera $U$ la matrice correspondant aux $l$ premières colonnes de $\mathbb{U}$ associées aux $l$ plus grandes valeurs singulières, c'est-à-dire celles que l'on juge significatives.

### 3.1.3 Approximation de l'opérateur de propagation d'erreur

Puisque les itérations de Schwarz réduites à l'interface s'écrivent $y^k = Py^{k-1} + c$, après $q$ itérations de Schwarz on a :

$$\left[y^q - y^{q-1}, \ldots, y^2 - y^1\right] = P\left[y^{q-1} - y^{q-2}, \ldots, y^1 - y^0\right] \tag{3.9}$$

et donc :

$$\begin{aligned} U^T\left[y^q - y^{q-1}, \ldots, y^2 - y^1\right] &= U^T P\left[y^{q-1} - y^{q-2}, \ldots, y^1 - y^0\right] \\ &= U^T PUU^T\left[y^{q-1} - y^{q-2}, \ldots, y^1 - y^0\right] \\ &\quad + U^T P\left(I - UU^T\right)\left[y^{q-1} - y^{q-2}, \ldots, y^1 - y^0\right]. \end{aligned} \tag{3.10}$$

Par construction, lorsque l'on tronque la SVD aux $l$ premières valeurs singulières, on a :

$$\left\|\left(I - UU^T\right)\left[y^{q-1} - y^{q-2}, \ldots, y^1 - y^0\right]\right\|_2 \leq 2\sigma_{l+1}. \tag{3.11}$$

De plus, si la méthode de Schwarz converge alors $\|P\|_2 < 1$. En négligeant $U^T P\left(I - UU^T\right)\left[y^{q-1} - y^{q-2}, \ldots, y^1 - y^0\right]$, on obtient l'approximation suivante de $U^T PU$ :

$$U^T PU \approx \widehat{P} := U^T\left[y^q - y^q, \ldots, y^2 - y^1\right]\left(U^T\left[y^{q-1} - y^{q-2}, \ldots, y^1 - y^0\right]\right)^+ \tag{3.12}$$

où $E^+$ désigne la pseudo-inverse de Moore-Penrose de la matrice $E$, qui est égale à $E^{-1}$ lorsque $E$ est inversible. Finalement, l'approximation de faible rang de l'opérateur de propagation d'erreur $P$ est $\widetilde{P} = U\widehat{P}U^T$.

Lorsque la méthode de Schwarz converge, l'ensemble des $q+1$ dernières traces du processus de Schwarz $[y^0, \cdots, y^q]$ converge vers $[y^\infty, \cdots, y^\infty]$. La matrice $U$ converge donc vers $[y^\infty/\|y^\infty\|_2, 0, \cdots, 0]$, et donc $\|UU^T y^\infty - y^\infty\|_2$ tend vers $0$.

### 3.1.4 Mise à jour de la décomposition en valeurs singulières

En pratique, le nombre de traces nécessaires à l'accélération peut varier d'une accélération à l'autre. La décroissance des valeurs singulières renseigne sur la qualité de l'espace, et donc sur l'accélération potentielle. Il est possible de construire la SVD au fur et à mesure des itérations de Schwarz. Cela revient à mettre à jour une SVD existante comme dans [68]. On considère la décomposition en valeurs singulières de $Y \in \mathbb{R}^{n_\Gamma \times l}$ suivante : $Y = USV^T$. Si on

ajoute $u$ colonnes à la matrice $Y$, on obtient la matrice $[Y\ Y_u] \in \mathbb{R}^{n_\Gamma \times (l+u)}$ dont la SVD peut être calculée de manière économique à partir de $U$, $S$, et $V$. Soit $L = U^T Y_u$ la projection orthonormée de $Y_u$ dans $span(U)$. Soit également $H_u = (I - UU^T)Y_u = Y_u - UL$ la partie de $Y_u$ qui est orthogonale à l'espace créé par les colonnes de la matrice $U$. On considère la décomposition QR $JK = H_u$ suivante :

$$\begin{aligned} \left[\begin{array}{cc} U & J \end{array}\right] \left[\begin{array}{cc} S & L \\ 0 & K \end{array}\right] \left[\begin{array}{cc} V & 0 \\ 0 & I \end{array}\right]^T &= \left[\begin{array}{cc} USV^T & UL + JK \end{array}\right] \\ &= \left[\begin{array}{cc} Y & Y_u \end{array}\right] \end{aligned} \tag{3.13}$$

où l'on a utilisé le fait que $JK = H_u = (I - UU^T)Y_u = Y_u - UL$. On note

$$Q = \left[\begin{array}{cc} S & L \\ 0 & K \end{array}\right]. \tag{3.14}$$

Cette matrice $Q$ est diagonale, sauf sur les $u$ dernières colonnes. Afin de mettre à jour la SVD, la SVD de $Q$ se calcule comme :

$$Q = U' S' V'^T. \tag{3.15}$$

La nouvelle SVD peut donc s'écrire comme :

$$[Y\ Y_u] = U'' S'' V''^T \tag{3.16}$$

avec

$$U'' = [U\ J]U',\ S'' = S',\ V'' = \left[\begin{array}{cc} V & 0 \\ 0 & I \end{array}\right] V'. \tag{3.17}$$

La mise à jour d'une SVD existante commence donc par la séparation des nouvelles données en deux parties : les composantes déjà incluses dans l'espace de la SVD existante, et celles qui sont orthogonales à cet espace. Plus de détails concernant cette méthode sont disponibles dans [74]. Le processus de mise à jour comporte donc quatre étapes :

- Orthogonaliser les vecteurs colonnes de $Y_u$ contre les colonnes de $U$, c'est-à-dire les vecteurs singuliers existants. Il s'agit donc d'une décomposition QR.
- Construire la matrice $Q$ définie dans l'équation (3.15) et calculer sa SVD comme dans l'équation (3.16).
- Appliquer l'équation (3.17) de manière à obtenir la nouvelle décomposition.
- Tronquer de cette nouvelle décomposition en valeurs singulières de manière à ne garder que les modes associés à des valeurs singulières significatives.

L'étape de la décomposition QR telle que décrite dans [75] utilise une méthode de Gram-Schmidt partielle modifiée de manière à réduire les coûts de calcul. Cependant, la méthode de Householder utilisée dans [44] permet d'obtenir une meilleure orthogonalité.

Finalement, le calcul de la SVD de $[Y\ Y_u] \in \mathbb{R}^{n_\Gamma \times (l+u)}$ est remplacé par une décomposition QR de $H_u \in \mathbb{R}^{n_\Gamma \times u}$ et une SVD de $Q \in \mathbb{R}^{(l+u)\times(l+u)}$. Cela permet donc d'économiser des calculs. Cependant, l'avantage principal de cette méthode est qu'il est possible de tronquer la SVD après chaque mise à jour, et donc d'utiliser moins de mémoire pour stocker la matrice $U$ : par exemple, considérons le cas où l'accélération d'Aitken est construite à partir de $40$ itérations de Schwarz, et que $20$ modes seront finalement gardés. On peut donc poser $u = 5$, c'est-à-dire orthogonaliser cinq traces à la fois. Dans ce cas, la matrice $[Y\ Y_u]$ aura au plus $20 + 5$ colonnes, ce qui permet d'économiser le stockage d'au minimum $15$ vecteurs de taille $n_\Gamma$. En réalité, la troncature à chaque mise à jour est nécessaire à la préservation de l'orthogonalité car les vecteurs singuliers associés aux petites valeurs singulières sont bruités numériquement. Ce bruit doit être écarté avant l'orthogonalisation des nouvelles données contre les colonnes de la matrice $U$. Remarquons tout de même que :

- Du fait de la troncature à chaque mise à jour, la qualité de la base, et donc l'efficacité de l'accélération sera réduite par la mise à jour adaptative.
- Lors de la phase de mise à jour de $U$, seuls les processeurs qui ont la charge de points de l'interface $\Gamma$ travaillent.

Cette approche est donc à réserver aux cas où le nombre d'inconnues aux interfaces artificielles $n_\Gamma$ est grand pour que l'économie de vecteurs de taille $n_\Gamma$ soit significative. Ce sera par exemple le cas lorsque l'on découpe de manière algébrique un problème discrétisé sur un maillage quelconque avec des éléments finis d'ordre élevés.

## 3.2 Préconditionnement des itérations de Richardson

Le calcul de l'accélération d'Aitken requiert une approximation de l'opérateur $P$. Il est possible d'utiliser cette approximation de l'opérateur $P$ pour préconditionner le processus de Richardson après l'accélération. Ce nouveau processus de Richardson préconditionné peut lui aussi être accéléré.

### 3.2.1 Construire un préconditionneur à partir de l'approximation de l'opérateur de propagation d'erreur

Jusqu'ici, la méthode d'Aitken-Schwarz consistait à itérer sur ces deux étapes :

- Effectuer $q+1$ itérations de la méthode additive de Schwarz classique $x^{k+1} = \left(I - M_{RAS}^{-1}A\right)x^k + M_{RAS}^{-1}b$ à partir de $x^0$.
- Calculer $\widetilde{y^\infty}$, la solution à l'interface accélérée par la formule d'Aitken, puis l'utiliser pour calculer une nouvelle solution initiale : $x_0 = R_\Gamma^T\widetilde{y^\infty} + (I - R_\Gamma^T R_\Gamma)x^{q+1}$.

Autrement dit, l'approximation de la matrice $P$ n'est pas utilisée pendant les itérations de Schwarz qui suivent une accélération. Une courbe de convergence typique du processus

d'Aitken-Schwarz est donnée en figure 3.5 page 64. On y observe que l'accélération d'Aitken se traduit par une diminution instantanée de l'erreur, mais que le taux de convergence semble le même avant et après l'accélération. En effet, le rayon spectral du processus de Richardson n'est pas modifié par l'accélération. On présente maintenant comment réutiliser l'approximation de $P$ obtenue pour augmenter la vitesse de convergence après la première accélération.

Lorsque l'on dispose d'une approximation $\widetilde{P}$ de la matrice $P$, il est possible d'effectuer une accélération d'Aitken entre chaque itération de Schwarz. Considérons d'abord uniquement le processus restreint aux interfaces artificielles :

$$\left\{ \begin{array}{rcl} y^{k+1/2} & = & R_\Gamma \left( \left( I - M_{RAS}^{-1} A \right) x^k + M_{RAS}^{-1} b \right) \\ y^{k+1} & = & \left( I - \widetilde{P} \right)^{-1} \left( y^{k+1/2} - \widetilde{P} y^k \right). \end{array} \right. \tag{3.18}$$

La première ligne de l'équation (3.18) correspond à une itération classique de la méthode de Schwarz additive, la deuxième est l'accélération d'Aitken qui peut être réécrite comme :

$$\begin{array}{rcl} y^{k+1} & = & \left( I - \widetilde{P} \right)^{-1} \left( y^{k+1/2} + y^k - y^k - \widetilde{P} y^k \right) \\ & = & y^k + \left( I - \widetilde{P} \right)^{-1} \left( y^{k+1/2} - y^k \right). \end{array} \tag{3.19}$$

Pour écrire le processus de Richardson sur le domaine complet, on choisit d'écrire :

$$x^{k+1} = R_\Gamma^T y^{k+1} + \left( I - R_\Gamma^T R_\Gamma \right) x^{k+1/2}. \tag{3.20}$$

Après la première accélération, on peut donc construire le processus de Richardson suivant :

$$x^{k+1} = x^k + \left( I + R_\Gamma^T \left( \left( I - \widetilde{P} \right)^{-1} - I \right) R_\Gamma \right) M_{RAS}^{-1} \left( b - A x^k \right). \tag{3.21}$$

Par la suite, on notera :

$$\widetilde{Q} = \left( I + R_\Gamma^T \left( \left( I - \widetilde{P} \right)^{-1} - I \right) R_\Gamma \right). \tag{3.22}$$

Dans [45] la matrice $\widetilde{Q} M_{RAS}^{-1}$ est appelée $M_{ARAS}^{-1}$ et elle est utilisée comme préconditionneur lors de la résolution de systèmes linéaires par une méthode de Krylov. Elle est ainsi construite avant les itérations de Krylov et reste constante durant ces itérations. L'avantage d'utiliser la méthode d'Aitken-Schwarz en tant que solveur est que cette matrice peut être construite naturellement à chaque accélération d'Aitken, c'est-à-dire pendant la résolution du système linéaire. Notons aussi qu'il est possible de construire cette matrice à partir de la base d'Arnoldi lorsque la méthode GMRES est utilisée.

### 3.2.2 Convergence purement linéaire du nouveau processus

On propose donc de résoudre le système linéaire en trois étapes :

1. Effectuer quelques itérations de la méthode de Schwarz additive classique $x^{k+1} = \left(I - M_{RAS}^{-1}A\right)x^k + M_{RAS}^{-1}b$ à partir de $x_0$.
2. Approcher la solution accélérée par la formule d'Aitken, puis $R_\Gamma x_0 = \widetilde{y^\infty}$.
3. Effectuer des itérations avec le processus $x^{k+1} = x^k + \widetilde{Q}M_{RAS}^{-1}\left(b - Ax^k\right)$.

Il reste donc à montrer comment boucler sur les étapes 2 et 3. Pour cela, il faut d'abord montrer que le nouveau processus de Richardson génère des traces qui convergent vers la même solution que la méthode de Schwarz. Ensuite, il faudra écrire la formule d'Aitken adaptée à ce nouveau processus.

**Théorème 5.** *La suite de vecteurs donnée par la restriction aux interfaces artificielles du processus de Richardson* (3.21) *converge purement linéairement vers $y^\infty$ si et seulement si $\rho(I - \widetilde{Q}M_{RAS}^{-1}A) < 1$.*

*Démonstration.* La restriction aux interfaces artificielles de l'équation (3.21) est donnée par

$$y^{k+1} = y^k + \left(I - \widetilde{P}\right)^{-1}\left(y^{k+1/2} - y^k\right). \tag{3.23}$$

Or la méthode de Schwarz converge purement linéairement, c'est-à-dire que l'on peut écrire que : $y^{k+1/2} = Py^k + c$, où $c$ est un vecteur constant. En injectant cette égalité dans l'équation ci-dessus, on obtient :

$$\begin{aligned} y^{k+1} &= y^k + \left(I - \widetilde{P}\right)^{-1}(P - I)\,y^k + \left(I - \widetilde{P}\right)^{-1}c \\ &= \left(I - \left(I - \widetilde{P}\right)^{-1}(I - P)\right)y^k + \left(I - \widetilde{P}\right)^{-1}c \\ &= Ty^k + \left(I - \widetilde{P}\right)^{-1}c. \end{aligned} \tag{3.24}$$

Ainsi, on a bien que $y^{k+1} - y^\infty = T(y^k - y^\infty)$ avec $T = \left(I - \left(I - \widetilde{P}\right)^{-1}(I - P)\right)$.

Puisque le nouveau processus de Richardson (3.24) converge purement linéairement, il est possible de l'accélérer. L'accélération d'Aitken de ce nouveau processus s'écrit comme :

$$\begin{aligned} y^\infty &= (I - T)^{-1}\left(y^{k+1} - Ty^k\right) \\ &= \left(\left(I - \widetilde{P}\right)^{-1}(I - P)\right)^{-1}\left(y^{k+1} - \left(I - \left(I - \widetilde{P}\right)^{-1}(I - P)\right)y^k\right) \\ &= (I - P)^{-1}\left(I - \widetilde{P}\right)\left(y^{k+1} - y^k + \left(I - \widetilde{P}\right)^{-1}(I - P)\,y^k\right) \\ &= y^k + (I - P)^{-1}\left(I - \widetilde{P}\right)\left(y^{k+1} - y^k\right) \\ &= y^k + (I - P)^{-1}\left(y^{k+1/2} - y^k\right) \\ &= (I - P)^{-1}\left(y^{k+1/2} - Py^k\right). \end{aligned} \tag{3.25}$$

On vient de montrer que l'accélération d'Aitken est la même pour le Schwarz classique que pour le processus de Richardson préconditionné. Les processus avec et sans préconditionnement convergent donc bien vers la même solution. □

### 3.2.3 Aitken-Schwarz préconditionné

On peut penser utiliser la méthode présentée ci-dessus de manière itérative : à chaque itération on construit un nouveau processus de Richardson qui sera accéléré à l'itération suivante. Cela donne l'algorithme 8. En général, la matrice de préconditionnement $Q$ de l'algorithme

**Algorithme 8** Aitken-Schwarz préconditionné

**Require:** $x_0$ an initial guess

1: $Q = I$
2: **repeat**
3: **for** $k = 0 \dots q$ **do**
4: $x^{k+1} \leftarrow x^k + QM_{RAS}^{-1}\left(b - Ax^k\right)$ //Richardson process
5: **end for**
6: Estimate the error propagation operator of the current Richardson process $\widetilde{T}$
7: $Q \leftarrow \left(I + R_\Gamma^T\left(\left(I - \widetilde{T}\right)^{-1} - I\right)R_\Gamma\right)Q$
8: $x_0 \leftarrow R_\Gamma^T\left(I - \widetilde{T}\right)^{-1}\left(R_\Gamma x^q - \widetilde{T}R_\Gamma x^{q-1}\right) + \left(I - R_\Gamma^T R_\Gamma\right)x^q$
9: **until** convergence

8 ne sera pas calculée explicitement.

De manière à accélérer le processus préconditionné, on peut penser à estimer directement l'opérateur de propagation d'erreur du processus préconditionné. Dans ce qui suit, on construit cette approximation à partir de l'approximation de l'opérateur $P$ du processus de la méthode Schwarz. Si l'on décompose l'itération de l'étape 4 de l'algorithme 8 en deux demi-itérations comme dans l'équation (3.18), alors on peut construire une nouvelle approximation de l'opérateur de propagation d'erreur du processus de Schwarz $P$. Si l'on note $\widetilde{P}_{new}$ cette nouvelle approximation, on peut écrire $\widetilde{T} = (I - (I - \widetilde{P})^{-1}(I - \widetilde{P}_{new}))$. Dans ce cas, remarquons que la matrice $Q$ est la matrice identité sauf sur le bloc correspondant aux interfaces artificielles entre les sous-domaines. À L'étape 7 de l'algorithme 8, on peut donc se contenter de calculer ce bloc. Si on note $Q_i$ la $i$ème valeur de $Q$, on a $Q_0 = I$, et $R_\Gamma Q_1 R_\Gamma^T = (I - \widetilde{P})^{-1}$, puis

$$\begin{aligned} R_\Gamma Q_2 R_\Gamma^T &= \left(I - \widetilde{T}\right)^{-1} R_\Gamma Q_1 R_\Gamma^T \\ &= \left(\left(I - \widetilde{P}\right)^{-1}\left(I - \widetilde{P}_{new}\right)\right)^{-1}\left(I - \widetilde{P}\right)^{-1} \\ &= \left(I - \widetilde{P}_{new}\right)^{-1} \end{aligned} \tag{3.26}$$

Puis par induction on a finalement que $R_\Gamma Q R_\Gamma^T = (I - \widetilde{P}_{new})$. La méthode peut donc se réécrire comme l'algorithme 9.

**Algorithme 9** Aitken-Schwarz préconditionné, en considérant $\widetilde{T} = (I-(I-\widetilde{P})^{-1}(I-\widetilde{P}_{new}))$.

**Require:** $x_0$ an initial guess
1: $\widetilde{P} \leftarrow 0$
2: **repeat**
3: **for** $k = 0 \dots q$ **do**
4: $x^{k+1/2} \leftarrow x^k + M_{RAS}^{-1}\left(b - Ax^k\right)$
5: $x^{k+1} \leftarrow R_\Gamma^T \left(I - \widetilde{P}\right)^{-1} \left(R_\Gamma x^{k+1/2} - \widetilde{P} R_\Gamma x^k\right) + \left(I - R_\Gamma^T R_\Gamma\right) x^{k+1/2}$
6: **end for**
7: Compute an orthonormal basis $U$
8: $\widehat{P} \leftarrow U^T R_\Gamma \left[x^{q+1/2} - x^{q-1/2}, \dots, x^{3/2} - x^{1/2}\right] \left(U^T R_\Gamma \left[x^q - x^{q-1}, \dots, x^1 - x^0\right]\right)^{-1}$
9: $\widetilde{P} \leftarrow U \widehat{P} U^T$
10: **until** convergence

Remarquons que le préconditionneur $\widetilde{Q}$ de l'équation (3.22) est très similaire au préconditionneur basé sur la déflation proposé dans [49].

### 3.2.4 Remarques à propos de la convergence

Lorsque la méthode de Schwarz additive est utilisée en tant que préconditionneur d'une méthode de Krylov, il est possible de ne considérer que les inconnues des interfaces artificielles. On peut donc se contenter de résoudre l'équation (2.30) pour obtenir la solution sur les interfaces artificielles [102]. La reconstruction de la solution à l'intérieur des sous-domaines se fera ensuite avec une résolution locale. Dans le cas de l'accélération d'Aitken telle qu'elle a été vue jusqu'ici, la solution accélérée restreinte à l'interface $\widetilde{y}^\infty$ est dans l'espace engendré par les colonnes de la matrice $U$. Or $U$ étant calculée à partir de la décomposition en valeurs singulières des $y^i$, on a :

$$\widetilde{y}^\infty \in span(U) = span(y^q, \cdots, y^0) = \mathcal{K}^{q+1}(r_0). \tag{3.27}$$

Puisque la solution accélérée reste dans l'espace de Krylov engendré par les $q$ itérations, il devient évident que le processus de Richardson qui inclut l'accélération d'Aitken (algorithme 9) convergera moins rapidement que la méthode GMRES. En effet, la solution après $q+1$ itérations de GMRES est le vecteur de l'espace de Krylov $\mathcal{K}^{q+1}(r_0)$ qui minimise le résidu. Ceci dit, chaque itération de GMRES préconditionné par la méthode de Schwarz est composée de résolutions locales et d'une orthogonalisation. Cette orthogonalisation demande de synchroniser tous les sous-domaines, et seuls les processeurs qui gèrent une partie de l'interface artificielle travaillent. La méthode d'Aitken-Schwarz classique demande moins de communications car l'orthogonalisation (ici c'est une SVD) a lieu toutes les $q$ itérations, et aucune

synchronisation globale n'est nécessaire entre les accélérations. Ce n'est plus le cas lorsqu'on préconditionne le processus de Richardson, car quelle que soit l'implémentation, il y a au moins une synchronisation globale lors de l'application du préconditionneur, même dans le cas où $(I - \widehat{P})^{-1}$ est dupliqué sur tous les processeurs aux interfaces.

Dans la suite, on propose d'exploiter la structure creuse de l'opérateur de propagation d'erreur afin d'effectuer l'accélération dans un sous-espace plus grand que l'espace de Krylov engendré avant l'accélération.

## 3.3 Respect de la structure creuse de l'opérateur de transfert d'erreur

L'accélération d'Aitken était jusqu'à présent effectuée dans un sous-espace de l'espace engendré par les $q+1$ solutions aux interfaces successives, c'est-à-dire un sous-espace de l'espace de Krylov $\mathcal{K}^{q+1}$. Le processus de Richardson avec accélération d'Aitken ne peut donc pas converger plus rapidement que la méthode GMRES sans *restart*. On exploite maintenant la structure de la matrice $P$ ce qui revient à effectuer l'accélération dans un sous-espace plus grand que l'espace de Krylov engendré par les $q+1$ premières itérations. Il s'agira d'estimer séparément les blocs de l'opérateur de propagation d'erreur $P$ associés à chaque sous-domaine. Cette idée peut donc être reliée à l'approximation creuse du complément de Schur de la matrice $A$ discutée dans [67] et à l'approximation locale de l'opérateur de correspondance Dirichlet-Neumann proposée dans [95].

Jusqu'ici, l'approximation de $P = R_\Gamma(I - M_{RAS}^{-1}A)R_\Gamma^T$, donnée par $U\widehat{P}U^T$ était pleine. Or, la matrice $P$ est en général très creuse car la solution à l'interface d'un sous-domaine ne dépend que de la solution aux interfaces des sous-domaines voisins.

Dans la suite, on propose une nouvelle façon de construire l'accélération d'Aitken permettant de conserver les blocs de $0$ de la matrice $P$ correspondant aux interfaces des sous-domaines disjoints. On se limitera d'abord au cas de deux sous-domaines.

### 3.3.1 Formule d'Aitken pour la méthode de Schwarz additive dans le cas de deux sous-domaines

On note $v_0^i$ et $v_1^i$ les solutions aux deux interfaces des sous-domaines à l'itération $i$. On note également $R_{\Gamma_0}$ et $R_{\Gamma_1}$ la restriction à ces deux interfaces. Ainsi, $v_0^i = R_{\Gamma_0}x^i$. Dans ce cas, la convergence linéaire s'écrit comme

$$\begin{bmatrix} v_0^{i+1} - v_0^i \\ v_1^{i+1} - v_1^i \end{bmatrix} = R_\Gamma \left(I - M_{RAS}^{-1}A\right) R_\Gamma^T \begin{bmatrix} v_0^i - v_0^{i-1} \\ v_1^i - v_1^{i-1} \end{bmatrix} \tag{3.28}$$

où la matrice $P = R_\Gamma(I - M_{RAS}^{-1}A)R_\Gamma^T$ peut se décomposer sous la forme :

$$P = \begin{bmatrix} 0 & P_0 \\ P_1 & 0 \end{bmatrix}. \tag{3.29}$$

Si l'on note $e_0^n = v_0^{i+1} - v_0^i$ et $e_1^i = v_1^{i+1} - v_1^i$ alors on a :

$$P_0 \left[ e_1^0, \ldots, e_1^{q-1} \right] = \left[ e_0^1, \ldots, e_0^q \right] \tag{3.30}$$

et

$$P_1 \left[ e_0^0, \ldots, e_0^{q-1} \right] = \left[ e_1^1, \ldots, e_1^q \right]. \tag{3.31}$$

Pour approcher l'accélération d'Aitken dans un espace de petite dimension, on effectue la décomposition en valeurs singulières des traces des deux domaines séparément : $U_i \Sigma_i V_i^T = [v_i^0, \ldots, v_i^{q+1}]$ pour $i = 0, 1$. Ainsi :

$$\left\{ \begin{array}{rcl} \widehat{P_0} & := & \left( U_0^T \left[ e_0^1, \ldots, e_0^q \right] \right) \left( U_1^T \left[ e_1^0, \ldots, e_1^{q-1} \right] \right)^{-1} \approx U_0^T P_0 U_1 \\ \widehat{P_1} & := & \left( U_1^T \left[ e_1^1, \ldots, e_1^q \right] \right) \left( U_0^T \left[ e_0^0, \ldots, e_0^{q-1} \right] \right)^{-1} \approx U_1^T P_1 U_0. \end{array} \right. \tag{3.32}$$

L'approximation de la matrice $P$ sera donc :

$$P \approx \begin{bmatrix} 0 & U_0 \widehat{P_0} U_1^T \\ U_1 \widehat{P_1} U_0^T & 0 \end{bmatrix} = \underbrace{\begin{bmatrix} U_0 & 0 \\ 0 & U_1 \end{bmatrix}}_{U} \times \underbrace{\begin{bmatrix} 0 & \widehat{P_0} \\ \widehat{P_1} & 0 \end{bmatrix}}_{\widehat{P}} \times \underbrace{\begin{bmatrix} U_0 & 0 \\ 0 & U_1 \end{bmatrix}^T}_{U^T}. \tag{3.33}$$

On notera que les blocs diagonaux de $0$ sont préservés, ce qui n'était pas le cas jusqu'ici. Le calcul de l'accélération d'Aitken est effectué après $q$ itérations, l'espace de Krylov associé aux $q$ itérations est donc de dimension $q$. La matrice $U$ qui est composée d'au plus $l \leq 2q + 2$ colonnes linéairement indépendantes, engendre un espace plus grand que l'espace de Krylov de dimension $q + 1$ dès lors que $l > q + 1$.

L'accélération d'Aitken en approchant les blocs de $P$ séparément est naturellement plus parallèle que lorsque la matrice $P$ est considérée pleine :

- Il faut effectuer une décomposition en valeurs singulières par interface au lieu d'une décomposition globale.
- Il est possible de réécrire la formule de l'accélération de manière à éviter un goulot d'étranglement en termes de communications. Cette réécriture est donnée en annexe A.1 p.145.

### 3.3.2 Extension au partitionnement 1D

Les résultats de la sous-section précédente se généralisent naturellement au cas d'un partitionnement rouge-noir. Dans ce cas, les matrices $P_0$ et $P_1$ sont bloc-diagonales. On peut préserver cette structure si l'on approche chaque bloc indépendamment. Cela revient à faire une décomposition en valeurs singulières par interface. Par exemple, dans le cas d'un problème 1D découpé en $4$ sous-domaines comme dans la figure 3.1, on obtient la structure donnée en équation (3.34).

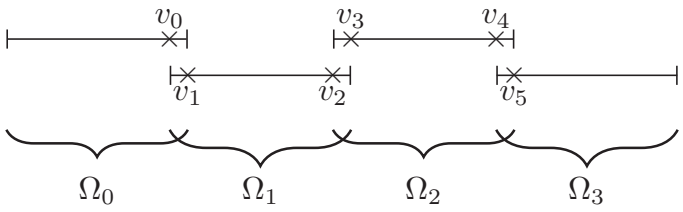


**FIGURE *3.1*** – *Exemple 1D de découpage en 4 sous-domaines.*

$$Pe^k = \begin{bmatrix} & & \bullet & & \\ & & & \bullet & \bullet \\ & & & \bullet & \bullet \\ \bullet & \bullet & & & \\ \bullet & \bullet & & & \\ & & \bullet & & \end{bmatrix} \times \begin{bmatrix} e_0^k \\ e_3^k \\ e_4^k \\ e_1^k \\ e_2^k \\ e_5^k \end{bmatrix} \tag{3.34}$$

où les $\bullet$ désignent les éléments non nuls. La généralisation à un problème à trois dimensions, découpé uniquement dans une seule dimension, est directe : chaque $\bullet$ représente un bloc de la taille d'un vecteur interface $v_i$. Si l'on note chaque bloc $P_i$, où $i$ est le numéro du sous-domaine, alors on a

$$P_i R_{\Gamma_i}^E e^k = R_{\Gamma_i} e^{k+1}, i = 0 \dots N-1 \tag{3.35}$$

où $R_{\Gamma_i}^E$ est l'opérateur de restriction aux interfaces extérieures du domaine $i$ et $R_{\Gamma_i}$ est l'opérateur de restriction aux interfaces internes. On a également

$$\widehat{P}_i = U_i^T P_i U_i^E = \left(U_i^T \left[e_i^1, \dots, e_i^q\right]\right) \left((U_i^E)^T \left[(e_i^E)^0, \dots, (e_i^E)^{q-1}\right]\right)^{-1}, i = 0 \dots N-1 \tag{3.36}$$

où l'exposant $E$ se réfère aux nœuds de l'interface extérieure au sous-domaine. Les matrices $U_i$ et $U_i^E$ sont bloc-diagonales, chaque bloc étant la base tronquée issue de la SVD de l'interface. En reprenant l'exemple de la figure 3.1 et en notant $\mathbb{U}_i$ la base obtenue par la décomposition en valeurs singulières de $\left[v_i^1 \cdots v_i^{q+1}\right]$, on obtient :

- $e_0^k = \left[v_0^{k+1} - v_0^k\right]$, $e_1^k = \begin{bmatrix} v_1^{k+1} - v_1^k \\ v_2^{k+1} - v_2^k \end{bmatrix}$, $e_2^k = \begin{bmatrix} v_3^{k+1} - v_3^k \\ v_4^{k+1} - v_4^k \end{bmatrix}$, $e_3^k = \left[v_5^{k+1} - v_5^k\right]$,
- $(e_0^E)^k = \left[v_1^{k+1} - v_1^q\right]$, $(e_1^E)^k = \begin{bmatrix} v_0^{k+1} - v_0^k \\ v_3^{k+1} - v_3^k \end{bmatrix}$, $(e_2^E)^k = \begin{bmatrix} v_2^{k+1} - v_2^k \\ v_5^{k+1} - v_5^k \end{bmatrix}$, $(e_3^E)^k = \left[v_4^{k+1} - v_4^k\right]$,
- $U_0 = \left[\mathbb{U}_0\right]$, $U_1 = \begin{bmatrix} \mathbb{U}_1 & 0 \\ 0 & \mathbb{U}_2 \end{bmatrix}$, $U_2 = \begin{bmatrix} \mathbb{U}_3 & 0 \\ 0 & \mathbb{U}_4 \end{bmatrix}$, $U_3 = \left[\mathbb{U}_5\right]$,
- $U_0^E = \left[\mathbb{U}_1\right]$, $U_1^E = \begin{bmatrix} \mathbb{U}_0 & 0 \\ 0 & \mathbb{U}_3 \end{bmatrix}$, $U_2^E = \begin{bmatrix} \mathbb{U}_2 & 0 \\ 0 & \mathbb{U}_5 \end{bmatrix}$, $U_3^E = \left[\mathbb{U}_4\right]$.

La méthode présentée ici se généralise naturellement à un partitionnement quelconque dès qu'il est possible de partitionner les points de $R_\Gamma x$ en sous-ensembles de points indépendants, c'est-à-dire dès qu'il y a des ensembles d'interfaces disjoints. En général il y aura au moins deux décompositions en valeurs singulières à effectuer par sous-domaine : une pour les inconnues de l'interface extérieure (celles où la condition limite de Dirichlet est imposée) et l'autre pour les inconnues de l'interface intérieure. Dans le cas du découpage 1D présenté ci-dessus, les interfaces internes et externes sont divisées en deux parties indépendantes (interfaces à gauche et à droite).

### 3.3.3 Illustration sur un problème de Poisson 2D

Sur un problème simple, on compare la convergence des méthodes suivantes :

- $GMRES$ sur le problème restreint à l'interface.
- $GMRES(9)$ : méthode GMRES sur le problème restreint à l'interface avec un *restart* de 9.
- $A{-}S(9)$ : Aitken-Schwarz préconditionné classique à partir de 9 traces.
- $SA{-}S(9)$ pour *Sparse Aitken-Schwarz* : Aitken-Schwarz préconditionné creux à partir de 9 traces.

Notons que le coût d'une itération est sensiblement le même quelle que soit la méthode, dans la mesure où il est largement dominé par l'application de $M_{RAS}^{-1}$.

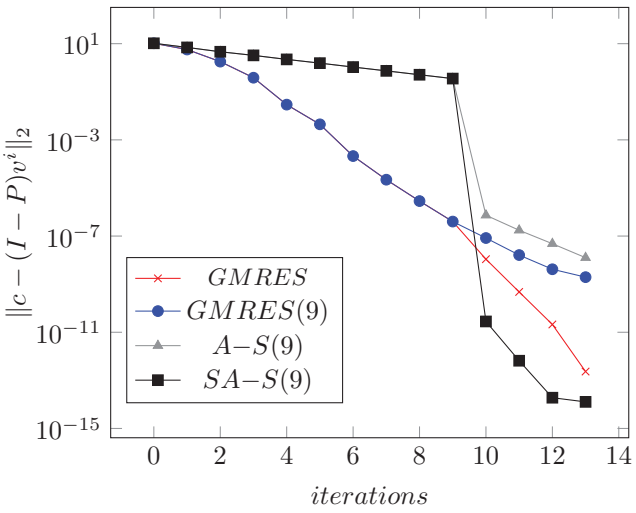


**FIGURE 3.2 –** *Résidus du problème interface, en fonction des itérations. Problème de Poisson 2D sur le carré unité discrétisé en* $20 \times 20$ *points.*

Le problème en question est un problème de Poisson 2D sur le carré unité discrétisé en $20$ points dans chaque direction. Le domaine est décomposé en quatre sous-domaines de même taille dans une seule direction. Le recouvrement entre ces sous-domaines est d'un point. Les résidus du problème restreint à l'interface sont tracés dans la figure 3.2. On constate d'abord que le résidu de la méthode d'Aitken-Schwarz classique $A-S$ est toujours supérieur au résidu obtenu par la méthode GMRES. On peut également voir que la méthode d'Aitken-Schwarz creuse $SA-S(9)$ converge plus rapidement que GMRES(9). Sur cet exemple, la convergence de $SA-S(9)$ est même plus rapide que celle du GMRES sans *restart*. Une comparaison des méthodes $SA-S$ et $A-S$ plus détaillée sera proposée dans la section 3.4.4 p.69.

## 3.4 Expérimentations numériques

Dans les expérimentations numériques suivantes, on considère un problème de l'écoulement dans un milieu poreux 3D. Ce problème sera similaire à un problème de Poisson 3D dans le cas particulier où la perméabilité est constante. Après avoir présenté le problème physique, l'implémentation parallèle de la méthode d'Aitken-Schwarz sera discutée. Premièrement, on illustre la mise à jour de la décomposition en valeurs singulières présentée en 3.1.4 p.51. Comme attendu, la mise à jour de la SVD conduit à une dégradation de la convergence et à une économie d'espace mémoire. La méthode d'Aitken-Schwarz classique $A-S$ sera ensuite comparée à la nouvelle méthode $SA-S$ sur des problèmes modèles. On montrera ainsi que la nouvelle méthode est bien plus extensible et robuste que la méthode classique pour le problème considéré. Ensuite, on illustrera l'effet du préconditionnement du processus de Richardson après l'accélération proposée en 3.4.5 p.72. Le dernier test met en évidence l'extensibilité des différentes composantes de l'implémentation $A-S$ proposée.

### 3.4.1 Présentation du problème

À l'échelle macroscopique, un milieu poreux peut être modélisé par un solide occupant un volume. Ici on considère que le domaine a une forme de parallélépipède (figure 3.3) discrétisé de telle façon que la taille d'un élément soit plus grande que la taille des pores.

L'écoulement en milieu saturé peut être modélisé en utilisant la loi de Darcy et la loi de conservation de la masse pour obtenir l'équation (3.37), où $u$ est la charge hydraulique et $K(x, y, z)$ est le champ de perméabilité.

$$\begin{cases} \nabla.\left(K(x,y,z)\,\nabla u\right) = 0 \text{ dans } \Omega \\ \qquad u = \alpha, \text{ sur } \Gamma_L \\ \qquad u = \beta, \text{ sur } \Gamma_R \\ \dfrac{\partial u}{\partial n} = 0, \text{ sur } \partial\Omega \setminus (\Gamma_1 \cup \Gamma_2) \end{cases} \tag{3.37}$$

Dans les résultats numériques, on considèrera plusieurs types de champs de perméabilité,

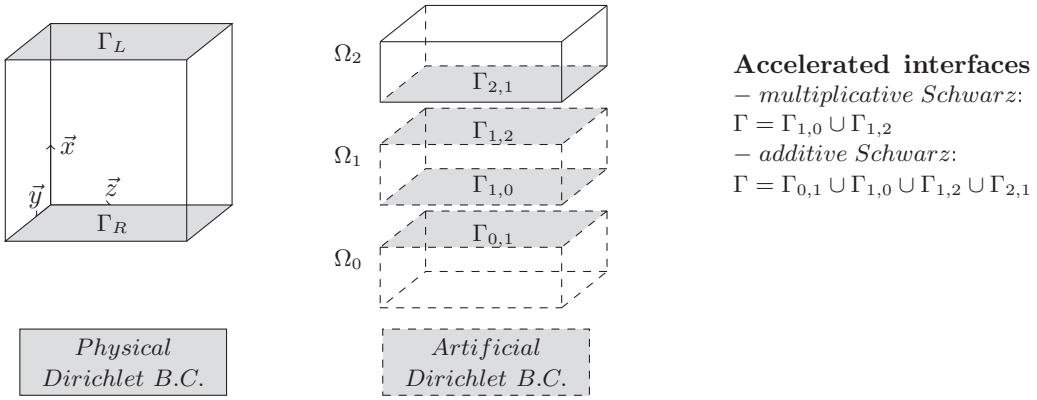


**FIGURE *3.3* –** *Exemple de découpage 1D d'un domaine 3D en trois sous-domaines.*

dont celui obtenu à l'aide du logiciel H2OLAB [1] qui est aléatoire et généré suivant une loi log-normale $Y = log(K)$ définie par sa moyenne $m_Y = 0$ ici et sa fonction de covariance :

$$C_Y(x, y, z) = \sigma^2 exp\left(-\left(\left(\frac{x}{\lambda_x}\right)^2 + \left(\frac{y}{\lambda_y}\right)^2 + \left(\frac{z}{\lambda_z}\right)^2\right)^{\frac{1}{2}}\right) \tag{3.38}$$

où $\sigma$ est l'écart-type du log de la conductivité hydraulique et $\lambda_x$, $\lambda_y$ et $\lambda_z$ sont les longueurs de corrélation dans chaque direction. Plus de détails sur la génération de champs de perméabilité peuvent être trouvés dans [124, 104]. On se limitera ici au cas d'un milieu isotrope : $\lambda = \lambda_x = \lambda_y = \lambda_z$. Ainsi, la fonction de covariance devient $C_y(r) = \sigma^2(-r/\lambda)$, où $r$ est la distance entre deux points. Typiquement, le conditionnement du système linéaire sera de l'ordre de $K_{max}/K_{min}$ où $K_{min}$ et $K_{max}$ sont les valeurs extrêmes du champ de perméabilité. Les paramètres $\sigma$ et $\lambda$ jouent un rôle sur la raideur du système linéaire à résoudre, et donc aussi sur la convergence de la méthode de Schwarz. Un exemple de coupe de trois champs de perméabilité d'un domaine de taille $128^3$ est donné en figure 3.4. Cet exemple permet de visualiser l'effet des paramètres $\lambda$ et $\sigma$ sur le champ de perméabilité puisque la même graine a été utilisée pour la génération des nombres aléatoires. On voit que $\lambda$ influe sur la raideur des variations, et que $\sigma$ influe sur l'échelle de grandeur des valeurs.

On va illustrer l'effet de ces paramètres sur la convergence de la méthode d'Aitken-Schwarz. Les graphiques présentés ici ont été obtenus avec la version multiplicative de la méthode dans le cas d'un découpage en deux sous-domaines. Dans ce cas, une seule interface est accélérée, et donc les méthodes de construction locale et globale de la base $U$ sont équivalentes.

La figure 3.5 montre la dégradation du taux de convergence de la méthode de Schwarz lorsque la valeur de $\sigma$ est augmentée. L'efficacité de l'accélération d'Aitken est également

[1] `http://h2olab.inria.fr`

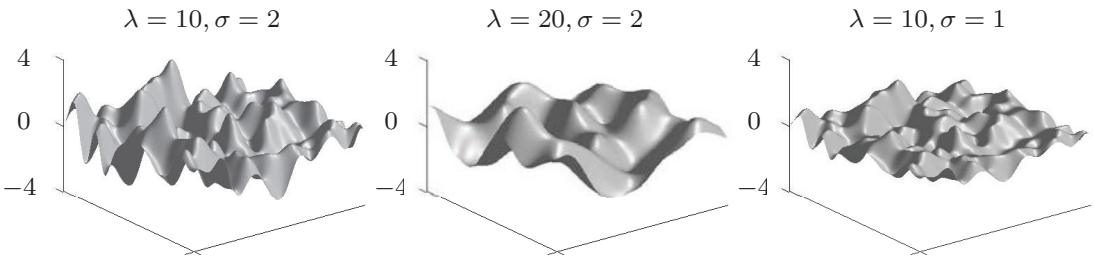


**FIGURE 3.4 –** *Valeurs de la perméabilité en échelle log10 pour différents paramètres sur une coupe à $z = 1$ d'un champ de taille $128^3$.*

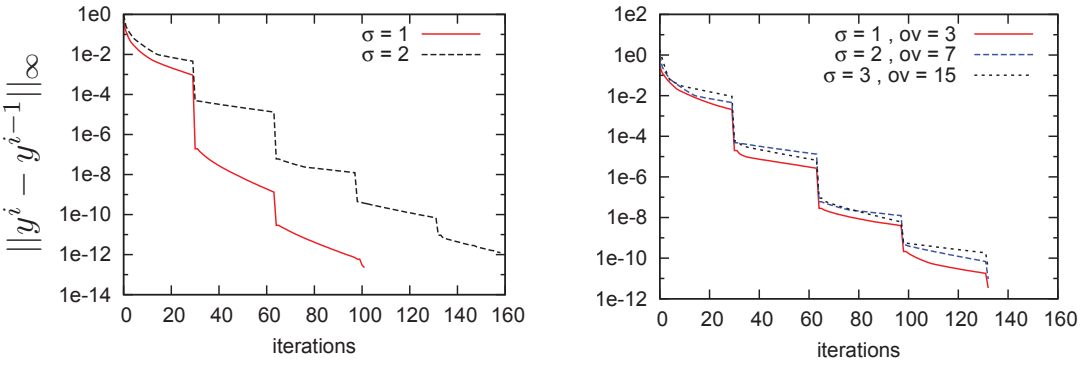


**FIGURE 3.5 –** *Norme de la différence de deux traces successives (notées $y$) en échelle log10. Sur le graphique de gauche, le recouvrement est 7 et sur celui de droite, l'augmentation de $\sigma$ est ici compensée par l'augmentation du recouvrement. Domaine de taille $310 \times 310 \times 248$ décomposé en deux sous-domaines. Algorithme de Schwarz multiplicatif.*

réduite. Heureusement, cette dégradation du taux de convergence peut ici être compensée par l'augmentation de la taille du recouvrement. Ainsi, les convergences pour $\sigma = 1, 2, 3$ dans la partie droite de la figure 3.5 sont presque identiques. L'augmentation de la zone de recouvrement implique cependant une augmentation de la taille des problèmes locaux, et donc de la charge par processeur.

L'influence du paramètre $\lambda$ est très différente : comme le montre la figure 3.6, le nombre total d'itérations de Schwarz nécessaires pour atteindre la convergence diminue lorsque $\lambda$ augmente. Ceci peut s'expliquer par le fait que la solution sur un sous-domaine dépend moins de celle de ses sous-domaines voisins lorsque $\lambda$ est petit. Cependant, le coût de chaque itération risque d'augmenter considérablement lorsque la valeur de $\lambda$ diminue. Les graphiques ont été obtenus en utilisant un gradient conjugué flexible préconditionné par un multigrille algébrique AGMG [100] pour résoudre les problèmes locaux. Le coût d'une itération de Schwarz est environ deux fois plus important lorsque $\lambda = 4$ que lorsque $\lambda = 20$.

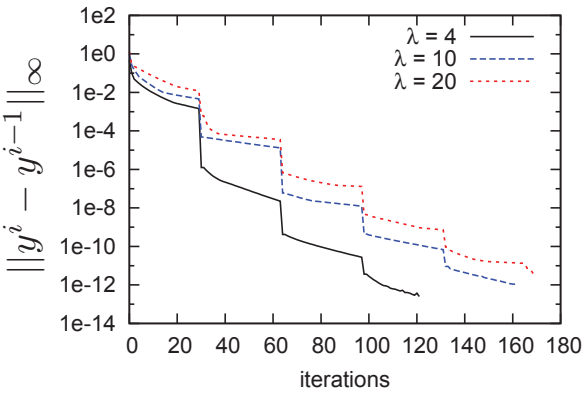


**FIGURE 3.6 –** *Norme de la différence de deux traces successives pour* $\lambda = \{4, 10, 20\}$, $\sigma = 2$. *Domaine de taille* $310 \times 310 \times 248$ *décomposé en deux sous-domaines. Algorithme de Schwarz multiplicatif.*

Notons également que la tolérance à laquelle sont effectuées les résolutions locales aura une influence sur la décomposition en valeurs singulières, et donc sur le nombre de modes gardés. La figure 3.7 montre la décroissance des valeurs singulières d'une matrice composée de 20 traces pour différents critères d'arrêt de la méthode de Krylov utilisée en solveur local.

### 3.4.2 Développement d'un logiciel utilisant deux niveaux de parallélisme

La manière la plus classique d'implémenter la méthode de Schwarz en parallèle est d'utiliser un processeur par sous-domaine. Même si ce choix peut parfois convenir lorsque la méthode de Schwarz est utilisée en tant que préconditionneur, il ne permettra pas de réali-

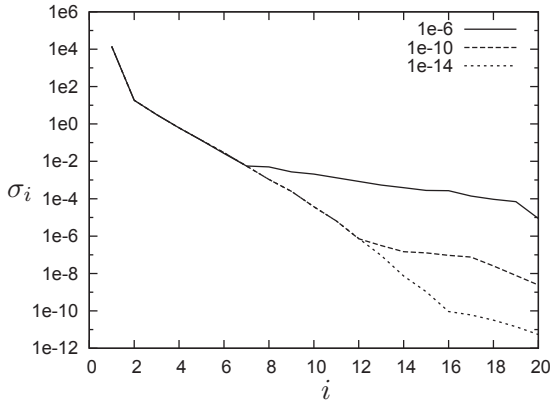


**FIGURE *3.7*** – *Valeurs singulières en échelle log10 de la matrice composée de 20 traces d'une méthode de Schwarz multiplicatif. Les tolérances relatives de la méthode de Krylov sont* $\epsilon = \{10^{-6}, 10^{10}, 10^{-14}\}$*. Domaine de taille* $420 \times 420 \times 1200$ *décomposé en trois sous-domaines. Paramètres du champ de perméabilité :* $(\lambda, \sigma) = (10, 1)$*.*

ser un solveur compétitif. Par exemple, si le problème est découpé en deux (un processeur par sous-domaine), il faudra un certain nombre d'itérations (jusqu'à plusieurs dizaines pour des problèmes 3D) pour converger vers la solution. Il faudra donc résoudre un problème deux fois plus petit jusqu'à plusieurs dizaines de fois. En général, il sera donc plus efficace d'utiliser un solveur parallèle classique (méthode de Krylov parallèle, méthode multigrille, etc.) qu'un solveur de type Schwarz. Cependant, ces solveurs parallèles classiques requièrent de nombreuses communications entre les processeurs, et leur extensibilité finit par atteindre une limite. Lorsqu'on veut résoudre un problème plus grand que ce que peut résoudre efficacement un solveur classique, il est toujours possible de découper le problème à l'aide de la méthode d'Aitken-Schwarz et d'utiliser ce solveur classique pour résoudre les problèmes de chaque sous-domaine. On propose donc d'utiliser deux niveaux de parallélisme :

- Le premier niveau de parallélisme correspond à un solveur parallèle de notre choix, tel qu'un multigrille algébrique, une méthode de Krylov, une factorisation LU multi-frontale.
- La méthode d'Aitken-Schwarz assurera le couplage entre plusieurs instances du solveur parallèle choisi.

Une illustration de l'utilisation de deux niveaux de parallélisme par une méthode de Schwarz est donnée en figure 3.8. La méthode de Schwarz ne requiert que des communications entre les processeurs aux interfaces des sous-domaines, et les messages envoyés sont de la taille de l'interface. Dans le cas d'une répartition régulière des processeurs aux interfaces, comme sur la figure 3.8, chaque processeur ne communique qu'avec un seul autre processeur lors de l'échange des conditions aux limites. Il a été choisi d'implémenter l'accélération d'Aitken en séquentiel, c'est-à-dire que le processeur $0$ de chaque interface récupère toutes

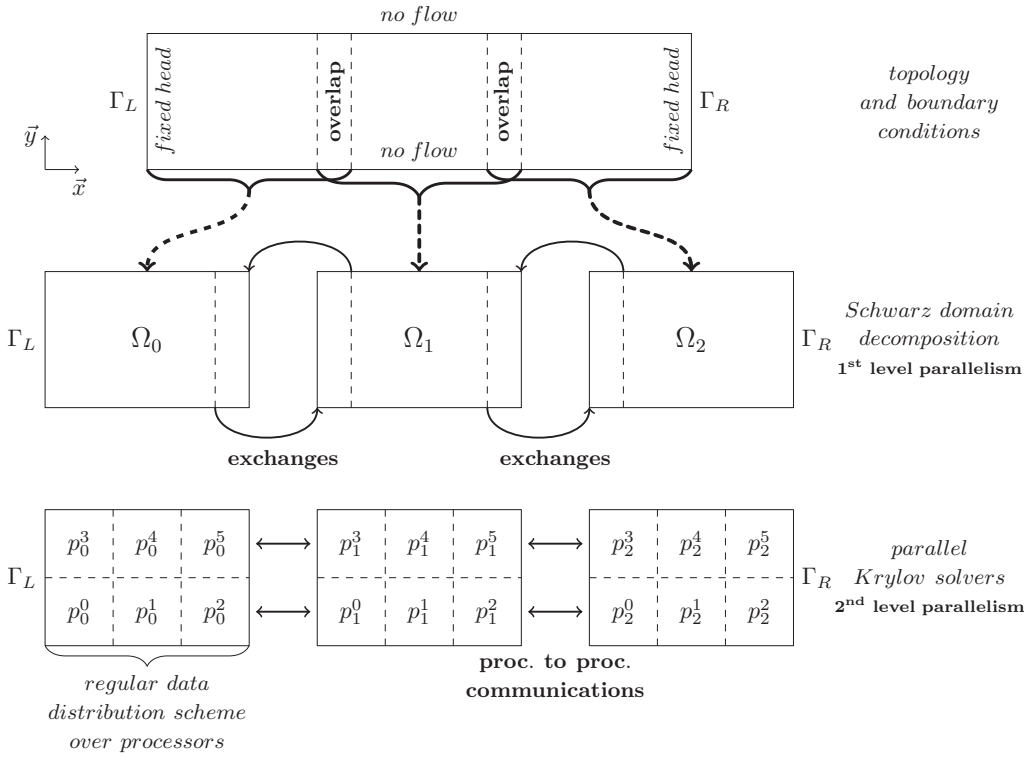


**FIGURE 3.8 –** *Exemple de l'utilisation des deux niveaux de parallélisme pour une version 2D du problème* (3.37)*. La figure du milieu montre la décomposition de domaine et les échanges qu'elle implique, la partie basse représente la distribution des données sur 18 processeurs.*

les traces du Schwarz de l'interface pour en calculer la décomposition en valeurs singulières. L'accélération dans un espace de très petite dimension s'effectuera sur le processeur 0. Même si cela constitue un goulot d'étranglement en termes de communications, cela s'est avéré plus performant en pratique pour les problèmes considérés. Les raisons sont les suivantes :

- Le temps de calcul de l'accélération d'Aitken, même sur un processeur, reste très inférieur à celui d'une itération de Schwarz.
- Les méthodes de décomposition en valeurs singulières parallèles [77] offrent une précision relative des calculs inférieure à celles des méthodes séquentielles. Les vecteurs singuliers associés à de petites valeurs singulières sont donc mieux approchés par un algorithme séquentiel, et permettent d'obtenir une meilleure accélération d'Aitken.

Les détails concernant l'implémentation MPI sont donnés en annexe B.3 p.150.

### 3.4.3 Mise à jour de la décomposition en valeurs singulières

On teste ici la mise à jour de la SVD sur la méthode $A-S$ multiplicative. Le maillage est de taille $336^2 \times 804$ (soit $91 \times 10^6$ inconnues). Le domaine est découpé en quatre sous-domaines, et le problème est résolu avec 384 cœurs. Les paramètres du champ de perméabilité sont $(\lambda, \sigma) = (10, 2)$. On compare trois approches :

- La méthode classique, où le nombre de traces est fixé à 15 entre chaque accélération.
- Le nombre de traces est choisi de manière adaptative : l'accélération est effectuée dès que l'ajout de $4$ traces n'apporte plus beaucoup d'informations, c'est-à-dire lorsque l'ajout de $4$ traces n'augmente pas le nombre de valeurs singulières supérieures à $\sigma_1 \times 10^{-12}$. La décomposition en valeurs singulières est recalculée entièrement toutes les $4$ itérations avec la fonction `DGESVD` de `Lapack` (en réalité on utilise l'implémentation d'Intel (MKL) de la bibliothèque `Lapack`).
- On choisit également le nombre de traces de manière adaptative, en utilisant la mise à jour de la SVD.

La comparaison est donnée en figure 3.9 : on constate que choisir le nombre de traces de manière adaptative permet d'économiser des itérations de Schwarz lorsque la SVD est recalculée à chaque ajout de traces. La taille maximale de la matrice $U$ est d'environ $165$Mo lorsque la fonction `DGESVD` est utilisé pour calculer la SVD à chaque ajout de traces, et de $70$Mo lorsque la SVD est mise à jour petit à petit. La perte d'information lors de la mise à jour de la SVD conduit à une dégradation de la qualité de l'accélération. Cette économie de mémoire est à relativiser dans la mesure où le stockage de la matrice est de l'ordre de $5$Go. La taille de l'interface est ici trop petite par rapport au nombre total de points, pour que le gain en mémoire apporté par la mise à jour de la SVD soit significatif.

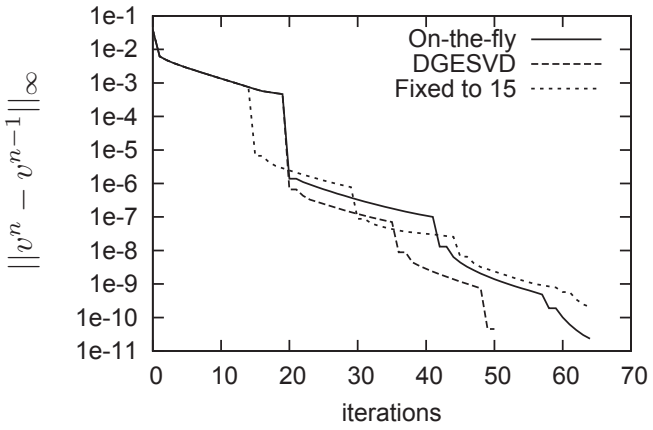


**FIGURE *3.9* –** *Comparaison de la convergence en échelle log10 : norme de la différence entre deux traces (notées $v$) successives. 'DGESVD' : le nombre de traces est choisi de manière adaptative, mais la SVD est recalculée avec* `Lapack` *entièrement à chaque fois. 'On-the-fly' : nombre de traces choisi de manière adaptative, SVD mise à jour. 'Fixed to 15' : l'accélération est faite toutes les 15 itérations de Schwarz.*

### 3.4.4 Comparaison des approches Aitken-Schwarz classique et Aitken-Schwarz creuse

Dans les tests suivants, on ne considérera pas la version avec préconditionneur de l'algorithme d'Aitken-Schwarz. Pour mettre en évidence le bruit numérique provoqué par l'accélération, on résout le problème suivant :

- Le domaine est $\Omega = [0, 1] \times [0, 1] \times [0, 30]$.
- Le maillage est de taille $64 \times 64 \times 1920$ points répartis sur $120$ cœurs.
- Le domaine divisé en $5$ sous-domaines avec un recouvrement de $4$ points.
- Le problème résolu est un Poisson homogène : $K(x, y, z) = 1$.
- Les conditions limites de Dirichlet sont de $1.0$ à gauche et $10.0$ à droite.
- Chaque sous-problème est résolu avec une précision relative de $10^{-8}$ par une méthode de Krylov préconditionnée.
- Le problème global est résolu par une méthode de Schwarz additive, en partant de la solution $x^0 = 0$ avec une tolérance absolue de $10^{-7}$.

Ce problème 3D est proche d'un problème 1D, ainsi on s'attend à ce que les solutions successives soient constantes dans le plan $xy$. En réalité, à cause de la résolution approchée des sous-problèmes, la solution n'est pas exactement constante sur les interfaces artificielles. L'écart-type de la solution (les valeurs ont été centrées en 0) aux interfaces est tracé en figure

3.10. Cela montre clairement que l'accélération d'Aitken produit du bruit sur les interfaces dans le cas de l'approche classique. Ce bruit est ensuite amorti par les itérations de Schwarz. Dans le cas de l'approche creuse, l'écart-type reste inférieur à $10^{-8}$ après la première accélération. La figure 3.11 permet de comparer le résidu dans les deux cas : la solution est obtenue

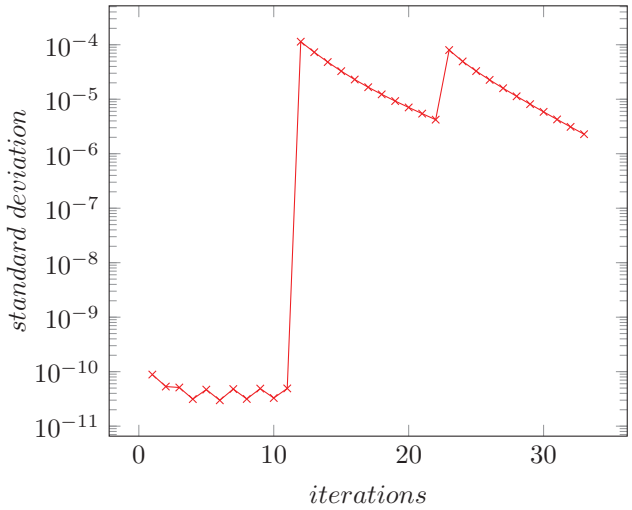


**FIGURE *3.10 –*** *Écart-type de la solution aux interfaces pour un problème de Poisson homogène en fonction des itérations de la méthode de Schwarz additive.*

à la tolérance désirée en calculant l'accélération avec l'approche creuse. Dans le cas de l'approche classique, la première accélération est moins performante et la seconde est inefficace. Cette inefficacité s'explique par le fait que le bruit n'a pas été suffisamment amorti avant de calculer la deuxième accélération : l'espace dans lequel est calculé la deuxième accélération ne permet pas de représenter correctement la solution exacte. Ce bruit numérique provoqué par l'accélération trouve son origine dans le calcul de la SVD. En effet, les traces globales sont constantes (à la tolérance de la méthode de Krylov près) par morceaux, mais les vecteurs colonnes de la matrice $U$ sont entachés d'un bruit numérique. De plus, ce bruit numérique est amplifié par le fait que l'approximation de $P$ de l'approche globale $A-S$ est dense. Lorsqu'on utilise l'approche creuse $SA-S$, la structure analytique de $P$ est respectée.

On considère ici un domaine $\Omega = [0,1] \times [0,1] \times [0,15]$ discrétisé en $128 \times 128 \times 1920$ points. La fonction $K$ du champ de perméabilité est la suivante :

$$K(x,y,z) = 10^{2 \times sin(\pi x) \times sin(\pi y) \times sin(\pi z)}. \tag{3.39}$$

La table 3.1 donne le nombre d'itérations de Schwarz ainsi que les temps de calcul en secondes. Les temps de calcul sont mesurés à partir du début de la première itération de

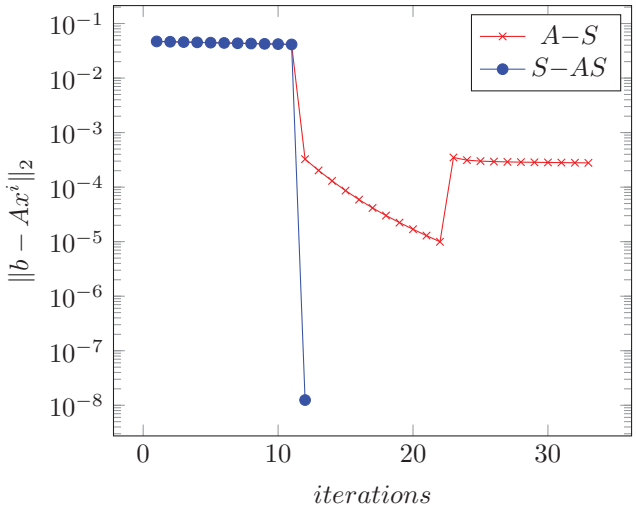


**FIGURE *3.11*** – *Résidus en fonction des itérations de la méthode additive de Schwarz pour un problème de Poisson homogène.*

Schwarz et jusqu'à la dernière. Ils excluent donc la phase d'assemblage et de distribution de la matrice, ainsi que les écritures dans les fichiers. On remarque que l'accélération est systématiquement meilleure dans le cas où est elle est calculée par l'approche creuse. De plus, cette différence s'accroît avec le nombre de sous-domaines. Cela s'explique par le fait que plus il y a de sous-domaines, plus la matrice $P$ analytique a une structure creuse. Le nombre de traces, c'est-à-dire le nombre d'itérations de Schwarz entre deux accélérations, a été arbitrairement choisi pour être supérieur au nombre de sous-domaines. Remarquons également

**TABLE *3.1*** – *Nombre d'itérations de la méthode de Schwarz additive et temps de calcul*

| Sous-domaines | Traces | $A-S$ | $SA-S$ |
|---|---|---|---|
| 2 | 9 | 18 (63.9$s$) | 9 (42.7$s$) |
| 5 | 9 | 120 (353.8$s$) | 18 (61.0$s$) |
| 10 | 19 | n.c | 19 (66.3$s$) |
| 15 | 19 | n.c | 19 (65.1$s$) |

$A-S$ : Aitken-Schwarz classique, $SA-S$ : *sparse Aitken-Schwarz*
Le maillage est réparti sur 120 cœurs.
C.L de Dirichlet de 1.0 à gauche et 10.0 à droite.
Sous-problèmes résolus par GMRES préconditionné par Hypre avec une tolérance relative de $10^{-10}$.
Tolérance relative de la méthode de Schwarz de $10^{-8}$ sur le résidu.
n.c : non-convergence après 200 itérations de Schwarz.

que les meilleurs temps de calcul sont obtenus pour 2 sous-domaines. En effet, le solveur

GMRES préconditionné par Hypre est très performant pour résoudre ce problème de taille modeste avec un champ de perméabilité relativement régulier. Ainsi, la première résolution des sous-problèmes a demandé $14.6$ secondes dans le cas de deux sous-domaines, et $11.2$ secondes dans le cas de $15$ sous-domaines. Autrement dit, le temps de résolution est multiplié par $1.3$ pour résoudre un problème $7.5$ fois plus grand. Il est important de noter que la première résolution est également la plus coûteuse car elle comprend la phase de construction des niveaux de grille.

### 3.4.5 Préconditionnement de la méthode de Richardson

On illustre ici le gain apporté par la méthode proposée en algorithme 9 qui consiste à préconditionner les itérations de Richardson après chaque accélération. Le domaine est ici un pavé de taille $256 \times 256 \times 512$ discrétisé avec un pas de $1$. Les paramètres du champ de perméabilité sont $(\lambda, \sigma) = (10, 1)$. Le domaine est divisé en quatre sous-domaines de même taille, auquel on ajoute un recouvrement de $4$ points. Au total, 128 cœurs sont utilisés (32 cœurs par sous-domaine).

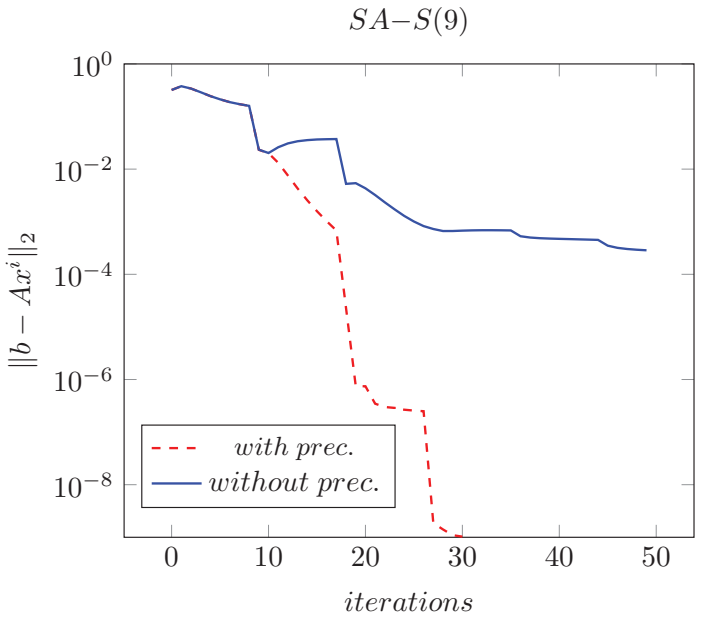


**FIGURE *3.12* –** *Résidus pour Aitken-Schwarz creux avec et sans préconditionnement.*

Une comparaison des résidus avec et sans préconditionnement est proposée en figure 3.12 et figure 3.13. L'accélération d'Aitken est construite à partir de neuf traces pour la figure 3.12. On constate que le préconditionneur améliore significativement la convergence,

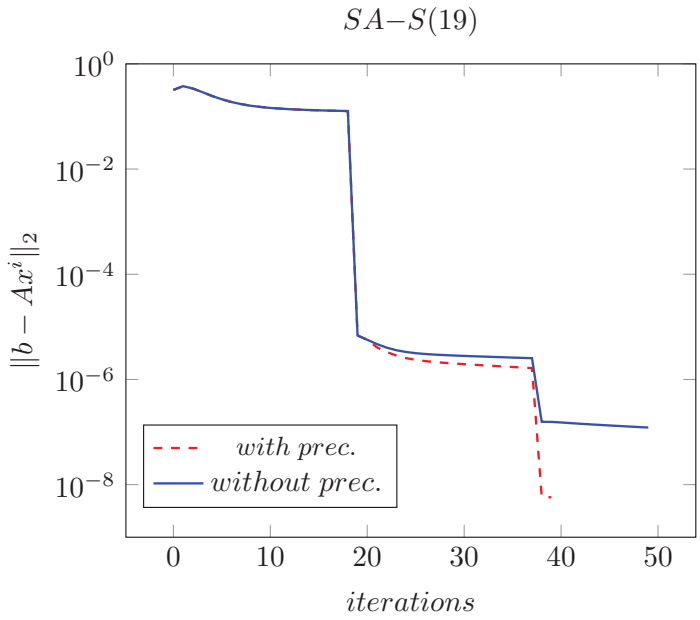


**FIGURE *3.13 –*** *Résidus pour Aitken-Schwarz creux avec et sans préconditionnement.*

par exemple en regardant la pente des courbes après la première accélération. Finalement, les neufs traces ne semblent pas suffisantes pour accélérer efficacement la convergence dans le cas non préconditionné, alors qu'elles sont suffisantes dans le cas préconditionné. Notons tout de même que l'application du préconditionneur a un coût qui est relativement faible ici : les $30$ itérations sont calculées en $519$ secondes avec préconditionneur, et $513$ secondes sans préconditionneur. Cependant, l'application du préconditionneur nécessite une opération de réduction commune à tous les sous-domaines, ce qui limite l'extensibilité du code. L'accélération d'Aitken et le préconditionneur sont calculés à partir de 19 traces sur la figure 3.13. Ces 19 traces permettent d'accélérer efficacement la convergence dans les deux cas.

### 3.4.6 Extensibilité

On considère ici uniquement la méthode d'Aitken-Schwarz creuse sans préconditionneur. La taille d'un sous-domaine est fixée à $512 \times 512 \times 256$, et le nombre de sous-domaines est augmenté. On effectue $9$ itérations de Schwarz avant l'accélération, puis une seule après l'accélération. La table 3.2 présente les temps de calcul et leur répartition. Le nombre d'itérations de Schwarz est de $10$ quelle que soit la taille du domaine considéré, cela signifie que la tolérance $\|b - Ax\|_2/\|b\|_2 < 10^{-5}$ est obtenue après la première accélération. Le temps de calcul pour $2$ domaines aurait pu être légèrement réduit en considérant seulement $7$ où $8$ itérations

**TABLE 3.2 –** *Répartition des coûts de calcul pour $SA{-}S(9)$ sur le cluster JADE (SGI Altix ICE 8200) du CINES*

| Sous-domaines | Cœurs | Temps (s) | Rés. locales | Aitken | Échanges | Reste |
|---|---|---|---|---|---|---|
| 2 | 512 | 752 | 99.444% | 0.123% | $1.54 \times 10^{-3}$% | 0.431% |
| 4 | 1024 | 811 | 99.051% | 0.186% | $2.86 \times 10^{-3}$% | 0.761% |
| 8 | 2048 | 828 | 98.548% | 0.168% | $4.35 \times 10^{-3}$% | 1.280% |
| 16 | 4096 | 817 | 98.063% | 0.208% | $5.00 \times 10^{-3}$% | 1.724% |

C.L de Dirichlet de 1.0 à gauche et 10.0 à droite.
Recouvrement de 8 points entre les sous-domaines.
Sous-problèmes résolus par FGMRES préconditionné par Hypre avec une tolérance relative de $10^{-12}$.
Tolérance relative de la méthode de Schwarz de $10^{-5}$ sur le résidu.

de Schwarz avant l'accélération. Dans les cas à 4, 8 et 16 sous-domaines, le nombre optimal de traces avant l'accélération pour converger le plus rapidement à la tolérance désirée est bien de 9. Voici les paramètres :

- Seulement les temps de calculs de la résolution sont mesurés. En particulier, le temps d'assemblage de la matrice est exclu de ces mesures. Les valeurs présentées dans ce tableau sont des moyennes sur deux essais (uniquement deux pour des raisons de disponibilité des ressources de calcul) . Un écart maximum de 12 secondes a été observé entre ces deux essais. C'est essentiellement le temps de résolution qui varie. Cela peut s'expliquer par le fait que la méthode de Krylov FGMRES est préconditionnée par le multigrille algébrique BoomerAMG de Hypre qui demande beaucoup de communications. Hors, la durée de ces communications peut varier en fonction de la charge du réseau.
- Les temps de calcul sont très largement composés des résolutions locales. Notons cependant que le nombre d'itérations de Krylov peut varier d'un sous-domaine à l'autre. Le temps des résolutions locales présentées dans la table 3.2 inclut donc les éventuels temps d'attente. Le temps des résolutions inclut également la construction des grilles grossières, qui n'est effectuée qu'à la première résolution.
- La mesure de temps de l'accélération d'Aitken comprend le calcul des SVD par interfaces, le calcul de la solution accélérée sur le processeur 0 et sa redistribution aux interfaces. La proportion du temps de calcul de l'accélération augmente avec le nombre de sous-domaines. En effet, lorsqu'on utilise $AS{-}S$, la dimension de l'espace dans lequel est effectuée l'accélération augmente en fonction du nombre de sous-domaines.
- Le temps des échanges de la méthode de Schwarz reste négligeable, même s'il augmente avec le nombre de sous-domaines. Notons qu'ici, le nombre de messages varie lorsque le nombre de sous-domaines varie, mais la taille des messages est constante.
- Le temps restant est principalement composé du test de convergence, mais il inclut également les entrées et sorties.

## 3.5 Conclusions

L'accélération d'Aitken d'une suite de vecteurs peut être appliquée aux méthodes de décomposition de domaine de type Schwarz car celles-ci peuvent s'écrire sous la forme d'itérations de Richardson. Cependant, la formule d'Aitken exacte peut être extrêmement coûteuse à construire dans le cas où les interfaces artificielles sont composées de nombreux points. C'est notamment le cas des problèmes 3D. Il a été proposé d'effectuer une approximation de l'accélération d'Aitken dans un sous-espace particulier. Il est donc possible d'effectuer un certain nombre d'itérations de Schwarz, et d'effectuer l'accélération d'Aitken dans un sous-espace engendré par les traces successives. Le processus de Schwarz restreint aux interfaces artificielles peut être vu comme un processus de Richardson. Dans ce cas, les solutions successives sont dans l'espace de Krylov, tout comme la solution accélérée. Ainsi, la méthode Aitken-Schwarz classique où l'accélération est effectuée à partir de $q+1$ traces convergera au mieux comme la méthode GMRES avec un *restart* de $q+1$. Ceci dit, $A-S$ nécessite moins de communications globales : l'orthogonalisation est effectuée toutes les $q+1$ itérations pour $A-S$ et toutes les itérations pour GMRES. Lors de l'accélération d'Aitken, on effectue une approximation de faible rang de l'opérateur de propagation d'erreur. Cette approximation peut être réutilisée après la première accélération pour préconditionner les nouvelles itérations de Richardson ou de GMRES. Dans ce cas, le préconditionneur devient très similaire aux préconditionneurs basés sur la déflation. Il a également été proposé d'effectuer l'accélération dans un sous-espace plus grand que l'espace de Krylov. Le calcul d'une base de ce sous-espace est effectué en considérant les interfaces artificielles séparément. Cela permet de construire une approximation de faible rang de l'opérateur de transfert d'erreur qui respecte sa structure : l'approximation de l'opérateur de transfert d'erreur respecte les blocs de $0$ de l'opérateur analytique. Autrement dit, les interfaces indépendantes dans l'espace physique restent indépendantes dans l'espace de l'accélération. Cette méthode a été appelée $SA-S$ pour *Sparse Aitken-Schwarz*. La méthode $SA-S$ peut converger plus rapidement que la méthode GMRES, puisque l'accélération s'effectue dans un espace différent de l'espace de Krylov. Cette méthode s'est avérée efficace sur un problème 3D d'écoulement dans un milieu poreux généré de manière aléatoire.

# Chapitre 4

# Mise à jour du préconditionneur *Restricted Additive Schwarz*

## Sommaire

La résolution de problèmes transitoires non linéaires s'effectue principalement à l'aide de méthodes implicites. Ces méthodes implicites demandent de résoudre un problème non linéaire à chaque pas de temps, ce qui est généralement fait par une méthode de Newton. Chaque itération de la méthode de Newton implique la résolution d'un système linéaire. Il faut donc résoudre plusieurs systèmes linéaires pour chaque pas de temps. La résolution de ces nombreux systèmes linéaires est en général la partie la plus consommatrice en temps de la simulation numérique. Afin d'économiser du temps de calcul et de la mémoire, ces systèmes linéaires peuvent être résolus de manière inexacte [47] sans mettre en péril le taux de convergence de la méthode de Newton. Ces méthodes sont connues sous le nom d'*inexact Newton methods*. On s'intéresse ici à une classe particulière de ces méthodes, dite de Newton-Krylov, où les systèmes linéaires sont résolus avec une méthode de Krylov. Une introduction à ces méthodes est donnée dans [79, Chapter 3] et leur convergence est étudiée dans [108]. Il est donc crucial d'accélérer la convergence, c'est-à-dire de réduire le nombre d'itérations de la méthode de Krylov en fournissant un préconditionneur. Un équilibre doit être trouvé entre la capacité du préconditionneur à réduire le nombre d'itérations et son coût de calcul. Ce coût de calcul comprend la mise en place du préconditionneur et le coût de son application à des vecteurs. La simulation numérique est donc principalement composée de la résolution d'une suite de systèmes linéaires où l'opérateur linéaire est la matrice jacobienne. En général, cette matrice jacobienne ne change pas de manière radicale d'un système à l'autre : les coefficients varient peu, voire pas du tout. De nombreuses méthodes ont été proposées pour optimiser la résolution d'une suite de systèmes linéaires dont l'opérateur ou le second membre varient légèrement. Toutes ces méthodes consistent à réutiliser des calculs faits lors des résolutions précédentes pour résoudre le prochain système. Lorsqu'on utilise une méthode de Krylov, l'idée la plus naturelle est de réutiliser l'espace de Krylov. Un aperçu de ces techniques est donné dans [105]. Remarquons néanmoins que l'une de ces techniques, proposée par ERHEL, BURRAGE et POHL dans [49], consiste à utiliser l'information contenue dans l'espace de Krylov pour mettre à jour le préconditionneur après un *restart* de GMRES. Lorsque la matrice jacobienne est coûteuse à construire, il peut être avantageux de la réutiliser durant plusieurs itérations de Newton. Cette méthode est appelée Newton simplifié dans [43, Chapter 1.3]. Dans ce cas particulier, l'opérateur linéaire est le même, et l'on pourra utiliser les méthodes basées sur la déflation.

Pour économiser du temps de calcul, on discutera ici d'une autre approche : puisque les opérateurs linéaires varient généralement peu, on peut réutiliser la même matrice de préconditionnement pendant plusieurs itérations de Newton. On peut donc penser à mettre à jour le préconditionneur plutôt que de le recalculer entièrement : cela peut être des mises à jour suivant la méthode de quasi-Newton, comme cela est discuté dans [17, 16, 13], ou une mise à jour directe des préconditionneurs stockés sous la forme de factorisations de matrices. Ainsi, la mise à jour du préconditionneur AINV dans le cas d'une séquence de matrice de la forme $A_k = A + \alpha_k I$ a été proposée dans [7]. Dans [116], il est proposé une formule

algébrique pour mettre à jour le préconditionneur ILU. Notons que cette mise à jour demande de disposer de la différence entre les matrices successives. Cette idée sera étendue dans [18] au préconditionneur bloc ILU, puis dans [46] aux méthodes *Jacobian-free*.

Dans la section 4.1, on donnera un aperçu des méthodes de Newton et de quasi-Newton utilisées dans la suite. On proposera ensuite deux façons de mettre à jour le préconditionneur RAS où les opérateurs sont stockés sous la forme de factorisations LU. Premièrement, ce préconditionneur sera mis à jour en utilisant la méthode de Broyden. Cette première partie reprendra les travaux [13, 14]. On proposera ensuite de ne recalculer que partiellement le préconditionneur RAS. Cette mise à jour partielle du préconditionneur sera faite de manière asynchrone [15, 12] ; autrement dit, les factorisations LU seront calculées par des processus additionnels indépendamment de l'intégrateur en temps.

## 4.1 Introduction : la résolution de problèmes non linéaires

Les méthodes de Newton et de Broyden sont passées en revue dans cette section, On détaillera en particulier l'estimation de la matrice jacobienne par différences finies en utilisant le coloriage. Ce coloriage sera systématiquement utilisé lors des tests numériques car, sans celui-ci, les temps de calcul de la matrice jacobienne peuvent être prohibitifs.

### 4.1.1 Méthodes de Newton

#### 4.1.1.1 Algorithme et convergence

La méthode de Newton pour la résolution de $F(x) = 0$ avec $x, F(x) \in \mathbb{R}^n$ est une méthode itérative dont les itérations sont données par l'équation (4.1).

$$x_{i+1} = x_i - F'(x_i)^{-1} F(x_i) \tag{4.1}$$

À chaque itération de l'équation (4.1), on résout le problème linéarisé par un développement de Taylor autour de $x_i$. Si l'on note $x^*$ la solution exacte de $F(x) = 0$ et $e_i = x - x^*$ l'erreur à l'itération $i$, alors on a le théorème suivant :

**Théorème 6.** *Considérons que :*

- *le problème $F(x) = 0$ admet une solution $x^*$.*
- *$F' : R^n \to R^{n \times n}$ est Lipschitz-continue et la constante de Lipschitz est $\gamma$.*
- *la matrice $F'(x^*)$ est non singulière.*

*Alors il existe une boule $\mathcal{B}(\delta)$ de rayon $\delta$ et centrée en $x^*$ telle que si $x_0 \in \mathcal{B}(\delta)$ alors la suite des $x_i$ donnée par l'équation* (4.1) *converge quadratiquement vers $x^*$. C'est-à-dire que l'on a :*

$$\|e_{i+1}\| \leq K \|e_i\|^2 \tag{4.2}$$

*avec $K = \gamma \|F'(x^*)^{-1}\|$.*

Une démonstration de ce théorème est disponible dans [78, chapitre 5].

Notons que l'équation (4.1) requiert la résolution d'un système linéaire. La méthode de Newton sera donc en pratique composée de deux étapes :

- Résolution du système linéaire : $J(x_i)\Delta x_i = -F(x_i)$
- Mise à jour de la solution : $x_{i+1} = x_i + \Delta x_i$

#### 4.1.1.2 Méthodes de globalisation de la convergence

La méthode de Newton converge donc quadratiquement au voisinage de la solution. En pratique, il peut être difficile de trouver un $x_0$ suffisamment proche de la solution pour que le théorème précédent s'applique. Les méthodes de globalisation ont pour but de permettre la convergence même lorsque $x_0$ est loin de la solution exacte. Il existe trois principaux types de méthodes de globalisation :

- Les méthodes basées sur la continuation (ou homotopie) consistent à modifier la fonction $F$ en ajoutant un paramètre $\tau$ tel que $F(x_0, \tau) = 0$ pour $\tau = 1$ et $F(x^*, \tau) = 0$ pour $\tau = 0$. Il pourra être possible de trouver $x^*$ en résolvant plusieurs problèmes linéaires pour une suite de valeurs de $\tau$ qui converge vers $0$. Ces méthodes de continuation sont détaillées dans [101, chapitre 7.5].
- La méthode de Newton classique est basée sur la linéarisation de la fonction autour de $x^*$. Lorsque $x^*$ est loin de $x^0$, cette linéarisation pourra être valable autour de $x_0$ mais pas autour de $x^*$. Les méthodes basées sur la région de confiance calculent l'incrément $\Delta x_i$ minimisant $\|F(x_i) - J(x_i)\Delta x_i\|$ sous la contrainte $\|\Delta x_i\| \leq \delta$. Ainsi, le paramètre $\delta > 0$ quantifiera la région autour de $x_i$ pour laquelle la linéarisation est correcte. Des détails concernant cette méthode peuvent être trouvés dans [43, chapitre 3.1].
- La méthode de recherche en ligne (*linesearch*) : on incrémente la solution par la formule $x_{i+1} = x_i + \lambda\Delta x_i$ où $\lambda$ est un paramètre minimisant $\|F(x_i + \lambda\Delta x_i)\|_2^2$. Plus de détails peuvent être trouvés dans [78, chapitre 8].

La convergence des méthodes de recherche en ligne et de région de confiance ne sera en général pas quadratique mais deviendra quadratique lorsque $x_i$ sera suffisamment proche de $x^*$.

Lors des expérimentations numériques de ce manuscrit, dans le cas où une méthode de globalisation sera utilisée, il s'agira de la méthode de recherche en ligne.

### 4.1.2 Estimation de la matrice jacobienne

Pour résoudre numériquement le problème non linéaire $F(x) = 0$ par une méthode de Newton, il faut disposer de la matrice jacobienne de la fonction $F$. Lorsque cette matrice jacobienne analytique n'est pas disponible, ou lorsqu'elle est trop coûteuse à calculer, on peut l'estimer à l'aide de la méthode des différences finies. Cette estimation requiert de nombreux appels à la fonction $F$, et peut donc représenter une part importante du temps de calcul.

#### 4.1.2.1 Calcul de la matrice par différences finies

On considère une fonction $F$ telle que $y = F(x)$ avec $y \in \mathbb{R}^m$ et $x \in \mathbb{R}^n$. Supposons qu'on estime la matrice jacobienne $F'(x)$ en utilisant la formule suivante :

$$F'(x)e_j \approx \frac{F(x + \epsilon e_j) - F(x)}{\epsilon}. \tag{4.3}$$

D'un point de vue informatique, il faudra donc effectuer $n + 1$ appels à la fonction $F$.

Il existe plusieurs approches permettant de diminuer le coût de la construction de la matrice jacobienne. L'ensemble de ces techniques présentées peut être trouvé dans le livre de GRIEWANK et WALTHER [71], spécialement dans le chapitre 8. La première, qui ne sera pas détaillée ici, consiste à écrire la fonction $F$ de manière à sauvegarder les calculs des valeurs intermédiaires afin de ne les recalculer que lorsque cela est nécessaire. Cette approche est appelée *sparse forward*.

Nous allons présenter ici une autre approche, qui vise à minimiser le nombre d'appels à la fonction $F$ lors de la construction de la matrice jacobienne. Cette approche, dite du coloriage, sera valable quelle que soit la façon dont est codée la fonction $F$ mais sera plus ou moins efficace selon la creusité de la matrice jacobienne. Cette méthode, aussi appelée *compression*, sera utilisée lors des tests numériques dans la suite de ce manuscrit.

L'idée principale de cette méthode est d'évaluer simultanément plusieurs colonnes de la matrice jacobienne lorsque ces colonnes correspondent à des variables indépendantes les unes des autres. Ainsi, on peut utiliser l'équation suivante :

$$F'(x)s \approx \frac{F(x + \epsilon s) - F(x)}{\epsilon} \tag{4.4}$$

où $s$ est un vecteur contenant des 0 et des 1. On définit la matrice $S = [s_1, \ldots, s_p]$, et on obtient donc la matrice $B$ qui représente la matrice jacobienne avec une compression de ses lignes :

$$B = F'(x)S \in \mathbb{R}^{m \times p}. \tag{4.5}$$

Ainsi, l'$i$ème ligne de $B$ est donnée par $b_i^T = e_i^T F'(x)^T S = (F_i'(x))^T S$.

On va maintenant chercher à minimiser le nombre de colonnes $p$ de la matrice $S$, ce qui revient à minimiser le nombre d'appels à la fonction $F$. L'idée du coloriage est de choisir les colonnes de $S$ de façon à ce que les $1$ soient associés à des variables indépendantes. On dira que ces variables indépendantes ont la même couleur, et il y aura donc $p$ couleurs différentes. Plus formellement, on dira que la variable numéro $1 \leq i \leq m$ est de la $j$ème couleur, avec $1 \leq j \leq p$ si $S_{i,j} = 1$. Le nombre minimum de colonnes de $S$ est appelé le nombre chromatique $\chi$. On peut montrer que le calcul de ce nombre est un problème NP-difficile. Cependant, il existe des méthodes heuristiques efficaces qui permettent d'obtenir un nombre de couleurs assez proche de $\chi$ (voir [65]). La première approche permettant de compresser et de reconstruire la matrice jacobienne est souvent appelée l'approche CPR pour CURTIS, POWELL et REID [36]. Elle est rapidement présentée ici :

Notons $c(j)$ la couleur de la $j$ème variable. La matrice $S$ peut être écrite comme :

$$S = \begin{bmatrix} e_{c(1)}^T \\ \vdots \\ e_{c(n)}^T \end{bmatrix} \tag{4.6}$$

où les $e_i$ sont les vecteurs unitaires de $\mathbb{R}^p$. On retrouve les coefficients non nuls de la jacobienne en utilisant :

$$J_{ij} = e_i^T J e_j = B_{ic(j)}. \tag{4.7}$$

Ainsi le cas idéal est celui où toutes les variables sont indépendantes car il y a une seule couleur :

$$F(x) = \begin{bmatrix} f_1(x_1) \\ \vdots \\ f_n(x_n) \end{bmatrix} \tag{4.8}$$

et donc :

$$\underbrace{\begin{bmatrix} f_1' & & \\ & \ddots & \\ & & f_n' \end{bmatrix}}_{J} \times \underbrace{\begin{bmatrix} 1 \\ \vdots \\ 1 \end{bmatrix}}_{S} = \underbrace{\begin{bmatrix} f_1' \\ \vdots \\ f_n' \end{bmatrix}}_{B} \tag{4.9}$$

la matrice jacobienne est donc estimée en $1 + 1 = 2$ appels à la fonction $F$ : $F'(x)s \approx \dfrac{F(x + \epsilon s) - F(x)}{\epsilon}$.

**Exemple 1.** *Si l'on considère un problème à une dimension discrétisé avec un schéma à* 3 *points, alors le nombre de couleurs nécessaires est* 3*. Ainsi, avec six inconnues, on obtient la compression suivante :*

$$\underbrace{\begin{bmatrix} b_1 & c_2 & & & & \\ a_1 & b_2 & c_3 & & & \\ & a_3 & b_3 & c_4 & & \\ & & a_3 & b_4 & c_5 & \\ & & & a_4 & b_5 & c_6 \\ & & & & a_5 & b_6 \end{bmatrix}}_{J} \times \underbrace{\begin{bmatrix} 1 & & \\ & 1 & \\ & & 1 \\ 1 & & \\ & 1 & \\ & & 1 \end{bmatrix}}_{S} = \underbrace{\begin{bmatrix} b_1 & c_2 & \\ a_1 & b_2 & c_3 \\ c_4 & a_2 & b_3 \\ b_4 & c_5 & a_6 \\ a_4 & b_5 & c_6 \\ & a_5 & b_6 \end{bmatrix}}_{B}. \tag{4.10}$$

*Ce type de compression s'étend naturellement à des problèmes de taille quelconque : une matrice jacobienne tridiagonale sera évaluée en* $3 + 1$ *appels à la fonction* $F$*, quelle que soit la taille du problème.*

Remarquons qu'à un élément non nul de la matrice jacobienne, correspond un élément non nul de la matrice $B$. Ici on a compressé les lignes de la matrice jacobienne ($JS = B$)

mais on aurait également pu compresser les colonnes, ce qui revient à compresser les lignes de la matrice transposée : $J^T W = C$. Dans les exemples précédents, l'économie réalisée est faible car les matrices sont très petites. La plupart des tests numériques présents dans ce manuscrit sont issus de la discrétisation d'équations differentielles sur un maillage, dans ce cas la méthode de compression des lignes sera efficace. Nous pouvons tout de même illustrer la méthode de compression des lignes sur le problème de Fekete de la répartition homogène de $N$ points sur une sphère. Ce problème peut être résolu par la méthode des forces qui consiste à résoudre l'EDA suivante :

$$\begin{cases} \dot{p} &= q + G^T(p)\mu \\ \dot{q} &= g(p,q) + G^T(p)\lambda \\ 0 &= G(p)q \\ 0 &= \phi(p) \end{cases} \tag{4.11}$$

avec $G = \dfrac{\partial \phi}{\partial p}$,

$$\phi_i(p) = p_{i,1}^2 + p_{i,2}^2 + p_{i,3}^2 - 1, \tag{4.12}$$

et

$$g_i(p,q) = \sum_{i \neq j} \frac{p_i - p_j}{\|p_i - p_j\|^2} - \alpha q_i. \tag{4.13}$$

Il y a donc 8N inconnues : $y^T = (p, q, \mu, \lambda) = (x, y, z, \dot{x}, \dot{y}, \dot{z}, \mu, \lambda)$. Les valeurs initiales sont $q(0) = 0$, ce qui implique $\mu(0) = 0$. Comme tous les points se repoussent, la vitesse d'un point dépend de la position de tous les points. La matrice jacobienne a donc la structure présentée en figure 4.1. La matrice jacobienne a $8N$ colonnes tandis que la matrice dont les

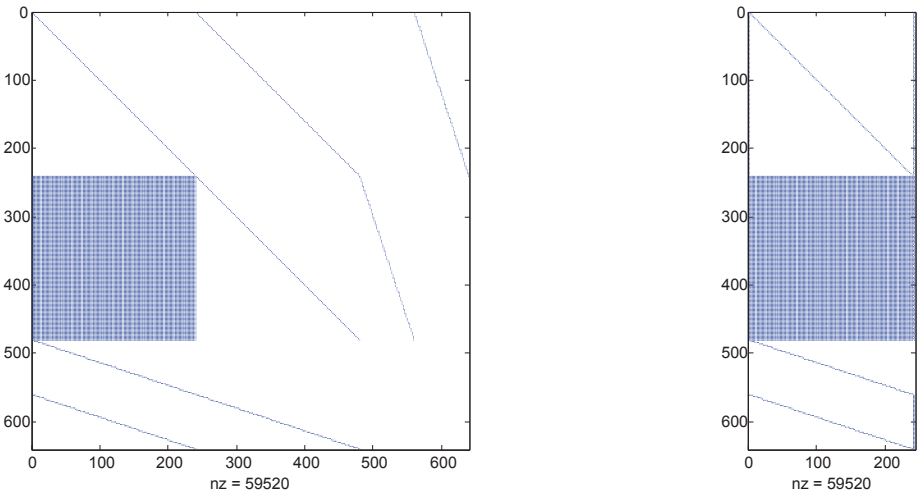


**FIGURE *4.1*** – *Structure de la matrice jacobienne de l'équation* (4.11) *non compressée (à gauche) et compressée (à droite).*

lignes ont été compressées aura $3N + 4$ colonnes. Une illustration de la matrice compressée

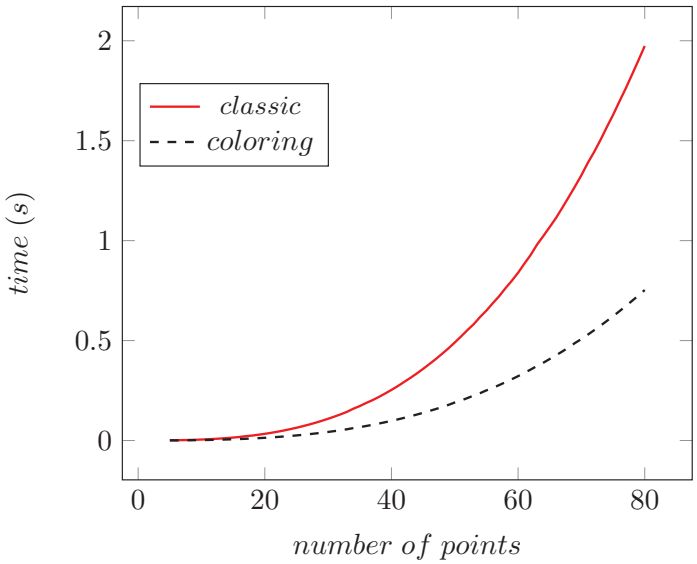


**FIGURE 4.2 –** *Temps de calcul (Matlab) de la matrice jacobienne en fonction du nombre de points $N$ pour l'approche classique (sans coloriage) et l'approche avec coloriage. Les temps sont des moyennes sur* 20 *exécutions.*

est donnée en partie droite de la figure 4.1. On s'attend à ce que les temps de calcul soient largement réduis par le coloriage. Ainsi, si l'on considère une implémentation Matlab de la fonction $F$ [1], on peut utiliser la librairie RRD (Rosin-Rammler Diagram) de BREZANI et ZELENAK [20] pour calculer les couleurs. Les temps de calcul de la matrice jacobienne avec et sans coloriage sont présentés en figure 4.2. Pour finir, remarquons que les évaluations de la fonction $F$ sont indépendantes. Cela signifie que ces évaluations peuvent être faites en parallèle : le calcul de la matrice jacobienne est codé comme un appel à la fonction $F$ à l'intérieur d'une boucle sur les couleurs. On peut donc facilement paralléliser le calcul de la matrice jacobienne en utilisant par exemple OpenMP. Les temps de calcul, pour une implémentation Fortran-OpenMP, sont donnés en figure 4.3. Il existe cependant des cas où ni la compression des lignes, ni celle des colonnes n'est efficace. Dans certains de ces cas, l'approche de COLEMAN et VERMA proposée dans [34] permet de contourner le problème en compressant simultanément les lignes et les colonnes. Il est également envisageable de donner la même couleur à des variables dépendantes et de retrouver les coefficients de la matrice jacobienne en faisant la résolution de systèmes linéaires [98].

Remarquons que la méthode du coloriage et l'approche *sparse forward* ne sont pas incompatibles, il est tout à fait possible d'utiliser une méthode de coloriage lorsque l'évaluation de la fonction $F$ a été optimisée en gardant les valeurs intermédiaires. Dans ce cas, on pourra trouver un équilibre entre l'effort mis sur l'optimisation de la fonction $F$ et celui mis sur le

[1] Version Fortran disponible à l'adresse `http://www.dm.uniba.it/~testset/problems/fekete.php`

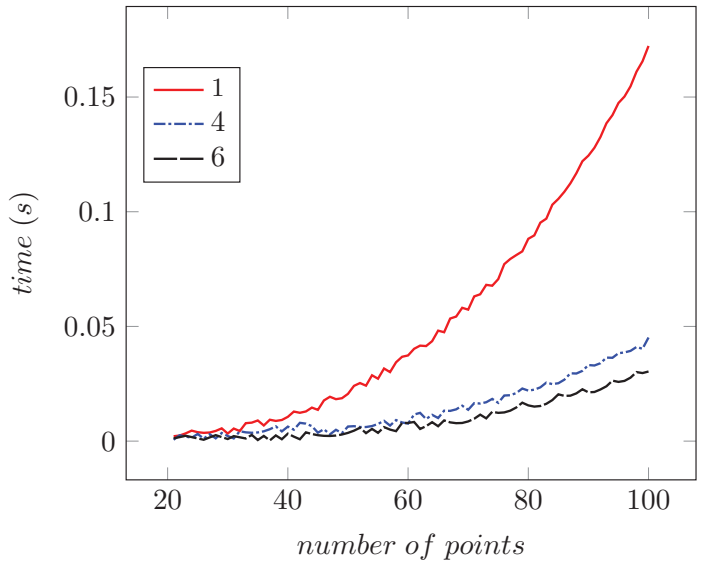


**FIGURE 4.3 –** *Comparaison du temps de calcul (s) d'une matrice jacobienne avec coloriage en fonction de $N$ pour 1, 4 et 6 threads. Implémentation Fortran 90/OpenMP.*

coloriage. Dans le cas où la fonction $F$ est pleinement optimisée, le coloriage ne permet pas de réduire le nombre de *flops*, mais sera quand même utile d'un point de vue informatique, en diminuant le coût dû à l'*overhead* lors de l'appel à la fonction $F$.

Chaque itération de Newton demande la résolution d'un problème linéaire de la forme $J(x)\delta x = -F(x)$. En réalité, ces méthodes de Krylov ne demandent pas de disposer de l'opérateur linéaire $J$, mais seulement d'être capable de calculer le produit matrice-vecteur $J(x)v$, $\forall v \in \mathbb{R}^n$. On parlera dans ce cas de méthode JFNK pour *Jacobian-free Newton-Krylov*. Nous rappelons ici le fonctionnement de cette méthode, plus de détails pourront être trouvés dans l'un des papiers originels [109] ou dans [82]. Ainsi, le produit matrice-vecteur peut être approché de la manière suivante :

$$J(x)v = \frac{F(x + \epsilon v) - F(x)}{\epsilon} \tag{4.14}$$

où $\epsilon \in \mathbb{R}$ est une petite perturbation choisie de manière judicieuse (voir par exemple [30, 109]). Notons que lorsque la matrice jacobienne est évaluée complètement, il est possible d'adapter au mieux la valeur de $\epsilon$ en fonction de la colonne de la matrice jacobienne considérée. Ce n'est plus cas avec la méthode JFNK. Afin d'assurer la convergence des méthodes de Krylov, il sera en général nécessaire de préconditionner (voir 2.1.4) le problème. Il ne sera donc plus possible d'utiliser une technique de préconditionnement qui nécessite de stocker l'opérateur $J$.

#### 4.1.2.2 Approximation de la matrice jacobienne dans la méthode de Broyden

On présente ici la méthode de Broyden (voir [78, chapitre 7]) pour résoudre le problème non linéaire $F(x) = 0$. Les itérations de Broyden peuvent s'écrire comme

$$x_{k+1} = x_k - J_k^{-1} F(x_k) \tag{4.15}$$

où $J_k$ est une approximation de la matrice jacobienne, construite par des mises à jour successives d'une matrice $J_0$ fournie par l'utilisateur. Nous utiliserons les notations $F_k = F(x_k)$, $\Delta F_k = F_{k+1} - F_k$ et $\Delta x_k = x_{k+1} - x_k$. Les méthodes de Broyden sont des méthodes de quasi-Newton où la mise à jour de la matrice jacobienne satisfait la condition de la sécante [40] :

$$J_{k+1} \Delta x_k = \Delta F_k. \tag{4.16}$$

Les itérations de quasi-Newton de l'équation (4.15) requièrent la résolution d'un système linéaire. Ceci peut être évité si l'on travaille directement sur l'approximation de l'inverse de la matrice jacobienne. Ainsi, démarrant d'une approximation $G_0$ de l'inverse de la matrice jacobienne, la mise à jour quasi-Newton de $G_k$ qui satisfait la condition de sécante :

$$\Delta x_k = G_{k+1} \Delta F_k \tag{4.17}$$

est donnée par :

$$G_{k+1} = G_k + (\Delta x_k - G_k \Delta F_k) \frac{v_k^T}{v^T \Delta F_k}. \tag{4.18}$$

Généralement, $v_k$ sera égal à $\Delta F_k$ ou à $G_k^T \Delta x_k$ :

- Si $v_k = G_k^T \Delta x_k$, alors $G_{k+1}$ minimise la norme de Frobenius de $\|G_{k+1}^{-1} - G_k^{-1}\|_F$.
- Si $v_k = \Delta F_k$, alors $G_{k+1}$ minimise $\|G_{k+1} - G_k\|_F$.

Dans les deux cas, la preuve est déduite directement de la preuve du théorème 4.1 décrite dans [39]. Il est montré dans [22, 42] que la convergence de l'algorithme 10 est localement superlinéaire.

Toutefois, même si $G_0$ est une matrice creuse, $G_k$ pour $k \geq 1$ ne l'est pas. Par conséquent, la matrice $G_k$ n'est jamais formée explicitement, on calcule seulement son application à un vecteur. Typiquement, on démarre avec $G_0$ donnée sous forme de la factorisation LU de la matrice jacobienne $J(x_0)$, et $G_k$ est sauvegardée comme $G_k = G_0 + U_k V_k^T$ où $U_k$ et $V_k$ sont des matrices $n \times k$ avec les $k$ colonnes correspondant aux $k$ mises à jour de rang 1. Donc l'application de $G_k$ à un vecteur $z$ demande une descente-remontée pour calculer $G_0 z$, et deux produits matrice-vecteur : $U_k(V_k^T z)$. Une autre possibilité est de mettre à jour directement la factorisation de la matrice jacobienne [41].

On peut ajouter qu'une méthode de *linesearch* peut également être utilisée ici : l'étape 4 de l'algorithme 10 est remplacée par $x_{k+1} = x_k + \lambda \Delta x$, où $\lambda$ minimise la norme du résidu $F_{k+1}$. Lorsque de nombreuses itérations sont requises pour converger à la tolérance désirée, le stockage des mises à jour peut devenir un problème. Des méthodes qui limitent l'augmentation de la mémoire ont été développées [99, 107, 125].

**Algorithme 10** Méthode de Broyden pour résoudre $F(x) = 0$

**Require:** $x_0$ and $G_0$
1: $F_0 \leftarrow F(x_0)$
2: $k \leftarrow 0$
3: **repeat**
4: $\quad \Delta x_k \leftarrow -G_k F_k$
5: $\quad x_{k+1} \leftarrow x_k + \Delta x$
6: $\quad F_{k+1} = F(x_{k+1})$
7: $\quad \Delta F_k \leftarrow F_{k+1} - F_k$
8: $\quad v^T \leftarrow \Delta x_k^T G_k$ or $v^T = \Delta F_k^T$
9: $\quad G_{k+1} \leftarrow G_k + (\Delta x_k - G_k \Delta F_k) \dfrac{v^T}{v^T \Delta F_k}$
10: $\quad k \leftarrow k + 1$
11: **until** convergence

### 4.1.3 Préconditionneur Schwarz Additif Restreint appliqué aux problèmes non linéaires

La résolution d'une EDO/EDA non linéaire par une méthode implicite requiert la résolution d'un problème non linéaire $F(x,t) = 0$ à chaque pas de temps. Lorsqu'on applique une méthode de Newton, on se retrouve à résoudre des systèmes linéaires de la forme :

$$J(x_k^l, t^l)\delta x_k^l = -F(x_k^l, t^l) \tag{4.19}$$

où l'indice $k$ est le numéro de l'itération de Newton et $l$ est le numéro du pas de temps. La matrice $J(x_k^l, t^l) \in \mathbb{R}^{n\times n}$ est la jacobienne de $F(\cdot, t^l)$ à la solution $x_k^l$. En pratique, le système linéaire donné en (4.19) est préconditionné pour assurer la convergence de la méthode de Krylov. Le système linéaire est donc réécrit sous la forme

$$(M_k^l)^{-1} J(x_k^l, t^l)\delta x = -(M_k^l)^{-1} F(x_k^l, t^l) \tag{4.20}$$

dans le cas d'un préconditionnement gauche, et

$$\left\{ \begin{array}{rcl} J(x_k^l, t^l)(M_k^l)^{-1} y & = & -F(x_k^l, t^l) \\ \delta x & = & (M_k^l)^{-1} y \end{array} \right. \tag{4.21}$$

dans le cas d'un préconditionnement droit. Dans les deux cas, la matrice de préconditionnement $M_k^l$ doit approcher $J(x_k^l, t^l)$ et son inverse doit être facilement calculable. Des préconditionneurs basés sur les méthodes de décomposition de domaine sont souvent utilisés car le calcul de leur produit matrice-vecteur nécessite uniquement la résolution de problèmes indépendants et locaux aux sous-domaines. La méthode qui combine les méthodes de Newton-Krylov avec un préconditionneur basé sur une méthode de Schwarz s'appelle Newton-Krylov-Schwarz [24] et est largement utilisée aujourd'hui pour tous types de problèmes.

Nous considérons ici uniquement le préconditionneur *Restricted Additive Schwarz* proposé dans [27] et donné en équation (2.22). Le préconditionnement sera ici appliqué au niveau du système linéaire. Il est cependant possible d'utiliser la méthode de Schwarz directement au niveau non linéaire en utilisant par exemple la méthode *additive Schwarz preconditioned inexact Newton* (ASPIN) proposée par CAI et KEYES dans [25]. Le préconditionnement non linéaire à gauche consiste à remplacer la résolution du problème $F(x) = 0$ par la résolution de $H(F(x)) = 0$, où $H$ est une fonction qui approche $F^{-1}$ dans un certain sens (par exemple $H(F(x)) \approx x$), et qui s'annule uniquement en $0$.

Afin de simplifier les notations et lorsque cela n'entraîne pas de confusion, on écrira $J$ au lieu de $J(x,t)$, et $F$ au lieu de $F(x,t)$. Le préconditionneur RAS du système linéaire $J(x)\Delta x = -F(x)$ s'écrit sous la forme :

$$M_{RAS}^{-1} = \sum_{i=0}^{N-1} \widetilde{R}_i^T \left(J_i(x)\right)^{-1} R_i \tag{4.22}$$

où $R_i$ est l'opérateur de restriction au $i$ème sous-domaine en incluant le recouvrement, et $\widetilde{R}_i$ est l'opérateur de restriction au $i$ème sous-domaine sans le recouvrement. La matrice $J_i(x) = R_i J(x) (R_i)^T$ est la sous-matrice de $J(x)$ correspondant au $i$ème sous-domaine avec le recouvrement (voir 2.2.2.2 p.42).

Une façon simple d'économiser du temps de calcul est donc de réutiliser le même préconditionneur pour quelques systèmes linéaires successifs. L'algorithme 11 met cette idée en pratique : le préconditionneur est *gelé* pour quelques itérations de Newton, voire pour quelques pas de temps. À partir de maintenant, on notera LRAS (*Lagged Restricted Additive Schwarz*) ce préconditionneur.

**Algorithme 11** Intégration en temps avec un préconditionneur LRAS

**Require:** an initial guess $x$, $k = 0$
**Require:** a partition of the unknowns into $N$ subsets
1: **for** each time step $l$ **do**
2:  **for** each Newton iteration $k$ **do**
3:  **if** the preconditioner is restarted **then**
4:  compute $M_{LRAS}^{-1} = \sum_{i=0}^{N-1} \widetilde{R}_i^T J_i^{-1} R_i$ (LU factorization of each $J_i$)
5:  **end if**
6:  Solve $J\delta x = -F$ preconditioned by $M_{LRAS}^{-1}$
7:  $x_{k+1}^l \leftarrow x_k^l + \delta x$
8:  **end for**
9: **end for**

Dans l'algorithme 11, la condition à laquelle le préconditionneur est recalculé n'est pas détaillée. En pratique, il est très difficile de choisir quand effectuer ce recalcul. Deux approches

heuristiques peuvent être utilisées :

- Le préconditionneur est recalculé lorsqu'un système linéaire a demandé plus de $K_{max}$ itérations de Krylov.
- Le préconditionneur est recalculé tous les $L$ systèmes linéaires.

La première approche semble plus flexible dans la mesure où elle permet d'économiser de nombreuses mises à jour inutiles. Cependant, le nombre d'itérations de Krylov peut varier fortement durant la simulation, et cette variation est principalement due à la raideur de la physique et non pas au retard pris par le préconditionneur. Dans l'idéal, il faudrait donc adapter $K_{max}$ durant la simulation. On peut par exemple déterminer les valeurs de $K_{max}$ a priori si le comportement numérique de l'intégrateur est déjà connu. On peut également utiliser des méthodes statistiques pour estimer la valeur de $K_{max}$ minimisant les temps de calcul.

## 4.2 Mise à jour de Broyden du préconditionneur RAS

Une revue des mises à jour de rang 1 de la jacobienne de type Broyden est donnée en section 4.1.2.2 p.86. Le but de cette section est d'étendre cette idée à la mise à jour des préconditionneurs basés sur méthodes de décomposition de domaine, tels que le préconditionneur RAS. On discute du besoin de redémarrage de l'algorithme, et on présente des résultats numériques sur le problème de la cavité entraînée.

### 4.2.1 Mise à jour de rang 1 du préconditionneur RAS

Nous proposons de mettre à jour le préconditionneur avec la technique de Broyden et en partant du préconditionneur RAS $G_0 = M_{RAS}^{-1}$. Le système linéaire préconditionné intervenant dans les itérations de Newton peut être écrit comme :

$$G_k J(x_k) \Delta x_k = -G_k F(x_k) \tag{4.23}$$

ou

$$J(x_k) G_k G_k^{-1} \Delta x_k = -F(x_k) \tag{4.24}$$

selon que le préconditionneur est appliqué à gauche ou à droite. La matrice $M_{RAS}^{-1}$ n'est pas calculée car la méthode de Krylov requiert uniquement le produit matrice-vecteur. Le produit de la matrice $M_{RAS}^{-1}$ et d'un vecteur quelconque est composé d'une résolution par sous-domaine et d'un échange des conditions aux limites artificielles. Dans la suite, on considérera uniquement le cas où les résolutions locales sont calculées à l'aide de la factorisation LU des opérateurs locaux $J^i$.

Les principales notations introduites en section 4.1.2.2 p.86 sont rappelées ici. En partant de l'approximation $G_0 = M_{RAS}^{-1}$ de l'inverse de la matrice jacobienne, la mise à jour quasi-Newton de $G_k$ qui satisfait la condition de sécante s'écrit :

$$G_{k+1} = G_k + \underbrace{\frac{(\Delta x_k - G_k \Delta F_k)}{v^T \Delta F_k}}_{u_k} v_k^T. \tag{4.25}$$

Il est commun de choisir $v_k = \Delta F_k$ pour minimiser $\|G_{k+1} - G_k\|_F$ ou $v_k = G_k^T \Delta x_k$ pour minimiser $\|G_{k+1}^{-1} - G_k^{-1}\|_F$.

L'algorithme 12 présente la mise à jour du préconditionneur dans le cas d'un intégrateur en temps, avec les critères de mise à jour de $G_k$ et de redémarrage qui seront définis plus tard.

**Algorithme 12** Intégrateur en temps avec mise à jour du préconditionneur

**Require:** a restart parameter $k_{max}$, an initial guess $x$, $k = 0$

**for** each time step **do**
  *// Newton iterations :*
  **repeat**
    **if** $k > k_{max}$ **then**
      $G_0 \leftarrow \sum_{i=0}^{N-1} (\widetilde{R}_i)^T (J_i(x))^{-1} R_i$ *// LU factorization of each* $J_i$
      $k \leftarrow 0$
    **end if**
    solve $J(x)\Delta x = -F(x)$ preconditioned by $G_k$
    $x \leftarrow x + \Delta x$
    compute $G_{k+1}$ from $G_k$
    $k \leftarrow k + 1$
  **until** convergence
**end for**

- Lorsque $v_k := \Delta F_k$, l'application du préconditionneur peut se réécrire comme :

$$G_{k+1}x = G_0 x + \sum_{i=0}^{k} u_i v_i^T x = G_0 x + [u_0 \cdots u_k] \, [v_0 \cdots v_k]^T \, x. \tag{4.26}$$

  Donc, le coût additionnel de l'application de $G_k$ comparé à $G_0$ correspond grossièrement à deux produits matrice-vecteur avec des matrices de taille $n \times k$. De plus, le calcul de $u_k$ demande une application de $G_k$. On peut aussi noter que les factorisations LU locales peuvent être réalisées de manière asynchrone, tout en continuant les itérations de Newton pendant la mise à jour du préconditionneur.

- Quand $v_k := G_k^T \Delta x_k$, le calcul explicite de $v_k$ doit être évité car il requiert l'application de la matrice $G_k^T$, ce qui demanderait *in fine* le calcul de $(M_{RAS}^{-1})^T$. Le produit matrice-vecteur $G_{k+1}x$ est donc généralement implémenté comme dans l'équation (4.27).

$$G_{k+1}x = \left( \prod_{i=k}^{0} (I - u_i \Delta x_i{}^T) \right) G_0 x \tag{4.27}$$

En utilisant une idée de MARTÍNEZ [94], il est prouvé dans [17], que si $G_0$ et $x_0$ sont d'assez bonnes premières conditions initiales, alors la norme $\|I - G_k J(x)\|$ peut être rendue aussi petite que voulu. Il devient donc pertinent d'utiliser la matrice $G_k$ comme préconditionneur de la matrice jacobienne. Ici le préconditionneur est aussi réutilisé d'un pas de temps sur l'autre. Intuitivement, comme la méthode de Newton converge plus rapidement que la méthode de Broyden, le préconditionneur perd progressivement de son efficacité et l'algorithme doit être redémarré, ce qui conduit à recalculer $G_0$ et écraser les vecteurs $v_k$ et $u_k$.

En termes de conditionnement, on ne s'attend pas à ce que le préconditionneur $G_k$ soit plus efficace que le préconditionneur $M_{RAS}^{-1}$ associé à la matrice jacobienne courante $J(x_k)$, mais son application à des vecteurs ne nécessite pas la factorisation des opérateurs locaux. Il est néanmoins possible de donner une borne inférieure du conditionnement du système linéaire après la mise à jour. Soient $\{\sigma_k\}$ et $\{\tau_k\}$ les valeurs singulières de $G_k J(x_{k+1})$ et $G_{k+1}J(x_{k+1}) = G_k J(x_{k+1}) + uw^T$ pour $w^T = v^T J(x_{k+1})$. Alors, la propriété d'entrelacement des valeurs singulières [70, théoreme 6.1] donne :

$$\begin{cases} \sigma_2 \leq \tau_1, \\ \sigma_{k+1} \leq \tau_k \leq \sigma_k, \quad 1 < k < n \\ 0 \leq \tau_n \leq \sigma_{n-1}, \end{cases} \tag{4.28}$$

donc :

$$\kappa_2(G_{k+1}J(x_{k+1})) = \frac{\tau_1}{\tau_n} \geq \frac{\sigma_2}{\sigma_{n-1}}. \tag{4.29}$$

Des résultats similaires peuvent être obtenus pour le préconditionnement droit des systèmes linéaires, étant donné que la mise à jour de rang 1 du préconditionneur conduit à une modification de rang 1 de l'opérateur préconditionné. Cette borne inférieure donne la limite de la procédure de mise à jour : elle ne sera pas efficace si le système préconditionné possède un ensemble important de très grandes ou de très petites valeurs singulières.

Nous pouvons tout de même illustrer l'effet de la mise à jour de Broyden sur un problème modèle. Soit $-\Delta_{FD2}$ l'opérateur discret du laplacien 1D avec des conditions aux limites homogènes, et soient $\{(U_i, \lambda_i)\}_{1 \leq i \leq n}$ les paires de valeurs et de vecteurs propres telles que $\lambda_i > \lambda_{i+1}$. On définit la fonction $F(v)$ en équation (4.30) qui s'annule pour $v = 0$ et sa matrice jacobienne $J(v)$ qui est définie positive. Les paires de valeurs et de vecteurs propres de

cette matrice sont $\{(\eta_i U_1 + U_i, \mu_i)\}_{1 \leq i \leq n}$. Le nombre de condition est donc : $\kappa_2(J(v)) = \frac{\mu_1}{\mu_n}$.

$$F(v) \stackrel{def}{=} \underbrace{(v,v)U_1U_1^T v}_{nonlinear} - \underbrace{\Delta_{FD2} v}_{linear} \tag{4.30}$$

$$J(v)h = 2(v,h)U_1U_1^T v + (v,v)U_1U_1^T h - \Delta_{FD2} h \tag{4.31}$$

$$\mu_1 = \Big(2\,(v,U_1)^2 + (v,v) + \lambda_1\Big), \mu_i = \lambda_i, 2 \leq i \leq n, \tag{4.32}$$

$$\eta_1 = 0, \eta_i = \frac{2(v,U_1)(v,U_i)}{\mu_i - \mu_1}, 2 \leq i \leq n. \tag{4.33}$$

Afin de simplifier les calculs, on prendra $X^0 = x_1^0 U_1$ et $G_0 = J(X^0)^{-1}$. Les itérations de Newton et de Broyden donneront donc le même $X^1 = \frac{2(x_1^0)^3}{\mu_1^0} U_1$. On écrit ensuite la matrice jacobienne $J(X^1)$ ainsi que son approximation $J^1$ en fonction de $J(X^0)$ :

$$\begin{aligned} J(X^1)h &= J(X^0)h + \Big[2(\delta X, h)(U_1, X^1) + 2(X^0, h)(\delta X, U_1) \\ &\quad + (2(X^0, \delta X) + (\delta X, \delta X))(h, U_1)\Big] U_1 \\ J^1 h &= J(X^0)h + F(X^1)\frac{(\delta X, h)}{(\delta X, \delta X)} U_1. \end{aligned}$$

Supposons que $X^0 = x_1^0 U_1$ alors on obtient :

$$\mu_1^0 = 3(x_1^0)^2 + \lambda_1,\ \delta X = -\frac{((x_1^0)^2 + \lambda_1) x_1^0}{\mu_1^0} U_1, \tag{4.34}$$

la valeur propre de $G_1 J(X^1)$ associée au vecteur $U_1$ est donnée par :

$$G_1 J(X^1) U_1 = \frac{{\lambda_1}^3 + 6\,(x_1^0)^2 {\lambda_1}^2 + 9\,\lambda_1 (x_1^0)^4 + 12\,(x_1^0)^6}{{\lambda_1}^3 + 7\,(x_1^0)^2 {\lambda_1}^2 + 17\,\lambda_1 (x_1^0)^4 + 19\,(x_1^0)^6} U_1$$

Ces résultats suggèrent que $G_1$ est un bon préconditionneur de $J(X_1)$, si $X^0$ est proche de la solution $X = 0$.

### 4.2.2 Mise en œuvre

On s'attend à ce que l'efficacité du préconditionneur non mis à jour décroisse d'une itération de Newton à l'autre. La mise à jour de type Broyden doit ralentir cette perte d'efficacité. Cependant, un redémarrage de l'algorithme est nécessaire comme la convergence de la méthode de Broyden est plus lente que la convergence de la méthode de Newton. Ce redémarrage (algorithme 12, étape 4) consiste à calculer un nouveau $G_0 = M_{RAS}^{-1}$ (et nécessite donc de nouvelles factorisations LU locales), correspondant à la matrice jacobienne courante.

Un inconvénient de la méthode proposée est l'accroissement du coût mémoire de deux vecteurs à chaque mise à jour. Quelques techniques peuvent être utilisées pour réduire ce

coût mémoire : la plus simple est de redémarrer le préconditionneur quand le nombre maximum de mises à jour est atteint. On peut aussi compresser les mises à jour en utilisant les décompositions en valeurs singulières tronquées de $[u_0 \cdots u_k][v_0 \cdots v_k]^T$ [125].

Le parallélisme des équations (4.26) et (4.27) peut être discuté :

- L'application du préconditionneur dans l'équation (4.26) à un vecteur $x$ met en jeu des communications globales comme les matrices $[u_0 \cdots u_k]$ et $[v_0 \cdots v_k]$ sont denses et distribuées entre les processeurs. Donc, suivant l'implémentation, l'équation (4.26) requiert des réductions globales additionnelles de $k$ valeurs, ou $k$ réductions dont les $k-1$ premières peuvent être recouvertes par des calculs.
- L'implémentation parallèle de l'équation (4.27) nécessite $k$ communications collectives séquentielles, et donc ne devrait pas être utilisée en pratique.

### 4.2.3 Résultats numériques sur le problème de la cavité entraînée

Les expériences numériques sont réalisées sur le problème de la cavité entraînée de dimension carré de longueur 1. Les méthodes de Newton-Krylov-Schwarz ont été appliquées très tôt à ce genre de problème de mécanique des fluides [80, 26].

La librairie PETSc [5] est utilisée pour l'implémentation. En particulier, on utilise l'implémentation de la formulation $(u, v, \omega, T)$ du problème disponible dans les exemples de la bibliothèque [33]. La méthode est alors implémentée au travers du contexte `SNES_SHELL` disponible dans PETSc. Le solveur linéaire utilisé dans ces expérimentations est BiCGStab [122] (voir l'algorithme 2 p.30) et les matrices jacobiennes sont calculées par la méthode des coloriages [65] (voir section 4.1.2.1 p.81).

$$\begin{cases} -\Delta(u) - \nabla_y(\omega) = 0 \\ -\Delta(v) + \nabla_x(\omega) = 0 \\ \dot{\omega} - \Delta(\omega) + \nabla.([u \times \omega, v \times \omega]) - \nabla_x(T) = 0 \\ \dot{T} - \Delta(T) + \nabla.([u \times T, v \times T]) = 0 \end{cases} \tag{4.35}$$

Ici $u$ et $v$ représentent les deux composantes du champ de vitesse, $\omega = -\nabla_y u + \nabla_x v$ est la vorticité et $T$ la température. La discrétisation spatiale est faite sur une grille régulière avec une molécule de calcul à 5 points et la discrétisation en temps est un schéma d'Euler implicite. La vitesse d'entraînement $u(x, 0)$ est constante et les autres conditions aux limites satisfont $u = v = 0$, $T = 0$ sur la paroi gauche , et $T = 1$ sur la paroi droite, $\partial T/\partial y = 0$ sur le dessus et sur le fond de la cavité.

La condition initiale est zéro partout sauf sur les parois, et la solution du pas de temps précédent est utilisée comme condition initiale pour le pas de temps en cours. Dans ces expériences, $G_0$ est le préconditionneur RAS associé à l'approximation de la jacobienne courante et le recouvrement d'un point. Dans les résultats qui suivent, les systèmes linéaires sont préconditionnés à droite. Ce choix est motivé par le fait que dans le préconditionnement gauche

le critère d'arrêt de la méthode de Krylov est basé sur la norme du résidu du système préconditionné. Donc pour comparer correctement l'effet de deux préconditionneurs différents, il faut que le critère d'arrêt soit basé sur la norme du problème non préconditionné $\|J\Delta x + F(x)\|_2$. Les itérations de Newton sont stoppées (c'est-à-dire que le pas de temps est accepté) quand la norme en valeur absolue du résidu est inférieure à $10^{-6}$.

**TABLE *4.1* –** *Nombre d'itérations de BiCGStab pour les préconditionneurs mis à jour et ceux gelés. La décomposition de la grille est régulière, avec le même nombre de sous-domaines dans chaque direction.* $1000$ *pas de temps de longueur* $10^{-3}$ *sont réalisés. L'algorithme est redémarré tous les* $40$ *pas de temps.*

| Proc. | Taille | Vitesse | Avec mise à jour | Sans mise à jour | Itér. économisées (%) |
|---|---|---|---|---|---|
| 8 | $128^2$ | 100 | 9009 | 9474 | 4.908 |
| 8 | $128^2$ | 300 | 16358 | 16748 | 2.329 |
| 16 | $128^2$ | 100 | 12724 | 13275 | 4.150 |
| 16 | $128^2$ | 300 | 17961 | 18345 | 2.093 |
| 16 | $256^2$ | 300 | 20011 | 20805 | 3.816 |
| 64 | $256^2$ | 300 | 28408 | 30114 | 5.665 |
| 64 | $256^2$ | 500 | 32599 | 32889 | 0.882 |

La table 4.1 compare le nombre d'itérations BiCGStab, dans le cas d'un préconditionnement droit, pour différentes tailles de grille et vitesses d'entraînement. Ces résultats montrent que la mise à jour de Broyden du préconditionneur peut réduire significativement le nombre d'itérations de Krylov. Pour une fréquence de redémarrage de $40$, le pourcentage d'itérations économisé décroît généralement avec l'accroissement de la vitesse d'entraînement. Ce résultat suggère qu'un algorithme de redémarrage plus approprié doit être conçu pour préserver l'efficacité de la mise à jour. Les résultats présentés ont été obtenus avec un pas de temps fixe. On s'attend à ce que l'efficacité de la mise à jour change si un algorithme de pas de temps adaptatif est utilisé, étant donné que le pas de temps est présent sur la diagonale de la matrice jacobienne.

## 4.3 Mise à jour partielle du préconditionneur RAS

### 4.3.1 Préconditionneur partiellement mis à jour

Dans l'algorithme 11, la mise à jour du préconditionneur est globale : toutes les factorisations LU sont recalculées simultanément. Nous pouvons étendre cette méthode au cas où seulement certaines parties du préconditionneur sont recalculées. En pratique, il peut y avoir des phénomènes non linéaires localisés dans certains sous-domaines. Dans ce cas, uniquement certains

blocs de la matrice jacobienne changent de manière significative d'une itération de Newton à l'autre. Par conséquent, la fréquence à laquelle la factorisation de la matrice jacobienne locale $J_i$ doit être effectuée peut varier d'un sous-domaine à l'autre. Ce nouveau préconditionneur est écrit en équation (4.36), où *AsRAS* signifie *Asynchronous Restricted Additive Schwarz*. Ce nom a été choisi car il est possible d'implémenter la mise à jour de manière asynchrone. Notons tout de même que dans le cas d'une méthode de Schwarz asynchrone [51, 114, 91], les communications entre les sous-domaines sont asynchrones. Ici les échanges entre les sous-domaines sont synchrones, mais c'est la mise à jour des problèmes locaux aux sous-domaines qui est asynchrone.

Le préconditionneur est maintenant composé de matrices jacobiennes locales évaluées à différentes itérations de Newton, voire à différents pas de temps. Il est utile de constater que si $t_i = t$ et $k_i = k$ pour $i = 0 \dots N-1$, alors $M^{-1}_{AsRAS} = M^{-1}_{LRAS}$.

$$M^{-1}_{AsRAS} = \sum_{i=0}^{N-1} \widetilde{R}_i^T J_i(x^{l_i}_{k_i}, t^{l_i})^{-1} R_i \tag{4.36}$$

Les méthodes de Schwarz asynchrones ont déjà été étudiées en tant que solveur pour des problèmes linéaires et non linéaires [3, 4, 51, 114]. Leur principe est d'éliminer, ou de relâcher la synchronisation présente à la fin de l'itération de Schwarz, qui consiste à faire l'échange des nouvelles conditions aux limites. Cette synchronisation implique qu'une itération de Schwarz dure aussi longtemps que la plus lente des résolutions locales. De manière à éliminer ces temps d'attente, cette synchronisation peut être supprimée, et ainsi chaque processeur passe à l'itération suivante, même si les conditions aux limites ne sont pas totalement mises à jour. L'inconvénient principal de ces solveurs asynchrones réside dans le fait qu'il est nécessaire de faire des hypothèses supplémentaires sur le problème et son découpage afin de garantir la convergence de la méthode. Ici, on peut directement appliquer le cadre théorique des méthodes de Newton-Krylov car la solution exacte du problème préconditionné est la même quelle que soit la matrice de préconditionnement. Ceci étant dit, les méthodes de Krylov approchent la solution à une certaine tolérance, et donc les décimales au-delà de cette tolérance peuvent changer lorsqu'on change de préconditionneur. On s'attend à ce que la mise à jour partielle permette d'économiser des itérations de Krylov durant la simulation, c'est ce que les résultats viendront confirmer. Cependant, on ne fera pas l'hypothèse que chaque mise à jour du préconditionneur améliore le conditionnement du système linéaire. Dans l'algorithme 13, $I$ est l'ensemble des indices des sous-domaines pour lesquels une factorisation LU sera calculée à la prochaine itération. Cet ensemble d'indices $I$ peut être défini a priori si le comportement non linéaire du problème est connu, ou après la résolution des systèmes linéaires en utilisant des critères numériques. L'implémentation séquentielle de l'algorithme 13 est triviale. Par contre, l'implémentation parallèle pose plus de difficultés car certains processeurs peuvent devoir attendre pendant que d'autres calculent une factorisation LU. Pour contourner ce problème, on propose de faire calculer ces factorisations LU par des processus additionnels.

**Algorithme 13** Mise à jour partielle du préconditionneur (AsRAS)

**Require:** une solution initiale $x$, $I = \{1, 2, \ldots, N\}$

**for** chaque pas de temps $l$ **do**
  $k \leftarrow 1$ // *itérations de Newton :*
  **repeat**
    **for** chaque $i \in I$ **do**
      $k_i \leftarrow k$, $l_i \leftarrow t$
      calculer la factorisation LU de $J_i(x_{k_i}^{l_i}, t^{l_i})$
      $M_{AsRAS}^{-1} = \sum_{i=0}^{N-1} \widetilde{R}_i^T J_i(x_{k_i}^{l_i}, t^{l_i})^{-1} R_i$
    **end for**
    résoudre $J\delta x = -F$ préconditionné par $M_{AsRAS}^{-1}$.
    $x_{k+1}^l \leftarrow x_k^l + \delta x$
    $k \leftarrow k + 1$
    définir l'ensemble des indices $I$ des sous-domaines à mettre à jour
  **until** convergence
**end for**

### 4.3.2 Implémentation parallèle du préconditionneur RAS asynchrone

La méthode présentée précédemment ne semble pas adaptée au calcul parallèle parce que la charge n'est pas équilibrée : certains processeurs vont devoir attendre que d'autres calculent une factorisation LU. Une illustration est donnée en figure 4.4 dans le cas de deux CPU et d'un sous-domaine par CPU. Seulement les parties les plus consommatrices en temps sont représentées dans cette figure : le calcul de la matrice jacobienne, les factorisations locales et la méthode de Krylov. Nous voyons qu'un temps d'attente apparaît lorsqu'un processeur calcule une factorisation LU et pas l'autre. Il y a cependant un cas où la mise à jour partielle du préconditionneur synchrone peut être efficace en parallèle : le cas où chaque processeur gère plusieurs sous-domaines.

L'idée principale de ce qui suit est que les factorisations LU peuvent être calculées par des processeurs additionnels, c'est-à-dire qui ne sont pas en charge de sous-domaines. Une illustration de cette méthode est donnée en figure 4.5 où les CPU 0 et 1 gèrent chacun un sous-domaine, tandis que le CPU 2 est dédié au calcul des factorisations LU. Les flèches en trait plein désignent l'envoi de matrices : les CPU 0 et 1 continuent leurs calculs jusqu'à la réception d'une factorisation. Lorsque la méthode de Krylov utilisée n'est pas une boîte noire, il est possible d'envoyer les matrices jacobiennes entre deux itérations de Krylov. Lorsque la méthode de Krylov autorise un préconditionneur variable, il est également possible de recevoir la factorisation LU entre deux itérations de Krylov. C'est ce qui est illustré en figure 4.6.

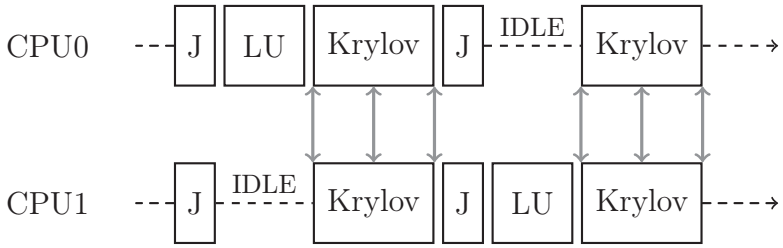


**FIGURE *4.4* –** *Enchaînement des étapes pour un préconditionneur mis à jour partiellement dans le cas de deux CPU.* J *correspond au calcul de la matrice jacobienne, et* Krylov *correspond à la résolution du système linéaire par une méthode de Krylov. Les communications de la méthode de Krylov impliquent un temps d'attente des CPU qui ne calculent pas la factorisation LU.*

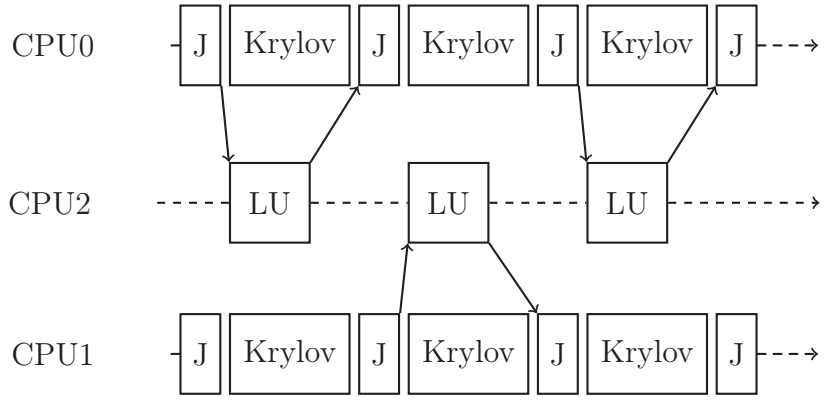


**FIGURE *4.5* –** *Enchaînement des étapes pour un préconditionneur mis à jour partiellement dans le cas de deux CPU associés à des sous-domaines, et un dédié aux factorisations LU.* J *correspond au calcul de la matrice jacobienne, et* Krylov *correspond à la résolution du système linéaire par une méthode de Krylov. Les flèches en trait plein représentent l'envoi d'une matrice jacobienne ou de sa factorisation LU. Cas où la méthode de Krylov utilisée est une boîte noire : impossible d'envoyer ou de recevoir des messages entre deux itérations de Krylov.*

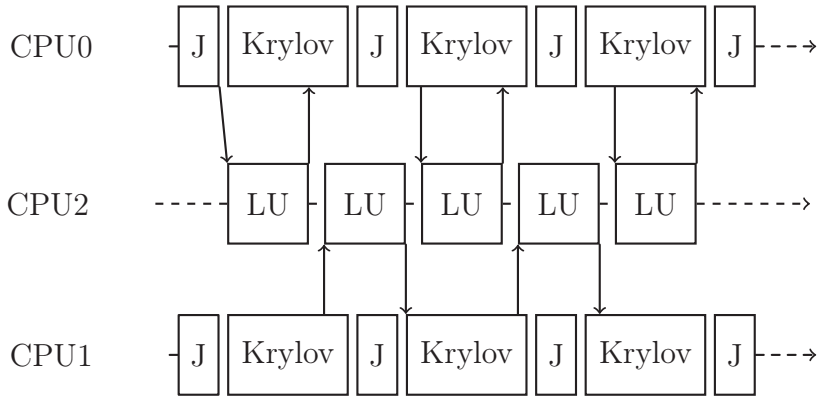


**FIGURE** ***4.6*** **–** *Enchaînement des étapes pour un préconditionneur mis à jour partiellement dans le cas de deux CPU associés à des sous-domaines, et un dédié aux factorisations LU.* J *correspond au calcul de la matrice jacobienne, et* Krylov *correspond à la résolution du système linéaire par une méthode de Krylov. Les flèches en trait plein représentent l'envoi d'une matrice jacobienne ou de sa factorisation LU. Cas où il est possible d'implémenter l'envoi et la réception de messages durant les itérations d'une méthode de Krylov qui autorise les préconditionneurs variables.*

#### 4.3.2.1 Algorithme client-serveur

Le point clé de l'implémentation est donc la définition de deux genres de tâches qui communiquent : les premières sont dédiées au calcul de la solution sur une partie du domaine, elles font donc les itérations classiques de Newton-Krylov. Les deuxièmes calculent uniquement les factorisations LU. De manière à résoudre le problème, on doit affecter la majorité des processeurs au premier genre de tâches, et les autres au second. L'algorithme 14 décrit comment implémenter cette méthode par une approche client-serveur. Les processus clients sont ceux qui sont chargés d'un sous-domaine, tandis que les serveurs calculent les factorisations LU. Les clients doivent être capables de continuer les itérations de Newton-Krylov entre l'envoi d'une matrice jacobienne au serveur, et le retour de la factorisation. La réception de la nouvelle matrice factorisée se fait dans le même espace mémoire que l'ancienne, c'est pourquoi la réception de la matrice est aussi la mise à jour partielle. La réception peut également être faite entre deux itérations de la méthode de Krylov, si celle-ci permet l'utilisation d'un préconditionneur variable [110, 112]. On s'attend à ce que le préconditionneur gelé LRAS perde petit à petit de son efficacité. Il est donc nécessaire de le recalculer de temps en temps sinon la méthode de Krylov risque de pas converger. Dans la mesure où l'implémentation d'AsRAS est asynchrone, il faut prévoir le cas où les mises à jour ne sont pas reçues. C'est pourquoi le recalcul global du préconditionneur par tous les processus clients est présent dans l'algorithme

d'AsRAS comme dans celui de LRAS. En pratique, il est possible que le nombre de processus clients soit trop petit pour éviter la perte d'efficacité du préconditionneur AsRAS. Dans ce cas, les mises à jour partielles permettent tout de même de ralentir cette perte d'efficacité.

Finalement, la combinaison des algorithmes client et serveur est donnée en algorithme 14 dans le cas d'une implémentation MPI. Dans cette implémentation, la réception de la nou-

**Algorithme 14** Implémentation client-serveur d'AsRAS

```
1:  // Client process                          1:  // Server process
2:  for each time step do
3:    // Newton iterations:
4:    repeat
5:      MPI_Test if LU factorization          2:  repeat
        is available                          3:    choose a client
6:      if LU factorization is available      4:    ask for a matrix J
        then                                  5:    receive the matrix
7:        partially update M^{-1}_{AsRAS}     6:    compute the LU factorization
8:      end if                                7:    warn that the factorization
9:      if global update then                       is ready
10:       M^{-1}_{AsRAS} = sum_{i=1}^{N} R̃_i^T J_i^{-1} R_i   8:    send the factorization
11:     end if                                9:  until the end of the integration
12:     Krylov method to solve
        Jδx = −F(x) preconditioned by
        M^{-1}_{AsRAS}
13:     x ← x + δx
14:     MPI_Test if server is ready
        to recieve the matrix J
15:     if server ready to recieve then
16:       send the matrix J
17:     end if
18:   until convergence
19: end for
```

velle factorisation LU se fait dans l'espace mémoire de l'ancienne. Il n'est donc pas possible de faire une réception non bloquante de cette factorisation. On contournera le problème de la façon suivante : le processus serveur prévient le processus client que la factorisation est prête en lui envoyant un message qui sera reçu de manière non bloquante. Une fois ce message reçu, le processus client fait la réception de la factorisation. Ainsi, dans l'algorithme 14, les flèches pleines représentent les communications synchrones, celles en pointillés représentent les communications asynchrones. L'implémentation est donc bien asynchrone dans la mesure

où les itérations de l'algorithme client continuent tant que la factorisation LU n'est pas effectuée. Cette méthode peut être vue comme une amélioration de la méthode de Newton-Krylov présentée en algorithme 11 car les deux méthodes sont équivalentes lorsque aucun processus n'est affecté à la mise à jour partielle du préconditionneur.

Comme cela a déjà été précisé , il y a en pratique plus de processus qui effectuent l'algorithme client que l'algorithme serveur. Par conséquent, seulement quelques mises à jour peuvent être calculées simultanément. Nous devons donc décider dans quel ordre les requêtes des sous-domaines devront être traitées. Il y a quelques critères numériques qui peuvent être utilisés pour décider quels sont les sous-domaines qui ont le plus besoin d'une mise à jour :

- On peut mettre à jour en priorité les sous-domaines dans lesquels la solution varie le plus rapidement en utilisant la norme $\|R_i x^l - R_i x^{l-1}\|$.
- Si le préconditionneur AsRAS préconditionne idéalement la partie de $J$ associée au sous-domaine $i$, alors $\widetilde{R}_i M_{AsRAS}^{-1} \widetilde{R}_i^T \widetilde{R}_i J \widetilde{R}_i^T \approx I$ et donc $\widetilde{R}_i M_{AsRAS}^{-1} \widetilde{R}_i^T \widetilde{R}_i J \widetilde{R}_i^T \widetilde{R}_i \delta x \approx \widetilde{R}_i \delta x$. On peut donc choisir le numéro du domaine $i$ à mettre à jour en fonction de la norme de $\| \widetilde{R}_i M_{AsRAS}^{-1} \widetilde{R}_i^T \widetilde{R}_i J \widetilde{R}_i^T \widetilde{R}_i \delta x - \widetilde{R}_i \delta x\|$ ou de l'angle entre les deux vecteurs $\widetilde{R}_i M_{AsRAS}^{-1} \widetilde{R}_i^T \widetilde{R}_i J \widetilde{R}_i^T \widetilde{R}_i \delta x$ et $\widetilde{R}_i \delta x$.

Dans la suite, on se limitera donc à un critère purement cyclique : on met d'abord à jour le premier sous-domaine, puis le second et ainsi de suite. Les raisons de ce choix sont les suivantes :

- En moyenne, aucun de ces critères ne s'est montré plus efficace que la mise à jour cyclique lors de nos expérimentations numériques.
- La méthode cyclique est générale, applicable à tous les problèmes.
- Toutes les parties du préconditionneur sont régulièrement mises à jour.
- Dans notre implémentation, un processus serveur n'aura que quelques clients avec qui il partage de la mémoire. L'ordre dans lequel sont effectuées les mises à jour est donc moins important que pour une implémentation où tous les processus serveurs peuvent servir n'importe quel processus client.

#### 4.3.2.2 Implémentation parallèle

On décrit maintenant une implémentation MPI possible de l'algorithme 14 dans le cas d'un cluster de calcul : plusieurs nœuds de calcul comptant chacun quelques cœurs qui partagent de la mémoire. Ces nœuds sont connectés entre eux par une connexion réseau. La principale difficulté pour arriver à une implémentation MPI efficace est le choix de la répartition des tâches sur les cœurs. Les points suivants doivent être pris en compte :

- Les cœurs qui calculent les factorisations LU échangent des messages de grande taille (matrices et factorisations LU) avec leurs clients.

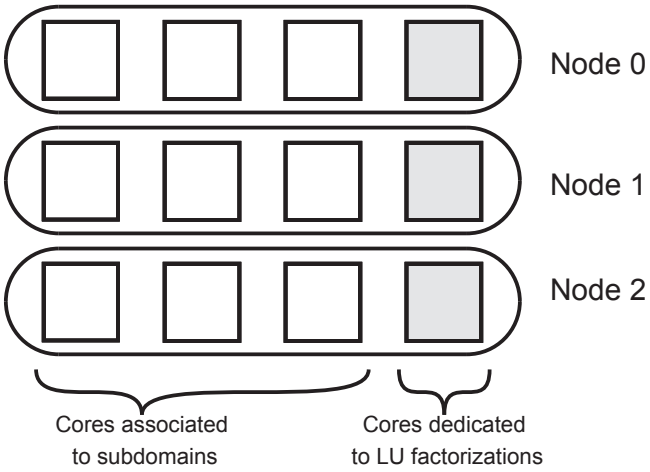


**FIGURE** *4.7 – Exemple de distribution des tâches sur les cœurs dans le cas de trois nœuds ayant chacun quatre cœurs.*

- Les processus associés à des sous-domaines voisins échangent régulièrement des messages de petites tailles, notamment lors des produits matrice-vecteur de la méthode de Krylov.

De manière générale, on cherchera à éviter l'envoi de matrices factorisées à travers le réseau. Il sera donc nécessaire d'utiliser une bibliothèque MPI qui effectue les communications de manière efficace entre les cœurs d'un même nœud. Ainsi, on pourra distribuer les processus de manière à ce que les processus serveurs et leurs clients partagent de la mémoire. Finalement, l'utilisateur devra définir le nombre de cœurs par nœud dédiés aux factorisations LU.

L'illustration donnée en figure 4.7 est un exemple de distribution : neuf cœurs sont associés à des sous-domaines physiques, et trois effectuent la mise à jour du préconditionneur. La distribution est telle que chaque nœud a trois cœurs en charge d'un sous-domaine, et un en charge des factorisations LU.

Comme dit précédemment, on se limite ici à une mise à jour cyclique : toutes les parties du préconditionneur sont mises à jour avec la même fréquence. Cette approche n'est pas optimale dans le sens où on aimerait mettre à jour de manière plus fréquente les sous-domaines où des phénomènes hautement non linéaires sont présents. Dans ce cas, la répartition des processus sur les processeurs est un point critique : cette distribution doit être faite de manière à équilibrer les calculs. Autrement dit, qu'il faudrait séparer les sous-domaines ayant besoin de mises à jour fréquentes. L'implémentation proposée n'est donc pas optimale mais est la plus générale.

#### 4.3.2.3 Variations possibles de la méthode

Nous suggérons ici trois variations possibles de la méthode : le cas de la parallélisation de factorisation LU, le cas des factorisations incomplètes, et le cas des méthodes *Jacobian-free* :

- On a considéré uniquement le cas où un processus client effectue la factorisation LU associée à un sous-domaine. La factorisation LU étant une étape couteuse, il peut être avantageux d'utiliser des solveurs parallèles [90, 1]. L'implémentation décrite dans la partie précédente considère que le calculateur est homogène. Dans le cas d'un calculateur ayant des coprocesseurs tels que les GPU, on peut penser à effectuer les factorisations LU sur ces derniers. Une accélération correcte des calculs peut être obtenue. Cependant, les coprocesseurs, et en particulier les GPU seront probablement sous-utilisés dans la mesure où les transferts de données représenteront une part importante du temps.
- Une extension naturelle de ce travail est de calculer des factorisations ILU (*Incomplete LU factorizations*). Dans ce cas, la résolution des problèmes locaux ne se fait plus par une factorisation LU, mais par une méthode de Krylov préconditionnée par un ILU. L'avantage de cette méthode est que le nombre d'itérations de Krylov locales devient un bon critère pour choisir quels sont les sous-domaines à mettre à jour.
- On peut également étendre cette méthode au cas d'un solveur du type *Jacobian-free Newton-Krylov* [82]. Dans ce cas, le processus client envoie la solution $x^l$ et le processus serveur calcule la matrice jacobienne et sa factorisation. Cette approche peut être utile dans le cas où la matrice jacobienne est coûteuse à calculer.

### 4.3.3 Tests numériques

On présente différents tests qui permettent de mettre en évidence le comportement de la méthode. Dans la première partie, on analyse le comportement de la méthode lorsque le paramètre de mise à jour globale $K_{max}$ varie. Ces expériences sont faites pour un problème de réaction-diffusion de petite taille. Dans la seconde partie, on considère le problème de la cavité entraînée 2D. Trois tests y sont proposés afin de faire varier le nombre de processus additionnels, la vitesse d'entraînement et la charge par processeur. Le calculateur utilisé pour ces expériences est un SGI Altix XE 1300 avec deux processeurs Intel Xeon 5650 par nœud. L'implémentation est faite en PETSc [5]. Le solveur non linéaire est une méthode de Newton avec *lineseacrh* où les matrices jacobiennes sont approchées en utilisant la méthode des différences finies avec coloriage.

#### 4.3.3.1 Problème de réaction-diffusion

On compare dans un premier temps le comportement de AsRAS à celui de LRAS pour le problème de réaction-diffusion donné en équation (4.37). Le domaine est le carré unité avec

des conditions aux limites périodiques.

$$\begin{cases} \dot{u} - \alpha_1 \Delta u = A + u^2 v - (B+1)u \\ \dot{v} - \alpha_2 \Delta v = Bu - u^2 v \end{cases} \tag{4.37}$$

On considère les paramètres suivants :

- Le domaine est discrétisé en $100 \times 100$ points, en utilisant un schéma à cinq points.
- Le maillage est décomposé en $16$ sous-domaines avec un recouvrement d'un point.
- Le problème est résolu pour $t \in [0, 10]$ avec un schéma d'Euler implicite avec un pas de temps adaptatif.
- Les coefficients de la réaction sont : $A = 3.5$, $B = 12$, $\alpha_1 = 1 \times h^2$ et $\alpha_2 = 2.6 \times h^2$, où $h$ est le pas d'espace.
- La matrice jacobienne est calculée par différences finies.
- La méthode de Krylov utilisée est BiCGStab [122].
- La mise à jour globale s'effectue lorsque le nombre d'itérations de Krylov atteint $K_{max}$.
- LRAS a utilisé 16 cœurs, et AsRAS 20 (16 pour les sous-domaines, et 4 pour les mises à jour partielles).

Les temps d'exécution étant relativement petits, on fait la moyenne du temps d'exécution et du nombre d'itérations de Krylov sur 5 exécutions. Les résultats sont donnés en figure 4.9 et 4.8. En effet, l'aspect asynchrone de AsRAS implique que le nombre d'itérations de Krylov, et donc les dernières décimales de la solution, peuvent varier d'une exécution à l'autre. Le nombre cumulé d'itérations de Krylov est donné en figure 4.9. Dans les deux cas, le nombre d'itérations de Krylov augmente lorsque $K_{max}$ augmente, c'est-à-dire lorsque le nombre de mises à jour globales diminue. On constate que la mise à jour partielle du préconditionneur permet de limiter cette augmentation.

Les temps d'exécution sont tracés en figure 4.9. Dans les deux cas, le plus petit temps est obtenu lorsque l'on trouve le meilleur équilibre entre le coût des itérations de Krylov, et le coût du préconditionneur. Le temps minimum est de $2.46$s pour LRAS et de $2.15$s pour AsRAS. Si l'on compare ces deux temps, le speedup est de $1.14$ ce qui est plus petit que le speedup superlinéaire de $1.25$. D'un autre coté, si l'on considère tous les tests, le speedup moyen est d'environ $1.5$ parce que AsRAS est moins sensible à la valeur de $K_{max}$ que LRAS.

#### 4.3.3.2 Problème de la cavité entraînée

Les résultats numériques de cette partie sont effectués sur le problème de la cavité entraînée dans le carré unité suivant :

$$\begin{cases} -\Delta(u) - \nabla_y(\omega) = 0 \\ -\Delta(v) + \nabla_x(\omega) = 0 \\ \dot{\omega} - \Delta(\omega) + \nabla.([u \times \omega, v \times \omega]) = 0. \end{cases} \tag{4.38}$$

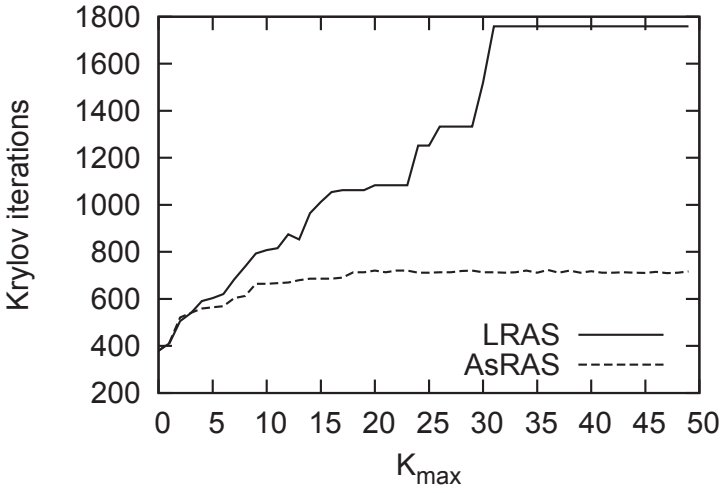


**FIGURE *4.8* –** *Nombre cumulé d'itérations de Krylov.*

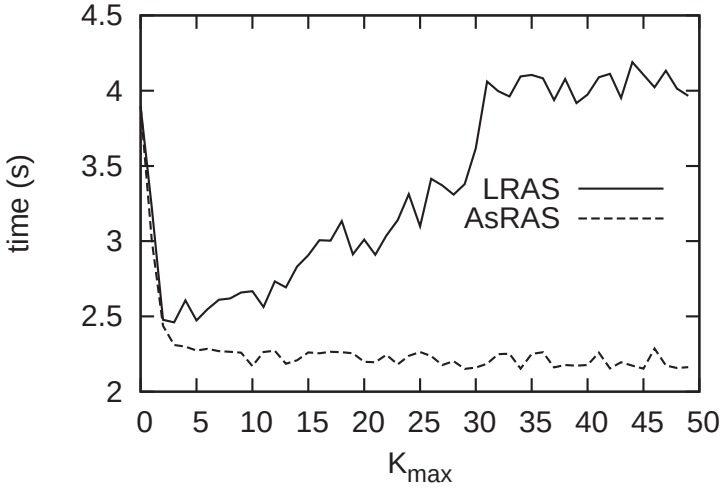


**FIGURE *4.9* –** *Temps de calcul en secondes.*

Dans l'équation (4.38), $u$ et $v$ sont les deux composantes du champ de vitesse, et $\omega = -\nabla_y u + \nabla_x v$ est la vorticité. La discrétisation en espace est effectuée sur une grille régulière avec un schéma à cinq points. L'intégration en temps se fait par une méthode d'Euler implicite. L'équation est traitée comme une équation différentielle ordinaire : les termes $\dot{u}$ et $\dot{v}$ sont ajoutés aux deux premières équations. Le pas de temps initial est de $10^{-4}$ et il est augmenté jusqu'à atteindre $100.0$.

**Accélération obtenue en externalisant la factorisation LU** Le premier test numérique vise à montrer l'effet de l'ajout de processus dédiés aux factorisations LU. Nous comparons donc les temps de calcul pour :

- Le problème de cavité entraînée discrétisé sur un maillage $900 \times 900$. Le domaine est décomposé en $100$ sous-domaines. Le problème est résolu par une méthode de Newton-Krylov-Schwarz pour $t \in [0, 8000]$.
- Le même problème, avec en plus $20$ processus additionnels dédiés aux factorisations LU. Sur chaque processeur (Intel Xeon 5650), un cœur sera dédié au serveur, et les $5$ autres seront dédiés à ses clients.

Nous comparons donc la résolution du problème en utilisant $100$ ou $100 + 20 = 120$ cœurs. Si l'on considère que la méthode de Newton-Krylov a une extensibilité linéaire pour de petites variations du nombre de processus, alors il devient intéressant d'affecter des processus additionnels aux factorisations LU si cela permet d'obtenir des accélérations superlinéaires. On pourra donc considérer que la méthode proposée est compétitive si l'ajout de $20$ processeurs permet de diviser le temps de calcul par un facteur d'au moins $1.2$. On peut également se focaliser sur les itérations de Krylov et définir l'accélération numérique comme le ratio des nombres d'itérations de Krylov. Le tableau 4.2 montre que :

- Plus souvent le préconditionneur complet est recalculé, moins les factorisations LU externalisées sont utiles. Par conséquent, les meilleurs speedups sont obtenus pour les plus grandes valeurs de $K_{max}$.
- Le préconditionneur *AsRAS* est plus robuste : les temps de calcul sont moins sensibles aux variations de $K_{max}$.
- Pour tous les tests présentés ici, l'accélération est au-delà de $1.2$, même pour $K_{max} = 100$ qui est le meilleur cas pour *LRAS*. Notons également que le coût des itérations de GMRES [110] n'est pas constant mais croît avec le nombre d'itérations.

**La proportion de processus additionnels** De manière à étudier le *strong scaling* de notre algorithme, on augmentera le nombre de processus additionnels pour une taille de problème fixée. Le tableau 4.3 présente les résultats pour une grille $512 \times 512$ décomposée en $16$ sous-domaines. La vitesse d'entraînement est de $10000$ et le problème est résolu pour $t \in [0, 32000]$.

**TABLE 4.2 –** *Temps d'exécution (s) et accélération pour une grille* $900 \times 900$.

| $K_{max}$ | 50 | 100 | 200 | 300 | 400 |
|---|---|---|---|---|---|
| *AsRAS* | 172.2 | 168.0 | 169.5 | 165.5 | 175.2 |
| *LRAS* | 221.1 | 215.3 | 221.9 | 227.4 | 298.9 |
| speedup | 1.284 | 1.281 | 1.309 | 1.374 | 1.706 |

La vitesse d'entraînement est de 100. *AsRAS* correspond à l'algorithme 14, *LRAS* correspond à l'algorithme 13. La méthode de Krylov est un GMRES préconditionné à gauche.

La méthode de Krylov est ici un BiCGStab préconditionné à droite. La valeur de $K_{max}$ est fixée à $50$. La première ligne du tableau 4.3 correspond donc au préconditionneur LRAS, puisque AsRAS et LRAS sont équivalents lorsque aucun processus n'est dédié aux factorisations LU. Pour être plus exhaustif, on peut également comparer LRAS et AsRAS pour un même nombre

**TABLE 4.3 –** *Temps d'exécution (s), nombre d'itérations de Krylov cumulées et accélération pour différents nombres de cœurs dédiés aux factorisations LU. Le nombre total de cœurs est* $16$ *plus le nombre de cœurs additionnels.*

| Cœurs additionnels | Temps (s) | Itérations de Krylov | Speedup (temps) | Speedup (iter.) | Speedup linéaire |
|---|---|---|---|---|---|
| 0 | 1397 | 19291 | 1.000 | 1.000 | 1.00 |
| 2 | 1118 | 14049 | 1.250 | 1.373 | 1.125 |
| 4 | 977 | 12549 | 1.430 | 1.537 | 1.25 |
| 8 | 934 | 11390 | 1.496 | 1.694 | 1.50 |

de cœurs. La dernière ligne du tableau 4.3 est donnée pour $24$ cœurs : $16$ sous-domaines et $8$ cœurs dédiés aux factorisations LU. Lorsqu'on résout le même problème avec $24$ sous-domaines et aucun processus additionnel (c'est-à-dire LRAS avec 24 cœurs), on obtient $19613$ itérations de Krylov, et $1341$ secondes. Lorsqu'on dispose de $24$ cœurs, il est donc préférable d'utiliser AsRAS plutôt que LRAS. Ceci s'explique par le comportement numérique du préconditionneur RAS : en général, le nombre total d'itérations de Krylov augmente lorsque le nombre de sous-domaines augmente pour une taille de problème fixe. D'un autre coté, le coût de préconditionneur devient moins important. Ici, lorsqu'on compare LRAS avec $16$ sous-domaines et LRAS avec $24$ sous-domaines, les deux effets se compensent à peu près : le temps de calcul passe de $1397$ à $1341$.

**Effet de la vitesse d'entraînement** Considérons maintenant une grille $500 \times 500$ décomposée en $50$ sous-domaines. Nous utilisons $10$ processus dédiés aux factorisations LU. La méthode choisie est un BiCGStab préconditionné à droite et le critère du recalcul global du préconditionneur est $K_{max} = 40$.

Les résultats présentés dans le tableau 4.4 montrent que le speedup varie, en restant

superlinéaire, lorsque la vitesse augmente. Ces résultats doivent être vus dans le contexte d'un $K_{max}$ fixe : pour LRAS, la valeur optimale de $K_{max}$ peut fortement changer lorsque la vitesse augmente.

**TABLE *4.4 –*** *Temps de calcul (s) pour LRAS et AsRAS sur le problème de la cavité entraînée*

| Vitesse d'entraînement | LRAS (s) | AsRAS (s) | Speedup |
|---|---|---|---|
| 100 | 88.32 | 63.26 | 1.40 |
| 200 | 106.0 | 65.40 | 1.62 |
| 300 | 104.7 | 68.95 | 1.52 |
| 400 | 111.6 | 69.95 | 1.60 |
| 500 | 105.1 | 74.39 | 1.41 |
| 600 | 111.9 | 76.52 | 1.46 |
| 1000 | 118.9 | 86.28 | 1.38 |

50 cœurs pour *LRAS* et 60 pour *AsRAS*, $t \in [0, 8000]$

Presque tous les systèmes linéaires requerront plus de $K_{max}$ itérations lorsqu'on augmente la vitesse d'entraînement en gardant la même valeur de $K_{max}$. Le préconditionneur sera donc globalement mis à jour quasi systématiquement. Dans ce cas, la mise à jour partielle ne sera plus utile.

**Augmentation de la charge par processus** On fait maintenant varier la taille de la grille pour un nombre de processeurs fixé. On peut s'attendre à une baisse de l'efficacité de la méthode proposée lorsque la taille par processeur augmente. Cela s'explique intuitivement par le fait que le temps de calcul des factorisations LU augmentera plus rapidement que le temps des itérations de Krylov.

Le tableau 4.5 présente les résultats pour le problème de la cavité entraînée pour différentes tailles de maillage, et différentes valeurs de $K_{max}$. Ce tableau donne le temps total d'exécution, le nombre total d'itérations de Krylov effectuées lors de la simulation, et les speedups. La plupart de ces speedups sont superlinéaires (c'est-à-dire plus grands que $1.2$ puisqu'on ajoute $20\%$ de processeurs) mais ils semblent décroître lorsque la charge par processeur augmente. Remarquons également que lorsque $K_{max} = 10$, un recalcul global du préconditionneur est effectué presque à chaque fois. Dans ce cas, il n'y a plus d'intérêt à utiliser AsRAS. Notre implémentation ne semble pas optimale car le nombre d'itérations de Krylov est augmenté par la mise à jour partielle lorsque $K_{max} = 10$ : une mise à jour asynchrone peut effacer une partie du préconditionneur qui est déjà à jour après un recalcul global. Pour être sûr d'éviter l'écrasement du préconditionneur déjà à jour, il faudrait utiliser une synchronisation globale, ce qui ralentirait l'exécution du code. Remarquons également que même si les itérations de Krylov représentent la partie la plus consommatrice en temps du code, le reste n'est pas négligeable. La recalcul global du préconditionneur et le calcul de la matrice jacobienne via la méthode des différences finies peuvent représenter une part

significative du temps d'exécution. Par exemple, le nombre d'itérations de Krylov est diminué d'un facteur $1.56$ pour la grille a $600 \times 600$ et pour $K_{max} = 50$, mais le temps d'exécution est seulement diminué d'un facteur $1.36$. Une limite de la méthode proposée est donc qu'elle diminue seulement le temps passé à calculer les itérations de Krylov, mais n'agit pas sur les autres parties du code.

**TABLE *4.5* –** *Speedups pour le problème de la cavité entraînée*

| Paramètres | | Temps (s) | | it. de Krylov | | Speedups | |
|---|---|---|---|---|---|---|---|
| Taille | $K_{max}$ | *LRAS* | *AsRAS* | *LRAS* | *AsRAS* | Temps (s) | it. de Krylov |
| $600 \times 600$ | 10 | 72.35 | 73.37 | 4076 | 4177 | 0.99 | 0.98 |
| $600 \times 600$ | 30 | 66.1 | 57.6 | 5390 | 4230 | 1.15 | 1.27 |
| $600 \times 600$ | 50 | 78.44 | 57.53 | 6580 | 4221 | 1.36 | 1.56 |
| $600 \times 600$ | 100 | 112.9 | 58.64 | 10366 | 4191 | 1.93 | 2.47 |
| $600 \times 600$ | 200 | 127.7 | 59.04 | 11420 | 4219 | 2.16 | 2.71 |
| $900 \times 900$ | 10 | 172.4 | 179.3 | 3698 | 3816 | 0.96 | 0.97 |
| $900 \times 900$ | 30 | 163.7 | 132.1 | 5473 | 3962 | 1.24 | 1.38 |
| $900 \times 900$ | 50 | 219 | 130.4 | 7824 | 3961 | 1.68 | 1.98 |
| $900 \times 900$ | 100 | 221.8 | 135.2 | 7869 | 4041 | 1.64 | 1.95 |
| $900 \times 900$ | 200 | 274.8 | 133.1 | 10008 | 3957 | 2.06 | 2.53 |
| $1200 \times 1200$ | 10 | 338.4 | 348.6 | 3576 | 3701 | 0.97 | 0.97 |
| $1200 \times 1200$ | 30 | 271.8 | 248 | 4815 | 3930 | 1.10 | 1.23 |
| $1200 \times 1200$ | 50 | 361.7 | 248.5 | 6730 | 3896 | 1.46 | 1.73 |
| $1200 \times 1200$ | 100 | 381.8 | 253 | 7183 | 3949 | 1.51 | 1.82 |
| $1200 \times 1200$ | 200 | 582.7 | 248.3 | 11642 | 3901 | 2.35 | 2.98 |
| $1500 \times 1500$ | 10 | 576.9 | 599.2 | 3428 | 3604 | 0.96 | 0.95 |
| $1500 \times 1500$ | 30 | 425.3 | 403 | 4515 | 3856 | 1.06 | 1.17 |
| $1500 \times 1500$ | 50 | 550.6 | 410.8 | 6232 | 3864 | 1.34 | 1.61 |
| $1500 \times 1500$ | 100 | 600.1 | 413.8 | 6903 | 3842 | 1.45 | 1.80 |
| $1500 \times 1500$ | 200 | 1115 | 417.1 | 13861 | 3859 | 2.67 | 3.59 |

Vitesse d'entraînement $u(0,t) = 400$ ; 100 cœurs pour *LRAS*, et 120 cœurs pour *AsRAS* ; BiCGStab préconditionné à droite ; $t \in [0, 8000]$

Ces résultats confirment que le préconditionneur AsRAS est beaucoup moins sensible au critère de recalcul du préconditionneur que LRAS.

## 4.4 Conclusions

La matrice de préconditionnement est souvent réutilisée pour des systèmes linéaires successifs afin d'économiser du temps de calcul. Il est également envisageable de mettre à jour le

préconditionneur plutôt que de le recalculer complètement. Une mise à jour du préconditionneur *Restricted Additive Schwarz* suivant la méthode de Broyden a été proposée. Le recalcul partiel du préconditionneur a ensuite été considéré.

La mise à jour de rang 1 de Broyden a donc été appliquée au préconditionneur RAS avec succès. Cette stratégie peut permettre d'accélérer les calculs. La réduction des temps de calcul a été significative, mais modeste, sur les tests proposés. Cette méthode s'avère néanmoins intéressante dans la mesure où son implémentation est simple.

Il a ensuite été proposé une méthode de préconditionnement, appelée AsRAS, où seulement une partie du préconditionneur est mise à jour. Cette méthode a été implémentée en ajoutant des processus dédiés à la mise à jour partielle et asynchrone du préconditionneur. Les résultats numériques ont montré qu'un speedup superlinéaire peut être obtenu par l'ajout de processus dédiés aux calculs des factorisations LU. De plus, comme le préconditionneur est continuellement mis à jour, les temps de calcul deviennent moins sensibles à la fréquence du recalcul global du préconditionneur. On s'est limité au cas où toutes les parties du préconditionneur sont recalculées avec la même fréquence, mais il aurait été pertinent de recalculer plus souvent les parties associées aux sous-domaines comportant de fortes non-linéarités. Le préconditionneur AsRAS pourrait donc être amélioré par l'utilisation d'un critère numérique permettant de choisir quelles parties du préconditionneur doivent être mises à jour. Une extension directe de ce travail serait de remplacer les factorisations LU par une méthode de Krylov préconditionné par une factorisation ILU. Dans ce cas, le nombre d'itérations de Krylov locales pourrait être le critère numérique permettant de choisir les parties du préconditionneur à mettre à jour.

# Chapitre 5

# Parallélisme à travers les pas de temps

**Sommaire**

L'idée de la parallélisation à travers le temps consiste à débuter la résolution du problème au temps $t_2$ alors que la convergence de la solution au temps $t_1 < t_2$ n'est pas achevée. Le principe de résoudre le problème à plusieurs pas de temps simultanément est à l'origine de plusieurs familles de méthodes :

- Les méthodes de tirs multiples décomposent le domaine temporel en sous-intervalles. Les résolutions sur chaque sous-intervalle peuvent être faites en parallèle. Il y aura ensuite une étape de correction du décalage entre la solution finale d'un intervalle et la condition initiale du pas de temps suivant. Ces méthodes sont généralement appliquées aux problèmes aux limites, même s'il est possible de les adapter aux problèmes à valeur initiale [81]. La méthode Parareal [87] peut être vue comme une méthode de tirs multiples (voir par exemple [54]).
- Il est également possible de résoudre simultanément le problème non linéaire de plusieurs pas de temps. C'est cette approche proposée par BELLEN dans [6] qui est considérée ici. Cette méthode sera appelée ici *pipelinage des pas de temps* : la première itération du deuxième pas de temps débute après la première itération du premier pas de temps.

Après avoir présenté la méthode du pipelinage des pas de temps, on utilisera la méthode de Broyden comme solveur à chaque pas de temps. Dans ce cas, nous proposons de propager les mises à jour de Broyden d'un pas de temps à l'autre [14]. Une méthode permettant de débuter les calculs sur tous les pas de temps simultanément sera également discutée. L'idée du pipelinage des pas de temps sera étendue à la recherche d'une solution initiale consistante d'une équation différentielle algébrique.

## 5.1 Pipelinage des pas de temps

Le parallélisme à travers les pas de temps peut être un moyen d'exploiter plus de processeurs efficacement lorsque la décomposition en espace n'est pas possible, ou qu'elle atteint ses limites en termes d'efficacité. On considère la résolution par une méthode d'Euler implicite d'un problème de la forme :

$$A\dot{x} = f(x,t) \tag{5.1}$$

avec une condition initiale et des conditions aux limites appropriées. À chaque pas de temps $i$, il faudra résoudre en $x^i$ le problème non linéaire

$$F\left(x^i, t^i, x_*^{i-1}\right) = M\left(x^i - x_*^{i-1}\right) - f\left(x^i, t^i\right) = 0 \tag{5.2}$$

où $x_*^{i-1}$ est la solution convergée du pas de temps précédent, et $M = A/\Delta t$. Ceci est généralement réalisé par une méthode de type Newton : la $k$ème itération de type Newton du $i$ème pas de temps peut s'écrire :

$$x_{k+1}^i = x_k^i - J\left(x_k^i, t^i\right)^{-1} F\left(x_k^i, t^i, x_*^{i-1}\right). \tag{5.3}$$

Dans la perspective du calcul parallèle, on souhaite démarrer le calcul de la solution au temps $t^i$ par un autre processus, même si la solution $x_k^{i-1}$ au temps $t^{i-1}$ n'a pas encore convergé vers $x_*^{i-1}$. L'idée sous-jacente au pipelinage en temps est de rassembler $N$ pas de temps dans un plus grand problème de la forme de l'équation (5.4), où $X_k = [x_k^1, \cdots, x_k^N]^T$.

$$\mathbb{F}(X_k) = \begin{bmatrix} F(x_k^1, t^1, x^0) \\ F(x_k^2, t^2, x_k^1) \\ \vdots \\ F(x_k^N, t^N, x_k^{N-1}) \end{bmatrix} = 0 \tag{5.4}$$

Alors l'itération de Newton sur les $N$ pas de temps simultanés peut s'écrire comme :

$$X_{k+1} = X_k - \begin{bmatrix} J\left(x_k^1, t^1\right) & & & \\ -M & J\left(x_k^2, t^2\right) & & \\ & \ddots & \ddots & \\ & & -M & J(x_k^N, t^N) \end{bmatrix}^{-1} \times \mathbb{F}(X_k)\,. \tag{5.5}$$

En notant $G_k^i = J(x_k^i)^{-1}$, on peut réécrire cette itération comme :

$$X_{k+1} = X_k - \begin{bmatrix} G_k^1 & 0 & \dots & 0 \\ G_k^2 M G_k^2 & G_k^2 & & \vdots \\ \vdots & \vdots & \ddots & 0 \\ G_k^N \prod_{i=1}^{N-1} (M G_k^i) & G_k^N \prod_{i=2}^{N-1} (M G_k^i) & \cdots & G_k^N \end{bmatrix} \times \mathbb{F}(X_k)\,. \tag{5.6}$$

Le calcul de $\Delta x_k^n$ dépend de $x_{k+1}^{n-1}$, ce qui implique un décalage du numéro de l'itération de Newton sur les différents pas de temps (voir la figure 5.1). Par conséquent, à un temps donné, la solution calculée est $\left(x_k^1, x_{k-1}^2, \cdots x_{k-N+1}^N\right)^T$. En d'autres termes, le calcul du $k$ème pas de temps démarre quand le premier pas de temps a déjà réalisé $k$ itérations de Newton. Le même décalage apparaît aussi en fin de calcul. Pour être efficace, il faut donc que le nombre de pas de temps rassemblés soit inférieur au nombre d'itérations de Newton nécessaires à la résolution.

Au niveau d'un pas de temps, l'incrément d'une itération de Newton est donc de la forme $\Delta x_k^n = -G_k^i F(x_k^i, t^i, x_{\mathbf{k+1}}^{i-1})$. Pour réduire les dépendances de données, on peut approximer $\Delta x_k^n$ par $\widehat{\Delta x_k^n} = -G_k^i F(x_k^i, t^i, x_{\mathbf{k}}^{i-1})$. Il s'ensuit que $\widehat{\Delta x_k^n} = -G_k^n M \Delta x_k^{n-1}$, et que l'itération de

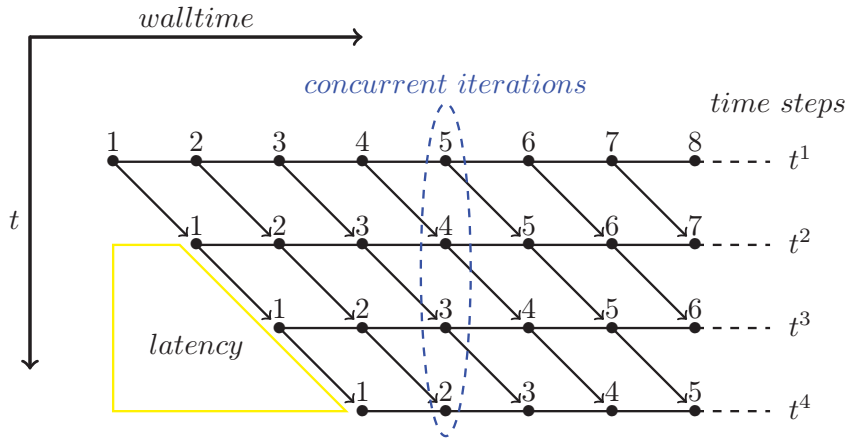


**FIGURE 5.1 –** *Pipelinage des pas de temps. Chaque • représente une itération du solveur non linéaire. Les flèches représentent l'envoi de la solution courante au pas de temps suivant.*

Newton est approchée par l'équation (5.7).

$$X_{k+1} = X_k - \begin{bmatrix} G_k^1 & & & \\ & G_k^2 & & \\ & & \ddots & \\ & & & G_k^N \end{bmatrix} \times \mathbb{F}\left(X_k\right) \tag{5.7}$$

En d'autres termes, la matrice jacobienne de $\mathbb{F}$ est approchée par une matrice bloc-diagonale, et donc des problèmes de convergence peuvent apparaître. Une méthode plus appropriée pour éviter le décalage entre les itérations est discutée dans ce qui suit.

Jusqu'à présent, on calculait $[x^1, \cdots, x^N]^T$ (sur $N$ processeurs), puis $[x^{N+1}, \cdots, x^{2N}]^T$ et ainsi de suite. L'inconvénient est que les $N-1$ premiers pas de temps par tranche sont surrésolus (voir la figure 5.2). En pratique, il y aura beaucoup plus de pas de temps à résoudre que de processeurs. Dans ce cas, si on note $s$ le nombre de processeurs, alors le processeur $r$ avec $0 \leq r < s$, débutera la résolution du pas de temps $t_0 + \Delta t(s \times (i) + r)$ dès lors qu'il aura terminé la résolution au temps $t_0 + \Delta t(s \times (i+1) + r)$. Autrement dit, les pas de temps seront distribués de manière cyclique sur les processeurs. Une implémentation en mémoire distribuée de cette méthode est proposée en algorithme 15. L'étape 13 de cet algorithme correspond à une communication collective : la variable `first` vaudra `true` uniquement sur le processeur en charge du plus petit temps $t^i$ en cours de traitement. De même, la variable `last` permet d'identifier le processus ayant la charge du dernier pas de temps de la tranche. Il faudra donc mettre à jour ces variables lorsqu'un problème non linéaire d'un pas de temps est résolu. Afin de simplifier les notations, on considérera dans la suite uniquement la réso-

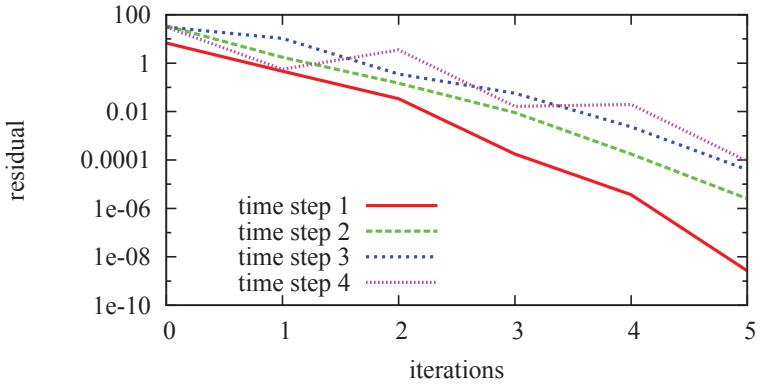


**FIGURE 5.2 –** *Résidus en échelle log10 pour quatre pas de temps simultanés sur le problème modèle de l'équation (5.10).*

lution d'un groupe de $N$ pas de temps sur $N$ processeurs. Les résultats numériques seront néanmoins donnés en considérant une répartition cyclique des pas de temps.

## 5.2 Pipelinage des itérations de quasi-Newton

Nous considérons à présent la mise à jour de Broyden de l'approximation de l'inverse de la matrice jacobienne $G_k$. Une première approche consiste à mettre à jour indépendamment $G_k^i$ pour chaque pas de temps comme cela est fait en équation (5.8), en partant de $G_0^i = J^i(x_0^i)^{-1}$.

$$G_{k+1}^i = G_k^i + \underbrace{(\Delta x_k^i - G_k^i \Delta F_k^i)}_{u_k^i} \underbrace{\frac{\Delta(F_k^i)^T}{\Delta(F_k^i)^T \Delta(F_k^i)}}_{(v_k^i)^T} \tag{5.8}$$

Afin de diminuer les coûts de calcul engendrés par l'évaluation de la matrice jacobienne, il est possible d'utiliser une même matrice jacobienne pour plusieurs pas de Newton ou pas de temps. Cette même idée s'applique à l'approximation de l'inverse de la matrice jacobienne. Quand les pas du schéma d'Euler sont calculés de manière séquentielle, on peut éviter le calcul de la factorisation LU de $G_0^i = J^i(x_0^i)^{-1}$ à chaque pas de temps en posant $G_0^i = G_*^{i-1}$ si cela permet la convergence de la méthode de Broyden. La même idée peut être appliquée dans le contexte parallèle : si l'on pose comme conditions initiales $G_0^1 = G_0^2 = \cdots = G_0^N$ alors on peut communiquer la mise à jour de Broyden d'un pas de temps sur l'autre.

Remarquons que si la fonction $f$ ne dépend pas explicitement de $t$, et s'il n'est pas possible de fournir une condition initiale différente pour chaque pas de temps ($X_0 = [x_0^1, \ldots, x_0^1]^T$), alors il s'ensuit naturellement que $G_0^1 = G_0^2 = \cdots = G_0^N$. Dans un tel cas, l'approximation de la matrice jacobienne inverse au pas de temps $i+1$ devrait aussi bénéficier de la mise à jour

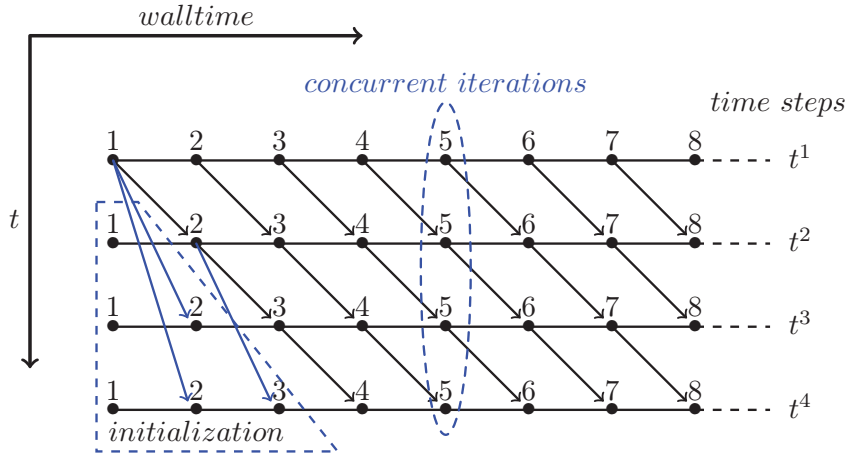


**FIGURE 5.3 –** *Pipelinage des pas de temps : chaque ligne représente un pas de temps. Le numéro de l'itération du solveur non linéaire local à chaque pas est donné. Chaque • représente une itération du solveur non linéaire. Les flèches représentent l'envoi de la solution courante au pas de temps suivant.*

de type Broyden du pas de temps $i$. La formule de mise à jour donnée par l'équation (5.9) généralise cette idée.

$$\begin{aligned} G_k^{i+1} &= G_k^i + u_k^i \left(v_k^i\right)^T \\ &= G_0^1 + \sum_{j=1}^{i} u_k^j \left(v_k^j\right)^T \end{aligned} \tag{5.9}$$

Toutefois, ces mises à jour ne peuvent pas être propagées lorsqu'on utilise différents $G_0^i$ pour chaque pas de temps. Par exemple, c'est le cas lorsque le même $G_0$ pour différents pas de temps ne garantit pas la convergence de la méthode.

### 5.2.1 Initialisation

Un des principaux inconvénients de cet algorithme est la phase d'initialisation : le $i$ème processeur associé au temps $t^i$ doit attendre que l'itération $i$ du Newton démarre. Quelques légères modifications des équations non linéaires peuvent réduire ce temps d'attente : si on estime $\dot{x}(t^i) \approx (x_1^i - x^0)/(i\Delta t)$, alors le $i$ème processeur peut démarrer sa première itération de Newton sans latence. Pour la deuxième itération, l'approximation sera $\dot{x}(t^i) \approx (x_2^i - x_1^1)/((i-1)\Delta t)$ et ainsi de suite. Le calcul de tous les pas de temps peut donc démarrer sans latence. Notons que l'on peut décider *a posteriori* de démarrer le calcul avec la condition

**Algorithme 15** Pipelinage en temps avec $s$ processeurs

**Require:** $r \leftarrow$ rank of this processor, $s \leftarrow$ number of processors
**Require:** $x_0^i$ for $i = 1...r$
**Require:** initial time $t_0$ and a steplength $\Delta t$
1: `last` $\leftarrow$ `false`, `first` $\leftarrow$ `false`
2: **if** $r = 0$ **then** `first` $\leftarrow$ `true` **end if**
3: **if** $r = s - 1$ **then** `last` $\leftarrow$ `true` **end if**
4: $i \leftarrow 0$
5: **if** not `last` **then** send $x_0^i$ to the $(r+1)$th processor **end if**
6: **repeat**
7: $t = t_0 + \Delta t(s \times i + r)$
8: **repeat**
9: **if** not `first` **then** recieve $x(t - \Delta t)$ from the processor number $r - 1 \pmod s$ **end if**
10: $G \leftarrow$ approximation of $(\partial F/\partial x(t))^{-1}$
11: $x(t) \leftarrow x(t) - GF(x(t), t, x(t - \Delta t))$
12: **if** not `last` **then** send $x(t)$ to the processor number $r + 1 \pmod s$ **end if**
13: collective update of `first` and `last`
14: **until** convergence
15: $i \leftarrow i + 1$
16: **until** final time

initiale $x_i^i$ ou avec la première condition initiale $x_0^i$. Par exemple, le calcul peut être démarré avec $x_i^i$ si $\|M(x_i^i - x_i^{i-1}) - f(x_i^i)\|_2 \leq \|M(x_0^i - x_i^{i-1}) - f(x_0^i)\|_2$.

Une illustration de cette initialisation est donnée en figure 5.3 : on peut voir que le calcul est démarré pour tous les pas de temps sans décalage mais l'initialisation nécessite un schéma de communication spécifique.

Une version simplifiée de la méthode incluant la phase d'initialisation est présentée dans l'algorithme 16 : tous les pas de temps (de 2 à $N$) sont initialisés et il n'y a pas de vérification de la solution après l'initialisation.

Finalement, le choix du paradigme de programmation parallèle va dépendre du problème. Si la méthode converge pour quelques pas de temps simultanés en utilisant la même matrice $G_0$ pour chaque pas de temps, alors une parallélisation en mémoire partagée semble être le bon choix car $G_0$ peut être partagée par les processus légers, et la mise à jour peut être propagée à tous. D'un autre côté, une implémentation en mémoire distribuée peut être meilleure si différentes matrices $G_0^i$ sont utilisées à chaque pas de temps. En fin de compte, les performances de la mise en œuvre dépendront du problème et des propriétés des calculateurs telles que la taille de la mémoire cache. Dans tous les cas, l'extensibilité (lorsque de plus en plus de pas de temps sont rassemblés) est limitée du fait de la nature séquentielle des EDO/EDA. Néanmoins, cette simple méthode peut être combinée avec la parallélisation en espace. Pour

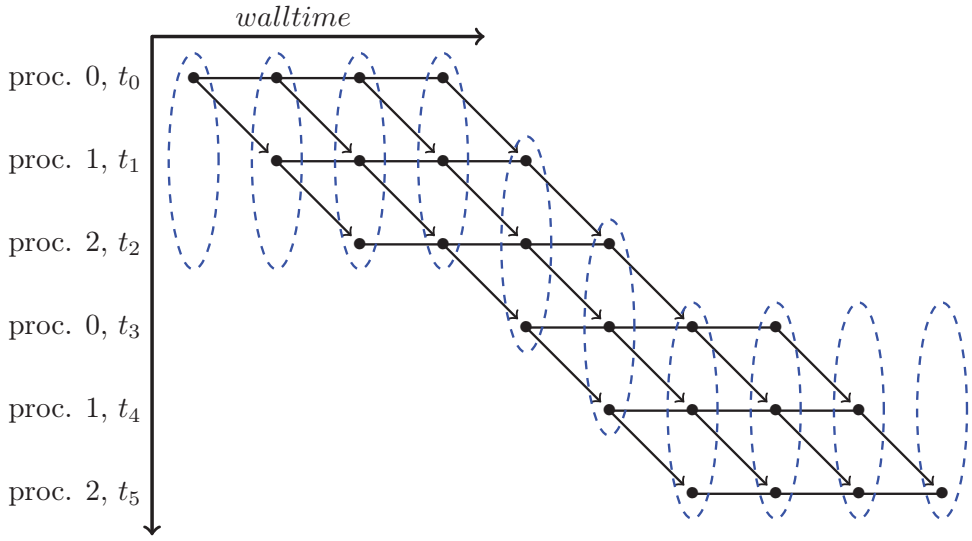


**FIGURE 5.4 –** *Pipelinage de* $6$ *pas de temps avec* $3$ *processus. Chaque* • *représente une itération du solveur non linéaire. Les itérations entourées sont faites en parallèle. Les flèches représentent l'envoi de la solution courante au pas de temps suivant.*

les problèmes plus complexes, une procédure de redémarrage doit être disponible pour calculer un nouveau $G_0 = J(x)^{-1}$ lorsque cela est nécessaire.

### 5.2.2 Résultats numériques

On présente les résultats numériques de la méthode de pipelinage en temps appliquée à une version dépendante du temps du problème de Bratu donné par l'équation (5.10). Le schéma de discrétisation en temps est la méthode d'Euler implicite et des différences finies sont utilisées pour la discrétisation en espace.

$$\begin{cases} \dfrac{\partial u(x,t)}{\partial t} = \Delta u(x,t) + \lambda(t) e^{u(x,t)},\ x \in ]0,1[,\ t \in ]0,T] \\ u(0,t) = c_1(t),\ u(1,t) = c_2(t) \\ u(x,0) = g(x),\ x \in [0,1] \end{cases} \tag{5.10}$$

On considère une implémentation Matlab de la méthode. Les paramètres sont $\lambda(t) = 2.4 \times cos(t)$, $c_1(t) = sin(t)\lambda(t)$, $c_2(t) = -sin(t)\lambda(t)$. Le problème est discrétisé avec $160$ pas de temps (avec $\Delta t = 4 \times 10^{-2}$), et $200$ inconnues ($\Delta x = 1/201$) en espace.

La table 5.1 donne le nombre total d'itérations de quasi-Newton pour exécuter $160$ pas en temps avec Euler implicite et pour différentes valeurs du nombre de pas de temps simultanés. Pas exemple, $N = 2$ signifie que nous résolvons le problème non linéaire sur deux pas de

**Algorithme 16** Pipeline en temps avec initialisation

**Require:** $x^0$, the solution at the previous time step
**Require:** $x_k^0 = x^0 \; \forall k \in \mathbb{N}$
**Require:** $x_0^i$ and $G_0^i$ for $i = 1...N$
**Require:** $\epsilon$ a tolerance
1: $k = 0$
2: **while** $\max_{i=1...N} \|F(x_k^i)\| > \epsilon$ **do**
3:  **for** $i = 1 \dots min(N, k)$ **do**
4:  $\Delta x_k^i = -G_k^i(M(x_k^i - x_k^{i-1}) - f(x_k^i, t^i))$
5:  $x_{k+1}^i = x_k^i + \Delta x_k^i$
6:  compute $G_{k+1}^i$ from $G_k^i$
7:  **end for**
8:  *//initialization :*
9:  **if** $k < N$ **then**
10:  **for** $i = k + 1...N$ **do**
11:  $\Delta x_k^i = \frac{-1}{(i-k)\Delta t} G_k^i \left( A \left( x_k^i - x_{k+1}^{k-1} \right) - f(x_k^i, t^i) \right)$
12:  $x_{k+1}^i = x_k^i + \Delta x_k^i$
13:  compute $G_{k+1}^i$ from $G_k^i$
14:  **end for**
15:  **end if**
16: **end while**

temps simultanément. Le problème a été initialisé en utilisant $G_0^i = J(x^0)^{-1}$ stocké sous la forme d'une factorisation LU. Un redémarrage de $r$ signifie que la matrice jacobienne exacte est calculée et factorisée tous les $r$ pas de temps. Par conséquent, pour $r = 10$ l'approximation de l'inverse de la matrice jacobienne est meilleure, mais $16$ factorisations LU ont été calculées. D'un autre côté, seule la factorisation LU initiale a été calculée pour $r = \infty$. Les deux symboles (I) et (II) font référence aux deux stratégies : (I), les mises à jour ne sont pas propagées d'un pas de temps sur l'autre ; (II) les mises à jour sont propagées d'un pas de temps à l'autre. Les résultats sont aussi donnés avec et sans la stratégie de *linesearch*.

Ces résultats montrent que :

- Le pipelinage réduit le nombre d'itérations séquentielles. Il est à noter que le coût d'une itération est constant pour tous les pas de temps. L'accélération numérique (de $1.8$ pour $N = 2$ et $r = \infty$ avec la méthode *linesearch*) doit être proche de l'accélération informatique avec MPI car la méthode ne demande qu'une communication processeur à processeur par itération. Seule la taille de ces communications est différente : de taille $n$ ($x$) pour (I) et de taille $3n$ ($x$, $u$ et $v$) pour (II).
- La propagation de la mise à jour réduit le nombre d'itérations pour $r = \infty$, c'est-à-dire

**TABLE *5.1*** – *Nombre total d'itérations pour les différentes stratégies de mise à jour : $N$ est le nombre de pas de temps simultanés, $r$ est le nombre de pas de temps entre chaque redémarrage (factorisation LU de $J$). (I) : La mise à jour n'est pas communiquée. (II) : La mise à jour est propagée d'un pas de temps à l'autre.*

| | sans linesearch | | | | avec linesearch | | | |
|---|---|---|---|---|---|---|---|---|
| | $r = \infty$ | | $r = 10$ | | $r = \infty$ | | $r = 10$ | |
| N | (I) | (II) | (I) | (II) | (I) | (II) | (I) | (II) |
| 1 | 512 | 512 | 453 | 453 | 430 | 430 | 409 | 409 |
| 2 | 285 | 275 | 239 | 244 | 249 | 233 | 214 | 212 |
| 4 | 194 | 178 | 175 | 175 | 178 | 175 | 172 | 171 |
| 5 | 186 | 174 | 173 | 173 | 176 | 173 | 173 | 173 |

quand $G$ n'est pas mis à jour. Dans ce cas, le nombre d'itérations de la stratégie (II) varie peu entre $r = 10$ et $r = \infty$. En d'autres termes, la propagation de la mise à jour permet d'éviter des factorisations LU.

- Le nombre moyen d'itérations de quasi-Newton est $430/160 \approx 2.68$ pour $N = 1$ et $r = \infty$. Donc, l'accélération maximale théorique est de $2.68$. Pour $N = 4$ et $r = \infty$, l'accélération numérique est de $2.42$, qui est proche de l'accélération maximale théorique. On peut donc s'attendre à une accélération satisfaisante pour un petit nombre de pas de temps simultanés ($2$ ou $4$ dans nos expériences).

Finalement, l'accélération par le pipelinage des pas de temps peut être meilleure que celle obtenue en espace notamment lorsque les sous-domaines deviennent petits.

## 5.3 Application à l'initialisation des EDA

La recherche d'une solution initiale consistante des équations différentielles algébriques est un problème complexe. Le calcul d'une solution initiale consistante peut parfois être fait de manière analytique, en calculant une solution satisfaisant à la fois les contraintes algébriques et les conditions initiales souhaitées. Il sera cependant nécessaire d'utiliser une méthode numérique dans le cas d'un logiciel de simulation. Cette phase d'initialisation peut représenter un coût de calcul important.

Le problème de l'initialisation des EDA sera détaillé en introduction en reprenant les notations de [84] et la méthode *Constrained Runs* sera présentée. Comme cette méthode consiste à effectuer plusieurs pas de temps, il est possible d'y appliquer le pipelinage des pas de temps. Des résultats numériques sur le problème de Fekete seront présentés dans la dernière partie.

### 5.3.1 Introduction

La forme la plus générale d'équation différentielle algébrique est donnée par l'équation (5.11) où $F : \Omega = \mathbb{I} \times \mathbb{D}_x \times \mathbb{D}_{\dot{x}} \to \mathbb{C}^m$, avec $\mathbb{I}$ un intervalle de $\mathbb{R}$, et $\mathbb{D}_x,\ \mathbb{D}_{\dot{x}}$ des ouverts de $\mathbb{C}^n$.

$$F(t, x, \dot{x}) = 0 \tag{5.11}$$

Les équations différentielles algébriques (EDA) sont une généralisation des équations différentielles ordinaires (EDO). Ainsi, lorsque l'équation $F(t, x, \dot{x}) = 0$ peut être résolue pour $\dot{x}$ de manière unique, l'EDA sera équivalente à une EDO. On peut également se ramener à une EDO de manière locale, en utilisant le théorème de la fonction implicite. Celui-ci dit que si $F$ est $C^1$ et si $F_{\dot{x}}$ est inversible au voisinage de $(t^*, \dot{x}^*, x^*)$ satisfaisant $F(t, x^*, \dot{x}^*) = 0$, alors localement on peut écrire $\dot{x} = f(x, t)$ pour $f$ une fonction $C^1$.

La notion de contrainte algébrique sera introduite sur les EDA linéaires, on s'intéressera en particulier à l'émergence de la notion d'indice des EDA.

#### 5.3.1.1 EDA linéaires à coefficients constants

Une équation différentielle sera appelée EDA lorsque la matrice jacobienne $F_{\dot{x}}$ est toujours singulière. C'est le cas lorsque des contraintes purement algébriques, au sens où elles ne font intervenir aucune composante de $\dot{x}$, sont incluses dans la fonction $F$. Les équations différentielles algébriques linéaires à coefficients constants sont la forme la plus simple d'EDA, elles s'écrivent :

$$E\dot{y} = Ay + f(t) \tag{5.12}$$

avec $E, A \in \mathbb{C}^{m,n}$ et $f \in C(\mathbb{I}, \mathbb{C}^m)$. Parfois ce système est associé à une condition initiale du type

$$x(t_0) = x_0. \tag{5.13}$$

Ces équations peuvent être traitées d'un point de vue purement algébrique, notamment à l'aide de la forme canonique de Kronecker. Dans la suite, on développe rapidement les principaux aspects de cette forme canonique pour pouvoir donner une définition de l'indice d'une EDA. L'idée principale est de réécrire l'équation (5.12) sous la forme

$$PEQ\dot{x} = PAQx + Pf, \tag{5.14}$$

avec $P \in \mathbb{C}^{m,m}$, $Q \in \mathbb{C}^{n,n}$ non singulières. On dira alors que les paires $(E, A)$ et $(PEQ, PAQ)$ sont fortement équivalentes, et on notera $(E, A) \backsim (PEQ, PAQ)$. On aura $(E_1, A_1) \backsim (E_2, A_2)$ s'il existe des matrices non singulières $P$ et $Q$ telles que $(E_2, A_2) = (PE_1Q, PA_1Q)$.

L'équation différentielle algébrique de la forme (5.12), avec $m = n$, sera résoluble pour une fonction $f$ suffisamment régulière si le polynôme $p$ défini par $p(\lambda) = det(\lambda E - A)$ n'est pas le polynôme nul. Dans ce cas, la paire $(E, A)$ sera dite régulière et la solution de l'EDA sera unique pour chaque condition initiale consistante.

**Lemme 1.** *Une paire de matrices fortement équivalente à une paire de matrices régulière est régulière.*

**Théorème 7.** *[84, p16] Si la paire $(E, A)$ est régulière alors il existe des matrices $J$ et $N$, avec $J$ sous la forme canonique de Jordan et $N$ nilpotente telles que :*

$$(E, A) \backsim \left( \begin{bmatrix} I & 0 \\ 0 & N \end{bmatrix}, \begin{bmatrix} J & 0 \\ 0 & I \end{bmatrix} \right) \tag{5.15}$$

Par conséquent, lorsque l'EDA est résoluble, il existe des matrices $P$ et $Q$ telles que :

$$PEQ = \begin{bmatrix} I & 0 \\ 0 & N \end{bmatrix}, PAQ = \begin{bmatrix} J & 0 \\ 0 & I \end{bmatrix} \tag{5.16}$$

L'EDA peut donc être décomposée en un système de deux équations :

$$\begin{cases} \dot{x_1} = Jx_1 + f_1 \\ N\dot{x_2} = x_2 + f_2 \end{cases} \tag{5.17}$$

$$Q^{-1}y = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}, Pf = \begin{bmatrix} f_1 \\ f_2 \end{bmatrix} \tag{5.18}$$

La deuxième équation du système (5.17) a une seule solution, qui est donnée par le lemme suivant :

**Lemme 2.** *Soit $\nu$ l'indice de nilpotence de $N$, lorsque $f \in C^\nu$ l'équation $N\dot{x_2} = x_2 + f_2$ a une unique solution définie par*

$$x_2 = \left( N\frac{d}{dt} - I \right)^{-1} f_2 - \sum_{i=0}^{\nu-1} N^i \frac{d^i f_2}{dt^i} \tag{5.19}$$

Une démonstration de ce lemme est disponible dans [84, p. 17]. Remarquons cependant que la solution de cette dernière équation est unique, et donne la solution initiale consistante à $t_0$. Le degré de nilpotence $\nu$ est donc une caractéristique très importante de l'équation différentielle algébrique puisque le calcul de la solution demandera d'expliciter les $\nu - 1$ premières dérivées de la fonction $f_2$. Ce degré de nilpotence $\nu$ de la matrice $N$, aussi appelé indice de Kronecker, définit l'indice de l'EDA.

Il convient de définir une notion d'indice similaire pour des formes plus générales d'EDA. Les deux définitions d'indice les plus présentes dans la littérature sont l'indice de différentiation et l'indice de perturbation : l'indice de différentiation est utilisé pour analyser la convergence des méthodes du type *différentiation rétrograde* et plus généralement des méthodes à pas multiples. L'indice de perturbation est quant à lui utilisé dans l'analyse de la convergence des méthodes de *Runge-Kutta implicites*.

La notion d'indice différentiel, ou indice de differentiation, a été introduite pour définir, dans un certain sens, la distance d'une EDA de la forme $F(t, x, \dot{x}) = 0$ par rapport à une EDO dont l'indice est $0$ par construction. Cette notion est importante car elle traduit les difficultés analytiques et numériques à résoudre les EDA : plus l'indice différentiel est élevé, plus l'EDA sera difficile à résoudre. La plupart des méthodes numériques disponibles permettent de résoudre uniquement les systèmes d'EDA d'indice 1, et des formes particulières d'EDA d'indice 2. Des méthodes analytiques (ou du moins symboliques) sont souvent utilisées pour réduire l'indice d'un système. On notera tout de même que l'apparition de systèmes d'EDA d'indice supérieur à trois est rare.

La définition de l'indice différentiel donnée dans [19, p. 17] est la suivante :

**Définition 1.** *L'indice différentiel est le nombre minimal de fois que tout ou une partie de $F(t, \dot{y}, y)$ doit être dérivée par rapport à $t$ pour pouvoir déterminer $\dot{y}$ comme une fonction continue de $y$ et $t$.*

Cette notion s'impose lorsque l'on cherche à transformer l'EDA en EDO afin de pouvoir utiliser les solveurs d'EDO disponibles. L'EDO obtenue par cette transformation est appelée EDO *sous-jacente*, la solution de l'EDA sera sur une variété invariante de la solution de l'EDO.

#### 5.3.1.2 Structures semi-explicites et implicites

La forme la plus générale d'EDA est donnée par formulation implicite $F(t, x, \dot{x}) = 0$. Lorsque l'on peut réécrire cette EDA de manière à séparer les variables algébriques des variables différentielles, on parlera d'EDA semi-explicite. On écrira ces EDA semi-explicites sous la forme (5.20) où $y$ est la variable différentielle et $z$ la variable algébrique.

$$\left\{ \begin{array}{rcl} \dot{y} & = & h(t, y, z) \\ 0 & = & g(t, y, z) \end{array} \right. \tag{5.20}$$

On peut faire apparaître un lien entre les EDO et les EDA semi-explicites en régularisant ces dernières à l'aide d'un paramètre $\epsilon$ proche de zéro :

$$\left\{ \begin{array}{rcl} \dot{y} & = & h(t, y, z) \\ \epsilon \dot{z} & = & g(t, y, z). \end{array} \right. \tag{5.21}$$

Cette régularisation n'est généralement pas utilisée pour résoudre numériquement les EDA semi-explicites à cause de la raideur de l'EDO obtenue. Après différentiation de l'équation (5.20) par rapport à $t$, on obtient :

$$\left\{ \begin{array}{rcl} \dot{y} & = & h(t, y, z) \\ g_y(t, y, z)\dot{y} + g_z(t, y, z)\dot{z} & = & -g_t(t, y, z) \end{array} \right. \tag{5.22}$$

Lorsque la jacobienne $g_z$ est non singulière, on obtient l'EDO implicite (5.22). Dans ce cas l'indice de différentiation de l'équation (5.20) est 1. En pratique, on pourra se contenter

de la non-singularité de $g_z$ uniquement au voisinage d'un triplet $(t^*, y^*, z^*)$ satisfaisant la contrainte algébrique. Finalement, on obtient l'EDO sous-jacente donnée par :

$$\begin{cases} \dot{y} &= h(t,y,z) \\ \dot{z} &= -g_z(t,y,z)^{-1}(g_y(t,y,z)\dot{y} + g_t(t,y,z)). \end{cases} \tag{5.23}$$

Lorsqu'une EDA a un indice élevé, il peut être judicieux de réduire son indice de manière à pouvoir effectuer une résolution numérique par des solveurs fonctionnant pour des EDA d'indice 1. Ces techniques sont généralement basées sur la différentiation de certaines équations du système, ce qui peut conduire à une perte de contraintes. En effet, les contraintes algébriques peuvent ne plus être satisfaites exactement, puisque l'espace des solutions du système d'indice réduit est plus large que celui du système original. Cependant, il est parfois possible de réintroduire les contraintes perdues dans le système par des manipulations analytiques.

### 5.3.2 Recherche d'une solution initiale

En pratique, la résolution des EDA pose plus de problèmes numériques que celle des EDO. Intuitivement cela peut s'expliquer par le fait que lorsque le pas de temps tend vers 0, la matrice jacobienne tend vers une matrice singulière. L'autre différence majeure entre les EDO et les EDA est que le calcul d'une solution initiale consistante n'est plus naturel. Un exemple proposé par SINCOVEC et al. [113] permet d'illustrer rapidement le problème :

**Exemple 2.** *Considérons le système d'équations :*

$$\begin{cases} \dot{x_1} + 2\dot{x_2} &= x_1 - x_2 \\ \dot{x_1} + 2\dot{x_2} &= 3x_1 - 2x_2 - 1. \end{cases} \tag{5.24}$$

*Les valeurs initiales $x_1(0) = 2$ et $x_2(0) = 1$ sont inconsistantes, car en soustrayant la deuxième équation de la première on obtient les contraintes cachées $2x_1 - x_2 = 1$ et $2\dot{x}_1 = \dot{x}_2$ que la solution doit satisfaire quel que soit $t$. On peut donc écrire le système étendu suivant :*

$$\begin{cases} \dot{x_1} &= 0.2x_1 - 0.2x_2 \\ \dot{x_2} &= 0.4x_1 - 0.4x_2 \\ 0 &= 2x_1 - x_2 - 1 \end{cases} \tag{5.25}$$

Dans la suite, on considère la méthode *Constrained Runs*. D'autres méthodes d'initialisation possibles sont passées en revue en annexe C p.153. Cette méthode à l'avantage de permettre d'imposer des valeurs initiales à certaines variables. Ainsi les inconnues du système d'équations différentielles sont partitionnées en $(u, v)$ où $u$ représente les variables dont l'on souhaite imposer une valeur à $t = 0$. La phase d'initialisation aura pour but de trouver $v$ tel que $(u, v)$ soit une solution initiale consistante. On considère donc l'algorithme 17 issue de [60]. Notons que dans [60], GEAR et KEVREKIDIS proposent une méthode plus générale, où le retour à $t = 0$ se fait par une extrapolation des derniers pas de temps. La méthode présentée

**Algorithme 17** *Constrained Runs* avec restriction

**Require:** a tolerance $\epsilon$, a steplength : $h$
**Require:** $u_0$ fixed, an initial guess for $v_0$
1: **repeat**
2: perform $m+1$ time steps $(u_0, v_0)$ : $(u_j, v_j)$ with $j = 1 \dots m+1$
3: $v_0 \leftarrow v_{m+1}$
4: **until** convergence

en algorithme 17 est constituée d'une succession de pas de temps. Il est donc possible de pipeliner les résolutions de ces pas de temps.

On considère le problème de Fekete qui est celui de trouver une répartition homogène de $N$ points sur une sphère. La résolution de ce problème peut se faire par la méthode des forces, qui consiste à résoudre le système d'équations (4.11) p. 83. Les valeurs initiales sont $q(0) = 0$, $\mu(0) = 0$ et on pose $\alpha = 1/2$. Ce problème semble artificiel dans la mesure où l'on pourrait faire disparaitre la contrainte algébrique en raisonnant en coordonnées polaires. Quoi qu'il en soit, ce genre de manipulations analytiques n'est pas toujours possible, et est difficile à mettre en pratique dans un logiciel de résolution d'EDA. On considèrera ici un nombre de points pairs, et souhaitera imposer que à $t = 0$ la moitié des points soit dans le plan $z = 1/2$ et l'autre dans le plan $z = -1/2$. On utilisera donc la méthode *Constrained Runs* avec comme solution initiale des points tirés aléatoirement dans ces deux plans et à l'intérieur de la sphère. À la fin de l'initialisation, les points se situeront aux intersections de la sphère et de ces deux plans. La figure 5.5 représente les solutions successives de l'algorithme 17 avant et après l'étape de la restriction. Les trajectoires obtenues par la résolution de l'équation (4.11) sont données en figure 5.6.

La parallélisation des itérations de la méthode *Constrained Runs* peut s'écrire comme l'algorithme 18 qui est le même que l'algorithme 15 p.117 auquel on ajoute la contrainte et la remise à zéro du temps.

### 5.3.3 Résultats numériques

On considère la recherche d'une solution initiale consistante du problème de Fekete telle qu'une moitié des points soit dans le plan $z = 1/2$ et l'autre dans le plan $z = -1/2$. Au départ, les coordonnées des points sont $(r_x, r_y, \pm 1/2)$ où $r_x$ et $r_y$ sont des réels aléatoires indépendants pris dans $[0, 1]$. Le générateur de nombres aléatoires utilisera toujours la même graine, quel que soit le nombre de processeurs. Le problème non linéaire de chaque pas de temps est résolu par une méthode de Newton exacte dans le sens où les systèmes linéaires sont résolus par factorisation LU. Les temps de calculs pour une implémentation Fortran/MPI sont donnés dans le tableau 5.2. La tolérance sur le résidu des itérations de Newton est de $10^{-7}$, et l'initialisation est acceptée dès que $\|d\|_2 < 10^{-5}$ où $d = [d_1, d_2, \cdots, d_N]^T$ est le

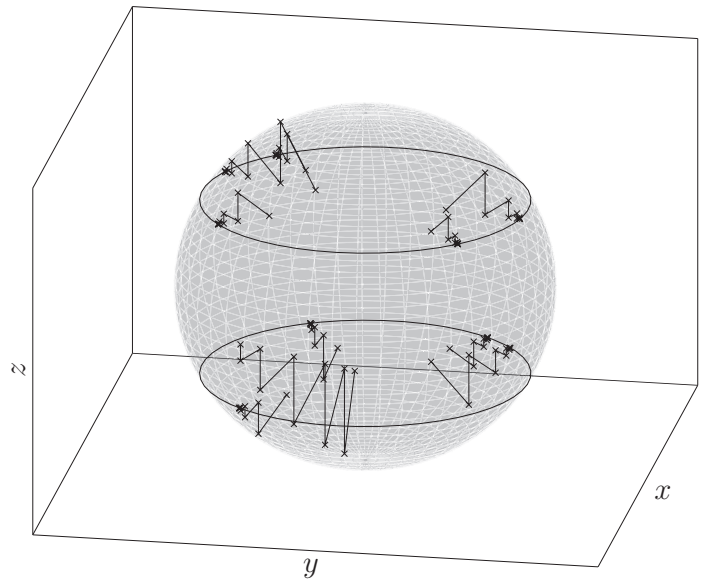


**FIGURE *5.5 –*** *Initialisation par* Constrained Runs *avec restriction : les cinq premières itérations. Les solutions de chaque itération avant et après la restriction sont tracées.*

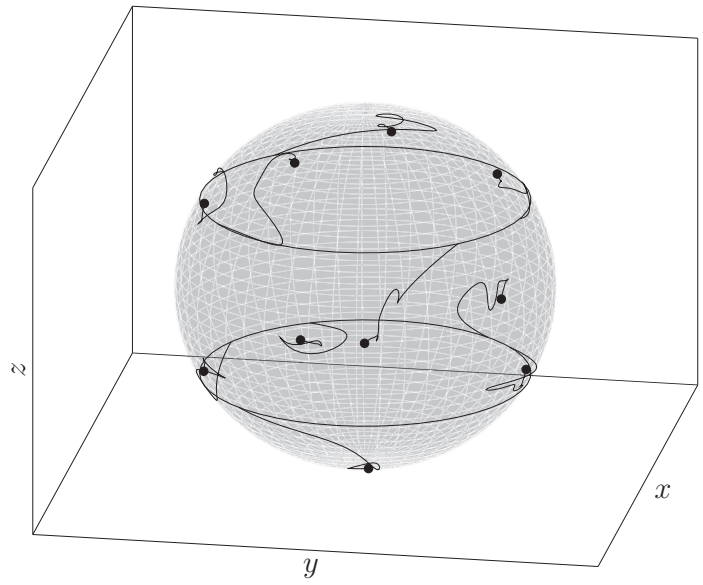


**FIGURE *5.6 –*** *Résolution : comportement chaotique des trajectoires. Les • symbolisent la solution finale.*

**Algorithme 18** Pipelinage des itérations $CR$ avec $s$ processeurs

**Require:** $m \leftarrow$ the number of timesteps per $CR$ iteration
**Require:** $r \leftarrow$ rank of the current process, $s \leftarrow$ number of processes
**Require:** $x_0^i$ for $i = 1...r$
**Require:** an initial time $t_0$ and a time step $\Delta t$
1: `last` $\leftarrow$ `false`, `first` $\leftarrow$ `false`
2: **if** $r = 0$ **then** `first` $\leftarrow$ `true` **end if**
3: **if** $r = s - 1$ **then** `last` $\leftarrow$ `true` **end if**
4: $i \leftarrow 0$
5: **if** not `last` **then** send $x_0^i$ to process $r + 1$ **end if**
6: **repeat**
7: $t = t_0 + \Delta t((s \times i + r) \pmod{m + 1})$
8: **repeat**
9: **if** not `first` **then** receive $x(t - \Delta t)$ from $r - 1 \pmod{s}$ **end if**
10: **if** $i - 1 \pmod{m + 1} = 0$ **then** constrain $x(t - \Delta t)$ **end if**
11: $G \leftarrow$ approximation of $(\partial F/\partial x(t))^{-1}$
12: $x(t) \leftarrow x(t) - GF(x(t), t, x(t - \Delta t))$
13: **if** not `last` **then** send $x(t)$ to $r + 1 \pmod{s}$ **end if**
14: collective update of `first` and `last`
15: **until** convergence
16: $i \leftarrow i + 1$
17: **until** final time

vecteur des distances à la sphère des points contraints : $d_i = x_i^2 + y_i^2 + 0.5^2 - 1$.

**TABLE 5.2 –** *Temps et accélération obtenus pour l'initialisation du problème de Fekete. Implémentation Fortran90/MPI.*

| | 1 CPU | 2 CPU | | 3 CPU | |
|---|---|---|---|---|---|
| Points | Time (s) | Time (s) | speedup | Time (s) | speedup |
| 100 | 3.03 | 2.19 | 1.39 | 1.97 | 1.54 |
| 200 | 22.71 | 15.66 | 1.45 | 13.99 | 1.63 |
| 300 | 75.26 | 50.55 | 1.49 | 44.45 | 1.69 |
| 400 | 179.52 | 117.48 | 1.53 | 104.46 | 1.72 |
| 500 | 363.89 | 221.02 | 1.65 | 201.26 | 1.81 |
| 600 | 736.49 | 441.87 | 1.67 | 392.63 | 1.88 |
| 700 | 1161.24 | 697.69 | 1.66 | 648.24 | 1.79 |
| 800 | 1715.23 | 1076.70 | 1.59 | 921.27 | 1.86 |

Les tests ont été limités à $3$ processeurs étant donné que la plupart des systèmes non linéaires sont résolus en moins de $4$ itérations de Newton. L'augmentation du temps de calcul en fonction du nombre de points s'explique en majeure partie par l'utilisation de factorisations LU. En effet, le nombre total d'itérations varie peu : en séquentiel, l'initialisation a requis 80 itérations de Newton pour le problème à 100 points, et 109 pour le problème à 800 points. En utilisant deux processeurs, le nombre total d'itérations de Newton a été de 130 pour le problème à 800 points.

## 5.4 Conclusion

La parallélisation à travers le temps, c'est-à-dire la décomposition du domaine temporel, peut être un bon moyen d'accélérer les calculs. Cependant, la nature séquentielle des équations limite l'efficacité de la parallélisation à travers le temps. Dans le cas d'un problème non linéaire, le pipelinage des pas de temps consiste ici à démarrer le solveur non linéaire du temps $t_i$ alors que la convergence du solveur non linéaire du pas de temps $t_{i-1}$ n'est pas achevée. Il a été proposé de pipeliner les pas de temps effectués lors de l'initialisation des EDA par la méthode *Constrained Runs*. Une implémentation MPI de la méthode a été testée sur un problème de Fekete, elle a permis de diminuer le temps de calcul d'environ $35\%$ lorsque $2$ processeurs sont utilisés. La parallélisation en temps a également été étudiée lorsqu'un solveur de type quasi-Newton est utilisé. Dans ce cas, l'efficacité numérique peut être augmentée en propageant les mises à jour de Broyden.

# Chapitre 6

# Conclusions et perspectives

Tout au long de ce manuscrit, nous nous sommes attachés à trouver de nouvelles sources de parallélisme au sein des méthodes de décomposition de domaine. Dans un premier temps, nous avons considéré la résolution de très grands systèmes linéaires creux sur des architectures massivement parallèles. Nous avons proposé et implémenté une nouvelle méthode d'accélération d'Aitken de la convergence des méthodes de décomposition de domaine de type Schwarz. Cette méthode consiste à estimer indépendamment chaque bloc de l'opérateur de propagation d'erreur. Dans le cas d'un problème d'écoulement en milieu hétérogène, la méthode proposée s'avère bien plus performante et extensible que la méthode d'Aitken-Schwarz classique lorsque le nombre de sous-domaines augmente. La méthode, bien que purement algébrique, a été implémentée et testée jusqu'à $4096$ cœurs dans le cas d'une grille régulière 3D en vue d'une intégration au logiciel H2OLAB. Une implémentation adaptée à un opérateur linéaire quelconque partitionné par Metis est cependant en cours de développement.

Par la suite, le préconditionneur RAS a été considéré dans le cas d'une suite de systèmes linéaires, où l'opérateur varie peu. De telles suites de systèmes linéaires apparaissent naturellement lorsque des équations différentielles algébriques ou ordinaires sont résolues par une méthode implicite. Dans ce cas, le même préconditionneur peut être utilisé pour plusieurs systèmes consécutifs. Cette méthode, qui permet d'économiser du temps de calcul, peut être améliorée. On a donc discuté de deux mises à jour du préconditionneur RAS : il a d'abord été proposé d'appliquer les mises à jour de rang 1 de quasi-Newton au préconditionneur. Le préconditionneur RAS a ensuite été mis à jour partiellement. Ce préconditionneur RAS mis à jour partiellement a été nommé AsRAS pour *Asynchronous Restricted Additive Schwarz* car son implémentation conduit à un algorithme asynchrone. Dans cet algorithme asynchrone, certains processeurs sont dédiés à la mise à jour du préconditionneur. Cette mise à jour sera communiquée aux processeurs effectuant la résolution classique du problème. Ainsi l'algorithme proposé est du type client-serveur : les processus en charge d'un sous-domaine sont les clients que le serveur approvisionne en mises à jour. Cette méthode nous a permis d'obtenir des accélérations superlinéaires des temps de calcul car les mises à jour permettent de réduire le nombre total d'itérations de Krylov de la simulation. Dans les tests de ce manuscrit, les mises à jour sont distribuées de manière cyclique aux processus clients, on pourrait cependant effectuer plus souvent les mises à jour des sous-domaines dans lesquels la physique

est la plus non linéaire. Il a été considéré uniquement le cas où le préconditionneur RAS est composé de la factorisation LU des opérateurs locaux. Par la suite, on pourrait envisager d'étendre cette méthode aux factorisations ILU, qui seraient éventuellement calculées sur un GPU.

La dernière partie de ce manuscrit a été l'occasion d'explorer le parallélisme à travers les pas de temps. Cette méthode peut être vue comme une méthode de décomposition du domaine temporel, où chaque domaine est composé d'un seul pas de temps. Elle consiste à débuter la résolution du problème à un pas de temps donné, alors que les problèmes des pas de temps précédents ne sont pas entièrement résolus. Cette technique a été appliquée au cas où la résolution des problèmes de chaque pas de temps s'effectue par une méthode de quasi-Newton. On a ainsi proposé de propager les mises à jour de Broyden d'un pas de temps à l'autre. La parallélisation des pas de temps a également été adaptée à la recherche d'une solution initiale consistante des équations différentielles algébriques par la méthode *Constrained Runs*. Les résultats numériques ont montré que cette méthode permet d'accélérer significativement les calculs, notamment lors de la résolution du problème de Fekete par la méthode des forces où la décomposition du domaine spatial n'est pas envisageable.

# Bibliographie


[1] P. R. AMESTOY, I. S. DUFF, J.-Y. LÉXCELLENT et J. KOSTER. “MUMPS : a general purpose distributed memory sparse solver”. Dans : *Applied Parallel Computing. New Paradigms for HPC in Industry and Academia*. Springer, 2001, p. 121–130.

[2] C. AREVALO, S. L. CAMPBELL et M. SELVA. “Unitary partitioning in general constraint preserving DAE integrators”. Dans : *Mathematical and Computer Modelling* 40.11 (2004), p. 1273–1284.

[3] J. BAHI et J.-C. MIELLOU. “Asynchronous multisplitting algorithms for differential-algebraic systems discretized by Runge-Kutta methods”. Dans : *Proceedings of the Eighth International Colloquium on Differential Equations (Plovdiv, 1997)*. Utrecht : VSP, 1998, p. 35–41.

[4] J. BAHI, J. C. MIELLOU et K. RHOFIR. “Asynchronous multisplitting methods for nonlinear fixed point problems”. Dans : *Numer. Algorithms* 15.3-4 (1997), p. 315–345. ISSN : 1017-1398.

[5] S. BALAY et al. *PETSc Users Manual*. Rap. tech. ANL-95/11 - Revision 3.3. Argonne National Laboratory, 2012.

[6] A. BELLEN. “Parallelism across the steps for difference and differential equations”. Dans : *Numerical methods for ordinary differential equations (L’Aquila, 1987)*. T. 1386. Lecture Notes in Math. Berlin : Springer, 1989, p. 22–35.

[7] M. BENZI et D. BERTACCINI. “Approximate inverse preconditioning for shifted linear systems”. Dans : *BIT Numerical Mathematics* 43.2 (2003), p. 231–244.

[8] L. BERENGUER, T. DUFAUD, T. P. DUC et D. TROMEUR-DERVOUT. “On-the-fly Singular Value Decomposition for Aitken’s Acceleration of the Schwarz Domain Decomposition Method”. Dans : *Applications Tools and Techniques on the Road to Exascale Computing*. Sous la dir. de K. D. BOSSCHERE, E. H. D’HOLLANDER, G. R. JOUBERT, D. P. F. PETERS et M. SAWYER. T. 22. Advances in Parallel Computing. Amsterdam : IOS press, 2012, p. 243–250. DOI : 10.3233/978-1-61499-041-3-243. URL : http://dx.doi.org/10.3233/978-1-61499-041-3-243.

[9] L. BERENGUER, T. DUFAUD et D. TROMEUR-DERVOUT. “Acceleration of Convergence for Domain Decomposition Methods”. Dans : *Computational Technology Reviews* 7 (2013), p. 1–24. URL : http://dx.doi.org/10.4203/ctr.7.1.

[10] L. BERENGUER et D. TROMEUR-DERVOUT. “Numerical and Parallel Efficiency of a purely algebraic linear solver for Large-scale groudwater flow problems”. Dans : *Parallel CFD 2014, Parallel Computational Fluids Dynamics*. Sous la dir. de T. KVAMSDAL, A. KVARVING, H. HOLM, C. JENSSEN, M. KUMAR et B. PETTERSEN. CIMNE, 2014, p. 75–76.

[11] L. BERENGUER, T. DUFAUD et D. TROMEUR-DERVOUT. “Aitken’s acceleration of the Schwarz process using singular value decomposition for heterogeneous 3D groundwater flow problems”. Dans : *Computer & Fluids* 80 (2013), p. 320–326. DOI : 10.1016/j.compfluid.2012.01.026. URL : http://dx.doi.org/10.1016/j.compfluid.2012.01.026.

[12] L. BERENGUER et D. TROMEUR-DERVOUT. “Asynchronous Partial Update of Domain Decomposition Preconditioners to Solve Nonlinear CFD Problems”. Dans : *25th International Conference on Parallel Computational Fluid Dynamics*. T. 61. Procedia Engineering, 2013, p. 130–135.

[13] L. BERENGUER et D. TROMEUR-DERVOUT. “Developments on the Broyden procedure to solve nonlinear problems arising in CFD”. Dans : *Computers & Fluids* (2013). ISSN : 0045-7930.

[14] L. BERENGUER et D. TROMEUR-DERVOUT. “Low-Rank Update of the Restricted Additive Schwarz Preconditioner for Nonlinear Systems”. Dans : *21th Int. Conf. on Domain Decomposition Methods DD21*. Springer, LNCSE collection, 2014, To appear.

[15] L. BERENGUER et D. TROMEUR-DERVOUT. “Partially Updated Restricted Additive Schwarz Preconditioner”. Dans : *22th Int. Conf. on Domain Decomposition Methods DD22*. Accepted.

[16] L. BERGAMASCHI, R. BRU et A. MARTINEZ. “Low-rank update of preconditioners for the inexact Newton method with SPD Jacobian”. Dans : *Math. Comput. Modelling* 54.7-8 (2011), p. 1863–1873.

[17] L. BERGAMASCHI, R. BRU, A. MARTINEZ et M. PUTTI. “Quasi-Newton preconditioners for the inexact Newton method”. Dans : *Electron. Trans. Numer. Anal.* 23 (2006), 76–87 (electronic). ISSN : 1068-9613.

[18] P. BIRKEN, J. DUINTJER TEBBENS, A. MEISTER et M. TUMA. “Preconditioner updates applied to CFD model problems”. Dans : *Applied Numerical Mathematics* 58.11 (2008), p. 1628–1641.

[19] K. BRENAN, S. CAMPBELL et L. PETZOLD. *Numerical Solution of Initial-Value Problems in Differential-Algebraic Equations*. second edition. SIAM, 1996.

[20] I. BREZANI et F. ZELENAK. “Improving the effectivity of work with Rosin-Rammler diagram by using MATLAB (R) GUI tool”. Dans : *Acta Montanistica Slovaca* 15.2 (2010), p. 152–157.

[21] P. BROWN, A. HINDMARSH et L. PETZOLD. “Consistent Initial Condition Calculation for Differential-Algebraic Systems”. Dans : *SIAM J. Sci. Comput.* 19 (1998), p. 1495–1512.

[22] C. G. BROYDEN, J. E. DENNIS Jr. et J. J. MORE. “On the local and superlinear convergence of quasi-Newton methods”. Dans : *J. Inst. Math. Appl.* 12 (1973), p. 223–245.

[23] X.-C. CAI et M. SARKIS. “A restricted additive Schwarz preconditioner for general sparse linear systems”. Dans : *SIAM J. Sci. Comput.* 21.2 (1999), 792–797 (electronic). ISSN : 1064-8275.

[24] X.-C. CAI, W. D. GROPP, D. E. KEYES et T. M. D. “Newton-Krylov-Schwarz methods in CFD”. Dans : *Proceedings of the International Workshop on Numerical Methods for the Navier-Stokes Equations*. Braunschweig : Vieweg, 1995, p. 17–30.

[25] X.-C. CAI et D. E. KEYES. “Nonlinearly preconditioned inexact Newton algorithms”. Dans : *SIAM Journal on Scientific Computing* 24.1 (2002), p. 183–200.

[26] X.-C. CAI, D. E. KEYES et V. VENKATAKRISHNAN. “Newton-Krylov-Schwarz : An Implicit Solver for CFD.” Dans : *Proceedings of the Eighth International Conference on Domain Decomposition Methods*. New York : Wiley, 1997, p. 387–400.

[27] X.-C. CAI et M. SARKIS. “A restricted additive Schwarz preconditioner for general sparse linear systems”. Dans : *SIAM J. Sci. Comput.* 21.2 (1999), 792–797 (electronic). ISSN : 1064-8275.

[28] S. CAMPBELL, C. KELLEY et K. YEOMANS. “Consistent initial conditions for unstructured higher index DAEs : A computational study”. Dans : *Proc. Conference on Computational Engineering in Systems Applications (CESA'96), Lille, France*. Citeseer. 1996, p. 416–421.

[29] S. L. CAMPBELL et E. MOORE. “Constraint preserving integrators for general nonlinear higher index DAEs”. Dans : *Numerische Mathematik* 69.4 (1995), p. 383–399.

[30] T. F. CHAN et K. R. JACKSON. “Nonlinearly preconditioned Krylov subspace methods for discrete Newton algorithms”. Dans : *SIAM Journal on scientific and statistical computing* 5.3 (1984), p. 533–542.

[31] A. CHAPMAN et Y. SAAD. “Deflated and augmented Krylov subspace techniques”. Dans : *Numerical linear algebra with applications* 4.1 (1997), p. 43–66.

[32] P. CHEVALIER et F. NATAF. “An optimized order 2 (OO2) method for the Helmholtz equation”. Dans : *C. R. Acad. Sci. Paris Sér. I Math.* 326.6 (1998), p. 769–774. ISSN : 0764-4442. DOI : 10.1016/S0764-4442(98)80047-5. URL : http://dx.doi.org/10.1016/S0764-4442(98)80047-5.

[33] T. S. COFFEY, C. T. KELLEY et D. E. KEYES. “Pseudotransient continuation and differential-algebraic equations”. Dans : *SIAM J. Sci. Comput.* 25.2 (2003), p. 553–569. ISSN : 1064-8275.

[34] T. F. COLEMAN et A. VERMA. “Structure and efficient Jacobian calculation”. Dans : *Computational Differentiation : Techniques, Applications, and Tools*. SIAM, 1996, p. 149–159.

[35] M. CROUZEIX et F. LISBONA. “The convergence of variable-stepsize, variable-formula multistep methods”. Dans : *SIAM J. Numer. Anal.* 21 (1984), p. 512–533.

[36] A. CURTIS, M. J. POWELL et J. K. REID. “On the estimation of sparse Jacobian matrices”. Dans : *J. Inst. Math. Appl* 13.1 (1974), p. 117–120.

[37] A. ST-CYR, M. J. GANDER et S. J. THOMAS. “Optimized multiplicative, additive, and restricted additive Schwarz preconditioning”. Dans : *SIAM J. Sci. Comput.* 29.6 (2007), 2402–2425 (electronic). ISSN : 1064-8275. DOI : 10.1137/060652610. URL : http://dx.doi.org/10.1137/060652610.

[38] J. DEMMEL, L. GRIGORI, M. HOEMMEN et J. LANGOU. “Communication-optimal parallel and sequential QR and LU factorizations”. Dans : *SIAM Journal on Scientific Computing* 34.1 (2012), A206–A239.

[39] J. E. DENNIS Jr. et J. J. MORÉ. “Quasi-Newton methods, motivation and theory”. Dans : *SIAM Rev.* 19.1 (1977), p. 46–89. ISSN : 0036-1445.

[40] J. DENNIS Jr et R. SCHNABEL. “Least change secant updates for quasi-Newton methods”. Dans : *Siam Review* 21.4 (1979), p. 443–459.

[41] J. E. DENNIS Jr et E. S. MARWILL. “Direct secant updates matrix factorizations”. Dans : *Math. Comp.* 38(158) (1982), p. 459–476.

[42] J. E. DENNIS Jr et H. F. WALKER. “Convergence theorems for least-change secant update methods”. Dans : *SIAM Journal on Numerical Analysis* 18.6 (1981), p. 949–987.

[43] P. DEUFLHARD. *Newton Methods for Nonlinear Problems*. Sous la dir. de NEW-YORK. Springer, 2004.

[44] T. P. DUC et D. TROMEUR-DERVOUT. “Proper Orthogonal Decomposition in Decoupling Dynamical Systems”. Dans : *Proceedings of the Second International Conference on Parallel Distributed Grid and Cloud Computing for Engineering*. Sous la dir. de B. TOPPING et P. IVANYI. T. 95 :paper 55. Civil-Comp Proceedings. Civil-Comp Press, 2011. DOI : 10.4203/ccp.95.55. URL : http://dx.doi.org/10.4203/ccp.95.55.

[45] T. DUFAUD et D. TROMEUR-DERVOUT. “Aitken’s acceleration of the restricted additive Schwarz preconditioning using coarse approximations on the interface”. Dans : *C. R. Math. Acad. Sci. Paris* 348.13-14 (2010), p. 821–824. ISSN : 1631-073X. DOI : 10.1016/j.crma.2010.06.021. URL : http://dx.doi.org/10.1016/j.crma.2010.06.021.

[46] J. DUINTJER TEBBENS et M. TUMA. “Preconditioner updates for solving sequences of linear systems in matrix-free environment”. Dans : *Numer. Linear Algebra Appl.* 17 (2010), p. 997–1019.

[47] S. C. EISENSTAT et H. F. WALKER. “Globally convergent inexact Newton methods”. Dans : *SIAM Journal on Optimization* 4.2 (1994), p. 393–422.

[48] B. ENGQUIST et H.-K. ZHAO. “Absorbing boundary conditions for domain decomposition”. Dans : *Appl. Numer. Math.* 27.4 (1998). Absorbing boundary conditions, p. 341–365. ISSN : 0168-9274. DOI : 10.1016/S0168-9274(98)00019-1. URL : http://dx.doi.org/10.1016/S0168-9274(98)00019-1.

[49] J. ERHEL, K. BURRAGE et B. POHL. “Restarted GMRES preconditioned by deflation”. Dans : *Journal of computational and applied mathematics* 69.2 (1996), p. 303–318.

[50] R. D. FALGOUT, J. E. JONES et U. M. YANG. “The design and implementation of hypre, a library of parallel high performance preconditioners”. Dans : *Numerical solution of partial differential equations on parallel computers*. Springer, 2006, p. 267–294.

[51] A. FROMMER et D. B. SZYLD. “On asynchronous iterations”. Dans : *J. Comput. Appl. Math.* 123.1-2 (2000). Numerical analysis 2000, Vol. III. Linear algebra, p. 201–216. ISSN : 0377-0427.

[52] A. FRULLONE et D. TROMEUR-DERVOUT. “A new formulation of NUDFT applied to Aitken-Schwarz DDM on nonuniform meshes”. Dans : *Parallel Computational Fluid Dynamics 2005*. 2006, p. 493–500.

[53] M. J. GANDER. “Schwarz methods over the course of time”. Dans : *Electronic Transactions on Numerical Analysis* 31.228-255 (2008), p. 5.

[54] M. J. GANDER et S. VANDEWALLE. “Analysis of the parareal time-parallel time-integration method”. Dans : *SIAM Journal on Scientific Computing* 29.2 (2007), p. 556–578.

[55] M. J. GANDER, F. MAGOULÈS et F. NATAF. “Optimized Schwarz methods without overlap for the Helmholtz equation”. Dans : *SIAM J. Sci. Comput.* 24.1 (2002), 38–60 (electronic). ISSN : 1064-8275. DOI : 10.1137/S1064827501387012. URL : http://dx.doi.org/10.1137/S1064827501387012.

[56] M. GARBEY et D. TROMEUR-DERVOUT. “On some Aitken-like acceleration of the Schwarz method”. Dans : *Internat. J. Numer. Methods Fluids* 40.12 (2002). LMS Workshop on Domain Decomposition Methods in Fluid Mechanics (London, 2001), p. 1493–1513. ISSN : 0271-2091. DOI : 10.1002/fld.407. URL : http://dx.doi.org/10.1002/fld.407.

[57] M. GARBEY et D. TROMEUR-DERVOUT. “Two level domain decomposition for Multi-clusters”. Dans : *12th Int. Conf. on Domain Decomposition Methods DD12*. Sous la dir. de T. C. Ã. EDITORS. ddm.org, 2001, p. 325–339.

[58] C. GEAR. “Differential-Algebraic equations index transformations”. Dans : *SIAM J.sci. Statist. Comput.* 9 (1988), p. 39–47.

[59] C. GEAR, H. HSU et L. PETZOLD. “Differential-algebraic equations revisited”. Dans : *Proc. ODE Meeting, Oberwolfach, West Germany*. 1981.

[60] C. W. GEAR et I. G. KEVREKIDIS. “Constraint-defined manifolds : a legacy code approach to low-dimensional computation”. Dans : *Journal of Scientific Computing* 25.1 (2005), p. 17–28.

[61] C. GEAR, T. KAPER, I. KEVREKIDIS et A. ZAGARIS. “Projecting to a Slow Manifold : Singularly Perturbed Systems and Legacy Codes”. Dans : *SIAM J. Appl. Dyn. Syst.* 4 (2005), p. 711–732.

[62] C. GEAR et I. G. KEVREKIDIS. “Computing in the past with forward integration”. Dans : *Physics Letters A* 321.5 (2004), p. 335–343.

[63] C. GEAR, B. LEIMKUHLER et G. GUPTA. “Automatic integration of Euler-Lagrange equations with constraints”. Dans : *Journal of Computational and Applied Mathematics* 12 (1985), p. 77–90.

[64] L. GEAR C. Petzold. “Differential/algebraic systems and matrix pencils”. Dans : *Lecture Notes in Mathematics* 973 (1983), p. 75–89.

[65] A. H. GEBREMEDHIN, F. MANNE et A. POTHEN. “What color is your Jacobian ? Graph coloring for computing derivatives”. Dans : *SIAM Rev.* 47.4 (2005), 629–705 (electronic). ISSN : 0036-1445.

[66] L. GERARDO-GIORDA et F. NATAF. “Optimized Schwarz methods for unsymmetric layered problems with strongly discontinuous and anisotropic coefficients”. Dans : *J. Numer. Math.* 13.4 (2005), p. 265–294. ISSN : 1570-2820. DOI : 10.1163/156939505775248338.

[67] L. GIRAUD, A. HAIDAR et Y. SAAD. *Sparse approximations of the Schur complement for parallel algebraic hybrid linear solvers in 3D*. Rapport de recherche RR-7237. INRIA, mar. 2010. URL : http://hal.inria.fr/inria-00466828.

[68] G. H. GOLUB. “Some modified matrix eigenvalue problems”. Dans : *Siam Review* 15.2 (1973), p. 318–334.

[69] G. H. GOLUB et C. F. VAN LOAN. *Matrix computations*. T. 3. JHU Press, 1996.

[70] C. GREIF et J. M. VARAH. “Minimizing the condition number for small rank modifications”. Dans : *SIAM J. Matrix Anal. Appl.* 29.1 (2006/07), 82–97 (electronic). ISSN : 0895-4798.

[71] A. GRIEWANK et A. WALTHER. *Evaluating derivatives : principles and techniques of algorithmic differentiation*. Siam, 2008.

[72] D. GRIGORIEFF. “Stability of multistep methods on variable grids”. Dans : *Numer. Math.* 42 (1983), p. 359–377.

[73] W. HACKBUSCH. *Multi-grid methods and applications*. T. 4. Springer-Verlag Berlin, 1985.

[74] P. HALL, D. MARSHALL et R. MARTIN. “Adding and subtracting eigenspaces with eigenvalue decomposition and singular value decomposition”. Dans : *Image and Vision Computing* 20.13 (2002), p. 1009–1016.

[75] P. HALL, D. MARSHALL et R. MARTIN. “Merging and splitting eigenspace models”. Dans : *Pattern Analysis and Machine Intelligence, IEEE Transactions on* 22.9 (2000), p. 1042–1049.

[76] H. A. SCHWARZ. “Vierteljahresschrift der Naturforschenden Gesellschaft”. Dans : *Zurich* 15 (1870), p. 272–286.

[77] V. HERNANDEZ, J. E. ROMAN et V. VIDAL. “SLEPc : a scalable and flexible toolkit for the solution of eigenvalue problems”. Dans : *ACM Trans. Math. Software* 31.3 (2005), p. 351–362. ISSN : 0098-3500. DOI : 10.1145/1089014.1089019. URL : http://dx.doi.org/10.1145/1089014.1089019.

[78] C. T. KELLEY. *Iterative Methods for Linear and Nonlinear Equations*. Sous la dir. de P. SIAM. 1995.

[79] C. T. KELLEY. *Solving nonlinear equations with Newton’s method*. T. 1. Fundamentals of Algorithms. Philadelphia, PA : Society for Industrial et Applied Mathematics (SIAM), 2003, p. xiv+104. ISBN : 0-89871-546-6.

[80] D. E. KEYES. “Aerodynamic applications of Newton-Krylov-Schwarz solvers”. Dans : *Fourteenth International Conference on Numerical Methods in Fluid Dynamics (Bangalore, 1994)*. T. 453. Lecture Notes in Phys. Berlin : Springer, 1995, p. 1–20.

[81] M. KIEHL. “Parallel multiple shooting for the solution of initial value problems”. Dans : *Parallel computing* 20.3 (1994), p. 275–295.

[82] D. A. KNOLL et D. E. KEYES. “Jacobian-free Newton-Krylov Methods : A Survey of Approaches and Applications”. Dans : *J. Comp. Phys.* 193 (2004), p. 357–397.

[83] A. KRONER, W. MARQUARDT et E. GILLES. “Computing Consistent Initial Conditions For Differential-Algebraic Equations”. Dans : *Comput. Chem. Engrg.* 16 (1992), p. 131–138.

[84] P. KUNKEL et V. MEHRMANN. *Differential-Algebraic Equation, Analysis and Numerical Solution*. European Mathematical Society, 2006.

[85] R. LAMOUR. “A shooting method for fully implicit index-2 differential algebraic equations”. Dans : *SIAM Journal on Scientific Computing* 18.1 (1997), p. 94–114.

[86] B. LEIMKUHLER, L. R. PETZOLD et C. W. GEAR. “Approximation methods for the consistent initialization of differential-algebraic equations”. Dans : *SIAM Journal on Numerical Analysis* 28.1 (1991), p. 205–226.

[87] J. LIONS, Y. MADAY et G. TURINICI. “A”parareal”in time discretization of PDE’s”. Dans : *Comptes Rendus de l’Academie des Sciences Series I Mathematics* 332.7 (2001), p. 661–668.

[88] P.-L. LIONS. “On the Schwarz alternating method. I”. Dans : *First International Symposium on Domain Decomposition Methods for Partial Differential Equations (Paris, 1987)*. Philadelphia, PA : SIAM, 1988, p. 1–42.

[89] P.-L. LIONS. “On the Schwarz alternating method. III. A variant for nonoverlapping subdomains”. Dans : *Third International Symposium on Domain Decomposition Methods for Partial Differential Equations (Houston, TX, 1989)*. Philadelphia, PA : SIAM, 1990, p. 202–223.

[90] X. S. LI, J. W. DEMMEL, J. R. GILBERT, L. GRIGORI, M. SHAO et I. YAMAZAKI. *SuperLU Users’ Guide*. Rap. tech. Lawrence Berkeley National Laboratory, 2011.

[91] F. MAGOULÈS. “Asynchronous Schwarz Methods for Peta and Exascale Computing”. Dans : *Developments in Parallel, Distributed, Grid and Cloud Computing for Engineering*. Sous la dir. de TOPPING, BHV AND IVANYI, P. T. 10. S, 2013, p. 229–248.

[92] F. MAGOULÈS, F.-X. ROUX et L. SERIES. “Algebraic approximation of Dirichlet-to-Neumann maps for the equations of linear elasticity”. Dans : *Computer Methods in Applied Mechanics and Engineering* 195.29 (2006), p. 3742–3759.

[93] L. D. MARINI et A. QUARTERONI. “A relaxation procedure for domain decomposition methods using finite elements”. Dans : *Numer. Math.* 55.5 (1989), p. 575–598. ISSN : 0029-599X. DOI : 10.1007/BF01398917. URL : http://dx.doi.org/10.1007/BF01398917.

[94] J. M. MARTÍNEZ. “An extension of the theory of secant preconditioners”. Dans : *J. Comput. Appl. Math.* 60.1-2 (1995). Linear/nonlinear iterative methods and verification of solution (Matsuyama, 1993), p. 115–125. ISSN : 0377-0427.

[95] F. NATAF, H. XIANG, V. DOLEAN et N. SPILLANE. “A coarse space construction based on local Dirichlet-to-Neumann maps”. Dans : *SIAM Journal on Scientific Computing* 33.4 (2011), p. 1623–1642.

[96] R. NATARAJAN. “Domain decomposition using spectral expansions of Steklov-Poincaré operators. II. A matrix formulation”. Dans : *SIAM J. Sci. Comput.* 18.4 (1997), p. 1187–1199. ISSN : 1064-8275. DOI : 10.1137/S1064827594274309.

[97] J. NEČAS. “Sur une méthode pour résoudre les équations aux dérivées partielles du type elliptique, voisine de la variationnelle”. Dans : *Ann. Scuola Norm. Sup. Pisa (3)* 16 (1962), p. 305–326.

[98] G. N. NEWSAM et J. D. RAMSDELL. “Estimation of sparse Jacobian matrices”. Dans : *SIAM Journal on Algebraic Discrete Methods* 4.3 (1983), p. 404–418.

[99] J. NOCEDAL. “Updating quasi-Newton matrices with limited storage”. Dans : *Mathematics of computation* 35.151 (1980), p. 773–782.

[100] Y. NOTAY. “An aggregation-based algebraic multigrid method”. Dans : *Electronic Transactions on Numerical Analysis* 37 (2010).

[101] J. M. ORTEGA et W. C. RHEINBOLDT. *Iterative solution of nonlinear equations in several variables*. T. 30. Siam, 2000.

[102] F. PACULL et S. AUBERT. “GMRES acceleration of restricted Schwarz iterations”. Dans : *Domain Decomposition Methods in Science and Engineering XXI*. To be published by Springer in the LNCSE collection. 2014.

[103] C. PANTELIDES. “The consistent initialization of differential-algebraic systems”. Dans : *J. Sci. Comput* 9 (1988), p. 213–231.

[104] E. PARDO-IGÚZQUIZA et M. CHICA-OLMO. “The Fourier Integral Method : an efficient spectral method for simulation of random fields”. Dans : *Mathematical Geology* 25 (1993), p. 177–216.

[105] M. L. PARKS, E. DE STURLER, G. MACKEY, D. D. JOHNSON et S. MAITI. “Recycling Krylov subspaces for sequences of linear systems”. Dans : *SIAM Journal on Scientific Computing* 28.5 (2006), p. 1651–1674.

[106] A. QUARTERONI et A. VALLI. *Domain decomposition methods for partial differential equations*. Numerical Mathematics and Scientific Computation. Oxford Science Publications. New York : The Clarendon Press Oxford University Press, 1999, p. xvi+360. ISBN : 0-19-850178-1.

[107] B. A. van de ROTTEN. “A limited memory Broyden method to solve high-dimensional systems of nonlinear equations”. Thèse de doct. Mathematisch Instituut, Universiteit Leiden, 2003.

[108] Y. SAAD et P. BROWN. “Convergence theory of nonlinear Newton-Krylov algorithms”. Dans : *SIAM J. Opt.* 4 (1994), p. 297–330.

[109] Y. SAAD et P. BROWN. “Hybrid Krylov methods for nonlinear systems of equations”. Dans : *SIAM J. Sci. Comput.* 11 (1990), p. 450–481.

[110] Y. SAAD. “A flexible inner-outer preconditioned GMRES algorithm”. Dans : *SIAM J. Sci. Comput.* 14.2 (1993), p. 461–469. ISSN : 1064-8275.

[111] Y. SAAD. *Iterative methods for sparse linear systems*. Siam, 2003.

[112] V. SIMONCINI et D. B. SZYLD. “Flexible inner-outer Krylov subspace methods”. Dans : *SIAM J. Numer. Anal.* 40.6 (2002), 2219–2239 (electronic) (2003). ISSN : 0036-1429.

[113] R. SINCOVEC, A. ERISMAN, E. YIP et M. EPTON. “Analysis of descriptor systems using numerical algorithms”. Dans : *Automatic Control, IEEE Transactions on* 26.1 (1981), p. 139–147.

[114] P. SPITERI, J.-C. MIELLOU et D. EL BAZ. “Parallel asynchronous Schwarz and multisplitting methods for a nonlinear diffusion problem”. Dans : *Numer. Algorithms* 33.1-4 (2003). International Conference on Numerical Algorithms, Vol. I (Marrakesh, 2001), p. 461–474. ISSN : 1017-1398.

[115] O. STEINBACH. *Stability estimates for hybrid coupled domain decomposition methods*. T. 1809. Lecture Notes in Mathematics. Berlin : Springer-Verlag, 2003, p. vi+120. ISBN : 3-540-00277-4. DOI : 10.1007/b80164. URL : http://dx.doi.org/10.1007/b80164.

[116] J. D. TEBBENS et M. TUMA. “Efficient preconditioning of sequences of nonsymmetric linear systems”. Dans : *SIAM Journal on Scientific Computing* 29.5 (2007), p. 1918–1941.

[117] W. F. TINNEY et J. W. WALKER. “Direct solutions of sparse network equations by optimally ordered triangular factorization”. Dans : *Proceedings of the IEEE* 55.11 (1967), p. 1801–1809.

[118] D. TROMEUR-DERVOUT. “Meshfree Adaptative Aitken-Schwarz Domain Decomposition with application to Darcy Flow”. Dans : *Parallel, Distributed and Grid Computing for Engineering*. Sous la dir. de TOPPING, BHV AND IVANYI, P. T. 21. Computational Science Engineering and Technology Series. Saxe-Coburg Publications, Stirlingshire, UK, 2009, 217–250. URL : http://dx.doi.org/10.4203/csets.21.11.

[119] J. UNGER, A. KRONER et W. MARQUARDT. “Structural analysis of differential-algebraic equation systems-theory and applications”. Dans : *Computers & chemical engineering* 19.8 (1995), p. 867–882.

[120] C. VANDEKERCKHOVE, I. KEVREKIDIS et D. ROOSE. “An efficient Newton-Krylov implementation of the constrained runs scheme for initializing on a slow manifold”. Dans : *Journal of Scientific Computing* 39.2 (2009), p. 167–188.

[121] C. VANDEKERCKHOVE, B. SONDAY, A. MAKEEV, D. ROOSE et I. G. KEVREKIDIS. “A common approach to the computation of coarse-scale steady states and to consistent initialization on a slow manifold”. Dans : *Computers & Chemical Engineering* 35.10 (2011), p. 1949–1958.

[122] H. A. VAN DER VORST. “Bi-CGSTAB : A fast and smoothly converging variant of Bi-CG for the solution of nonsymetric linear systems”. Dans : *SIAM J. Sci. Stat. Comput.* 13 (1992), p. 631–644.

[123] P. WILDERS et E. BRAKKEE. “Schwarz and Schur : an algebraical note on equivalence properties”. Dans : *SIAM Journal on Scientific Computing* 20.6 (1999), p. 2297–2303.

[124] YAO. “Reproduction of the mean, variance, and variogram model in spectral simulation”. Dans : *Mathematical Geology* 36 (2004), p. 487–506.

[125] M. ZIANI et F. GUYOMARC'H. “An autoadaptative limited memory Broyden's method to solve systems of nonlinear equations”. Dans : *Appl. Math. Comput.* 205.1 (2008), p. 202–211.

# Annexes

# Annexe A

# Accélération d'Aitken

## A.1 Réécriture de l'accélération dans le cas d'un partitionnement rouge-noir

L'accélération d'Aitken fait intervenir la matrice $(I-P)^{-1}$ que l'on peut construire en utilisant les égalités de Strassen suivantes :

$$\begin{bmatrix} A & B \\ C & D \end{bmatrix}^{-1} = \begin{bmatrix} A^{-1} + A^{-1}B(D-CA^{-1}B)^{-1}CA^{-1} & -A^{-1}B(D-CA^{-1}B)^{-1} \\ -(D-CA^{-1}B)^{-1}CA^{-1} & (D-CA^{-1}B)^{-1} \end{bmatrix} \tag{A.1}$$

et

$$\begin{bmatrix} A & B \\ C & D \end{bmatrix}^{-1} = \begin{bmatrix} \left(A-BD^{-1}C\right)^{-1} & -\left(A-BD^{-1}C\right)^{-1}BD^{-1} \\ -D^{-1}C\left(A-BD^{-1}C\right)^{-1} & D^{-1}+D^{-1}C\left(A-BD^{-1}C\right)^{-1}BD^{-1} \end{bmatrix}. \tag{A.2}$$

Lorsque $A$ et $D$ sont inversibles, on peut également écrire :

$$\begin{bmatrix} A & B \\ C & D \end{bmatrix}^{-1} = \begin{bmatrix} \left(A-BD^{-1}C\right)^{-1} & -\left(A-BD^{-1}C\right)^{-1}BD^{-1} \\ -(D-CA^{-1}B)^{-1}CA^{-1} & (D-CA^{-1}B)^{-1} \end{bmatrix} \tag{A.3}$$

cela donne ici :

$$(I-P)^{-1} = \begin{bmatrix} I & -P_0 \\ -P_1 & I \end{bmatrix}^{-1} = \begin{bmatrix} S_0^{-1} & S_0^{-1}P_0 \\ S_1^{-1}P_1 & S_1^{-1} \end{bmatrix} \tag{A.4}$$

avec $S_0 = (I - P_0P_1)$ et $S_1 = (I - P_1P_0)$. La formule de l'accélération d'Aitken peut donc s'écrire :

$$\begin{bmatrix} v_0^\infty \\ v_1^\infty \end{bmatrix} = (I-P)^{-1}\left(u^{n+1} - Pu^n\right) \tag{A.5}$$

$$= \begin{bmatrix} S_0^{-1} & S_0^{-1}P_0 \\ S_1^{-1}P_1 & S_1^{-1} \end{bmatrix} \times \begin{bmatrix} v_0^{n+1} - P_0v_1^n \\ v_1^{n+1} - P_1v_0^n \end{bmatrix}. \tag{A.6}$$

En développant, on obtient :

$$\begin{bmatrix} v_0^\infty \\ v_1^\infty \end{bmatrix} = \begin{bmatrix} S_0^{-1}\left(v_0^{n+1} - P_0 v_1^n + P_0\left(v_1^{n+1} - P_1 v_0^n\right)\right) \\ S_1^{-1}\left(v_1^{n+1} - P_1 v_0^n + P_1\left(v_0^{n+1} - P_0 v_1^n\right)\right) \end{bmatrix} \tag{A.7}$$

$$= \begin{bmatrix} S_0^{-1}\left(v_0^{n+1} - P_0 v_1^n + P_0 v_1^{n+1} - P_0 P_1 v_0^n\right) \\ S_1^{-1}\left(v_1^{n+1} - P_1 v_0^n + P_1 v_0^{n+1} - P_1 P_0 v_1^n\right) \end{bmatrix} \tag{A.8}$$

$$= \begin{bmatrix} S_0^{-1}\left(v_0^{n} - P_0 v_1^{n-1} + P_0 v_1^{n+1} - P_0 P_1 v_0^n\right) \\ S_1^{-1}\left(v_1^{n} - P_1 v_0^{n-1} + P_1 v_0^{n+1} - P_1 P_0 v_1^n\right) \end{bmatrix} \tag{A.9}$$

$$= \begin{bmatrix} S_0^{-1}\left(P_0\left(v_1^{n+1} - v_1^{n-1}\right) + \left(I - P_0 P_1\right) v_0^n\right) \\ S_1^{-1}\left(P_1\left(v_0^{n+1} - v_0^{n-1}\right) + \left(I - P_1 P_0\right) v_1^n\right) \end{bmatrix} \tag{A.10}$$

$$= \begin{bmatrix} S_0^{-1} P_0\left(v_1^{n+1} - v_1^{n-1}\right) \\ S_1^{-1} P_1\left(v_0^{n+1} - v_0^{n-1}\right) \end{bmatrix} + \begin{bmatrix} v_0^n \\ v_1^n \end{bmatrix} \tag{A.11}$$

L'avantage de la formule (A.11) est que l'accélération peut être calculée indépendamment sur les deux sous-domaines.

# Annexe B

# Implémentation de la méthode d'Aitken-Schwarz en PETSc

## B.1 Présentation générale

Le code `AS3D` implémente la méthode d'Aitken-Schwarz et repose sur la bibliothèque PETSc. La résolution s'effectue sur un maillage (Distributed Mesh ou `DM`) régulier par une discrétisation avec un schéma à 7 points. Il a été développé initialement pour la plateforme HYDROLAB, mais une vesion alpha du code est disponible sur demande à l'adresse `http://cdcsp-recherche.univ-lyon1.fr`. Son objectif est de permettre la résolution de très grands problèmes linéaires issus de la discrétisation d'un problème 3D sur une grille régulière par un schéma à 7 points.

## B.2 Utilisation

Pour utiliser cette bibliothèque, il faut disposer de PETSc. La compilation se fait de la même manière que pour un code PETSc classique. Les deux exemples d'utilisation concernent l'utilisation de la méthode d'Aitken-Schwarz en tant que solveur. Celle-ci peut toutefois être utilisée en tant que préconditionneur. Dans le premier exemple, l'utilisateur fournit une matrice globale et un second membre. Dans le second exemple, l'utilisateur fournira également une matrice locale. On fournira ensuite l'ensemble des paramètres pouvant être passés à l'exécution du programme.

### B.2.1 Exemple 1 : définir une matrice et un second membre

La matrice globale, le second membre et la solution doivent être stockés dans `worldA`, `worldB` et `worldX` de la structure `AitkenSchwarz`. Voici un exemple d'utilisation :

```
1  int main(int argc, char **args)
2  {
3    AitkenSchwarzCtx info;
4    MPI_Init(&argc,&args);
5    PetscInitialize(&argc,&args,(char *)0,help);
6    AitkenSchwarzInitialize(&argc,&args,&info);
7    //user function to compute the matrix and the RHS:
8    ComputeMatrixAndRHS(&info,info.worldA,info.worldB);
9    CreateMacroMatrices(&info,info.worldA);
10   AitkenSchwarzSolve(&info);
11   AitkenSchwarzDestroy(&info);
12   PetscFinalize();
13   MPI_Finalize();
14   return 0;
15 }
```

Toutes les fonctions de cet exemple doivent être appelées dans cet ordre :

- La fonction `MPI_Init` doit être appelée puisque le communicateur `PETSC_COMM_WORLD` est un sous-communicateur de `MPI_COMM_WORLD`.
- La fonction `AitkenSchwarzInitialize` crée la structure `info` à partir des options passées lors de l'exécution.
- L'appel à la fonction `CreateMacroMatrices` crée les matrices locales aux macro-domaines. La création des matrices locales peut représenter un coût de calcul important, et beaucoup de communications car il faut redistribuer la matrice globale.
- La fonction `AitkenSchwarzSolve` effectue la résolution.
- La fonction `AitkenSchwarzDestroy` désalloue la mémoire utilisée.

Il est également important de signaler que tous les objets PETSc créés sur le communicateur `MPI_COMM_WORLD` doivent être détruits en utilisant la fonction PETSc appropriée sinon une erreur se produira lors de l'appel à `PetscFinalize`.

Les vecteurs `worldX` et `macroX` stockent leurs valeurs dans le même tableau, ce qui n'est pas vraiment prévu par PETSC. Avant d'utiliser une fonction PETSc sur un vecteur stocké dans une structure de type `AitkenSchwarz`, il est fortement conseillé d'appeler la fonction `UpdateCache` sur ce vecteur.

### B.2.2 Exemple 2 : définir l'opérateur local au sous-domaine

Dans l'exemple précédent, la matrice globale était redistribuée. Cette redistribution est handicapante lorsque la matrice est de grande taille. C'est pourquoi l'utilisateur peut également fournir sa propre fonction pour calculer les opérateurs de chaque sous-domaine.

```
1  PetscErrorCode ComputeMatrix(AitkenSchwarzCtx *ctx, Mat A, Vec
       rhs);
2  PetscErrorCode ComputeMatrixLoc(void * user, Mat *Aloc);
3
4  int main(int argc, char **args)
5  {
6      AitkenSchwarzCtx info;
7      User myCtx; //user-defined Structure
8      MPI_Init(&argc,&args);
9      PetscInitialize(&argc,&args,(char *)0,help);
10     MPI_Wtime();
11     AitkenSchwarzInitialize(&argc,&args,&info);
12     ComputeMatrix(&info,info.worldA,info.worldB);
13
14     //fill the user-defined context
15     myCtx.as = &info;
16
17     // Provide the user context
18     info.userCtx = &myCtx;
19     info.CreateMacroMatricesFunc  = ComputeMatrixLoc;
20
21     CreateMacroMatrices(&info,A);
22     AitkenSchwarzSolve(&info);
23     AitkenSchwarzDestroy(&info);
24     PetscFinalize();
25     MPI_Finalize();
26     return 0;
27 }
```

L'utilisateur peut définir une fonction (ici `ComputeMatrixLoc`) qui sera appelée lors d'appel à `CreateMacroMatrices`. Cette fonction prendra en paramètre une structure définie par l'utilisateur, contenant toutes les données nécessaires à la création des matrices locales. La structure de cet exemple `User` contient la structure `info`. On pourra ainsi accéder aux données relatives au sous-domaines depuis la fonction `CreateMacroMatrices`.

### B.2.3 Paramètres

Voici la liste des paramètres que l'on peut passer à l'exécution du programme.

**TABLE *B.1 –*** *Liste des paramètres*

| Noms | Valeurs | Défaut | Commentaires |
|---|---|---|---|
| `NX` | $> 8$ | 128 | Nombre total de points en $X$ |
| `NY` | $> 8$ | 128 | Nombre total de points en $Y$ |
| `NZ` | $> 8$ | 128 | Nombre total de points en $Z$ |
| `overlap` | $\geq 1$ | 2 | Doit être $<$ au nombre de points en $Z$ par processeur |
| `macros` | $> 1$ | 2 | Nb total de macro-domaines |
| `delay` | $\geq 0$ | 0 | Nb d'itérations de Schwarz après accélération |
| `nbTraces` | $\geq 0$ | 10 | Nb de traces nécessaires à une accélération |
| `nbIterMax` | $\geq 0$ | 100 | Nb maximal d'itérations de Schwarz |
| `validAccel` | $[0, 1]$ | 100 | Accélération effectuée si diminution de $\|b - Ax\|$ |
| `tolAbs` | $\mathbb{R}^+$ | $10^{-10}$ | Tolérance absolue |
| `tolRel` | $\mathbb{R}^+$ | $10^{-10}$ | Tolérance relative |
| `tolSVDInvRel` | $\mathbb{R}^+$ | $10^{-10}$ | Troncature des v.s. lors des pseudo-inversions |
| `tolSVDInvAbs` | $\mathbb{R}^+$ | $10^{-10}$ | Troncature des v.s. lors des pseudo-inversions |

v.s. : valeurs singulières

## B.3 Détails de l'implémentation

### B.3.1 Topologie MPI

Le schéma donné en figure B.1 présente l'ensemble des communications MPI et les communicateurs sur lesquelles elles reposent.

- Les communications les plus intenses sont effectuées au niveau de la résolution des sous-problèmes par une méthode de Krylov. La quantité et le type des communications peut varier en fonction de la méthode de Krylov choisie et du préconditionneur. En général, il y aura essentiellement des produits matrice-vecteur (communications entre les processeurs voisins) et des produits scalaires (réductions). Certains préconditionneurs, tels que les préconditionneurs multigrilles peuvent demander des communications encore plus intensives.

- L'échange de Schwarz se fait processeur à processeur et ne constitue pas un goulot d'étranglement : il y a moins de données à échanger que lors d'un produit matrice-vecteur.

- Les traces sont rassemblées sur le processeur $0$ de chaque interface. Celui-ci calculera la décomposition en valeurs singulières de ces traces et projettera les traces dans la base $U$ obtenue. Il a été choisi de rassembler l'interface sur un processeur afin de calculer

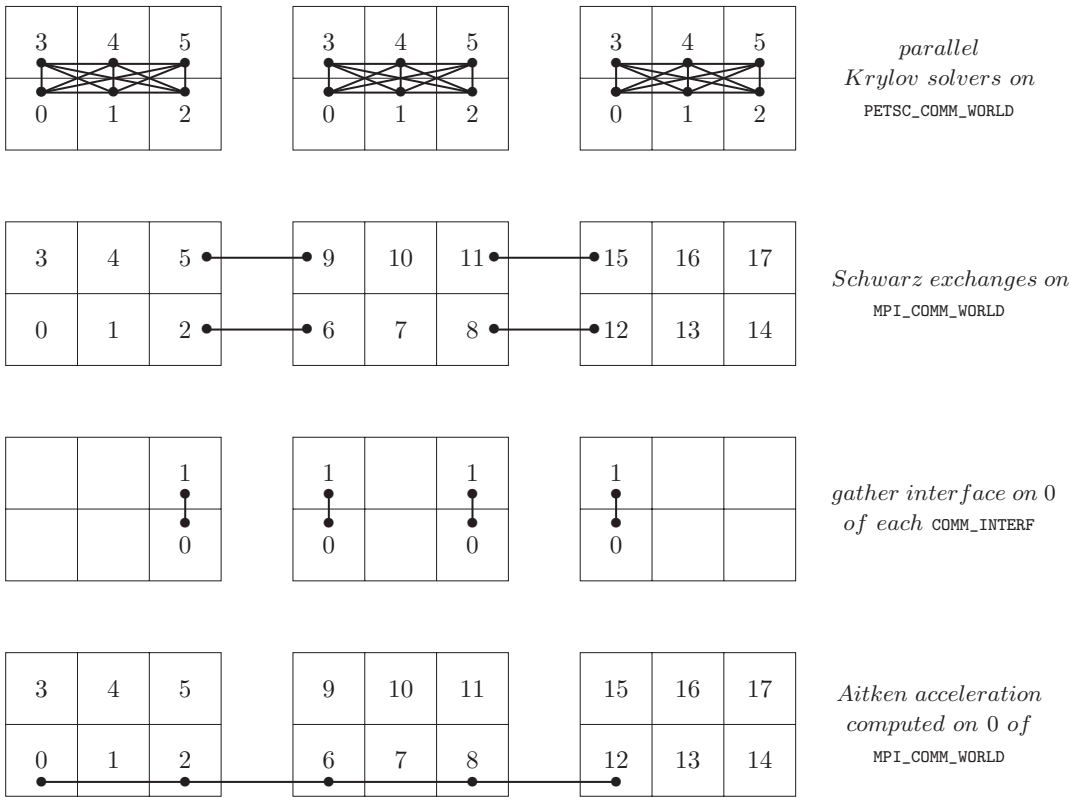


**FIGURE *B.1*** – *Communications et communicateurs MPI sur un exemple 2D. Les traits épais relient les processeurs qui communiquent.*

la décomposition en valeurs singulières de manière séquentielle plutôt que d'utiliser un algorithme de décomposition SVD parallèle moins robuste.

- Les traces dans l'espace de la SVD sont rassemblées sur le processeur $0$ de `MPI_COMM_WORLD` qui calculera l'accélération d'Aitken. Les messages sont de la taille de $(q)^2$ nombres en double précision lorsque l'accélération s'effectue sur $q$ traces.

- La solution accélérée est ensuite renvoyée aux processeurs $0$ des interfaces, qui projetteront la solution accélérée dans l'espace physique avant de la redistribuer aux autres processeurs de l'interface.

### B.3.2 Structures de données

Toutes les informations nécessaires à la méthode d'Aitken sont stockées dans la structure `AitkenSchwarz` qui comprend, entre autre :

- Des objets PETSc tels que le maillage, la matrice globale, les matrices locales, le second membre `B` et la solution `X`. Par exemple, si `ctx` est un objet de type `AitkenSchwarz`, alors on utilisera `ctx.worldA` pour accéder à la matrice globale, ou `ctx->macroA` pour accéder à la matrice locale au macro-domaine.
- Une structure `numParams` de type `NumericalParameters` qui contient les paramètres numériques tels que le nombre de traces entre deux accélérations et les diverses tolérances.
- Une structure `topo` de type `Topology` qui contient toutes les informations relatives à la géométrie du problème. On y trouvera par exemple le nombre de points dans chaque direction pour le domaine global (`worldNX`), pour le macro-domaine `macroNY`, ou pour le processeur `procNZ`. Cette structure contient également les informations concernant les topologies `MPI` des communicateurs `MPI_COMM_WORLD` et `PETSC_COMM_WORLD`.
- Une structure `buf` de type `Buffers` qui contient les espaces mémoires nécessaires à l'accélération d'Aitken sur le processeur 0. On y retrouvera donc les traces du processus de Schwarz, et l'espace de travail de la bibliothèque `LAPACK` pour le calcul des décompositions en valeurs singulières.

# Annexe C

# Initialisation et résolution des EDA

## C.1 Méthodes de résolution des problèmes à valeur initiale

Les tests numériques présents dans ce manuscrit concerneront uniquement les méthodes de type *Backward Differenciation Formulas* (BDF), dont la plus simple est la méthode d'Euler implicite. D'autres familles importantes de méthodes existent, en particulier les méthodes de Runger-Kutta, mais ne sont pas présentées ici.

### C.1.1 Méthodes à pas multiples

Les méthodes BDF sont les plus communément utilisées pour la résolution des EDO et EDA $F(t, y, y')$. Une méthode à $k$ pas consiste à remplacer la variable $y'$ par la dérivée du polynôme qui interpole la solution aux $k+1$ pas de temps précédents. Si on considère un pas de temps $h = t_n - t_{n-1}$, cela revient à trouver la solution de l'équation (C.1) où les $\alpha_i$ sont les coefficients de la méthode considérée.

$$F\left(t_n, y_n, \frac{1}{h}\sum_{i=0}^{k} \alpha_i y_{n-i}\right) = 0 \tag{C.1}$$

À chaque pas de temps l'équation non linéaire (C.1) est résolue par une méthode de Newton, en utilisant la jacobienne de $F$ par rapport à $y_n$, notée $F_{y_n}$ et définie par (C.2)

$$F_{y_n} = \frac{\alpha_0}{h}\frac{\partial F}{\partial y'} + \frac{\partial F}{\partial y}. \tag{C.2}$$

Notons que le conditionnement de la matrice $F_{y_n}$ impliquée dans le système linéaire à inverser est en $O(h^{-\nu})$ pour les EDA linéaires à coefficients constants.

La convergence des méthodes BDF a été largement étudiée [58], les principaux théorèmes sont rappelés dans la suite.

**Théorème 8.** *Une méthode BDF à $k$ pas $(k < 7)$ appliquée à une EDA linéaire à coefficients constants d'indice $\nu$ est convergente d'ordre $O(h^k)$ après $(\nu - 1)k + 1$ pas.*

Toujours pour des EDA linéaires à coefficients constants, et dans le cas où le pas est variable, on peut montrer [59] que l'ordre de convergence est $O(h_{max}^q)$ avec $q = min(k, k -$

$\nu + 2)$ sous réserve que le ratio entre des pas successifs soit borné. On remarquera que dans le cas d'une EDA linéaire d'indice 3, la convergence n'est pas garantie puisque pour $k = 1$ on obtient une erreur en $O(1)$. Cela peut poser un problème dans la mesure où l'utilisation des méthodes du premier ordre est fréquente pour le calcul des premiers pas de temps.

Les méthodes à pas multiples ainsi que les méthodes de type Runge-Kutta peuvent s'avérer instables pour des EDA d'indice 2 et supérieur [64].

Le théorème suivant concernant la convergence des méthodes BDF sont démontrés dans [19].

**Théorème 9.** *Soit une EDA d'indice 1 sur un intervalle de temps $I$. La solution numérique de cette EDA sur $I$ par une méthode BDF à $k$ pas, avec un pas fixe $h$ est convergente pour $k < 7$ en $O(h^k)$ si les valeurs initiales sont données en $O(h^k)$ et que la solution du système non linéaire est obtenue avec une prévision en $O(h^{k+1})$.*

Dans le cas des méthodes BDF à pas de temps variables, on peut montrer la convergence sous certaines conditions dans la variation des pas de temps [72, 35]. Des résultats similaires s'obtiennent pour les EDA semi-explicites d'indice 2, à la différence que la condition initiale doit être estimée avec une erreur en $O(h^{k+1})$ pour avoir une convergence globale en $O(h^k)$. Dans le cas où la solution initiale est donnée avec une erreur en $O(h^k)$, l'erreur globale en $O(h^k)$ est obtenue après $k + 1$ pas. Sous certaines conditions (pas de temps constant, surrésolution des équations algébriques), on peut obtenir la convergence d'une méthode BDF pour les EDA d'indice 3 sous forme de Hessenberg.

Pour les EDA implicites d'indice 2 et supérieur, l'application d'une méthode de type BDF ne garantit pas la convergence de la solution. Dans ce cas un traitement préalable des équations peut permettre d'obtenir une solution, par exemple l'indice peut être réduit par différenciations successives. Dans ce cas, les contraintes ne sont plus imposées strictement, et on peut observer un glissement des contraintes au cours des itérations en temps. Lorsque les équations analytiques le permettent, les contraintes peuvent être imposées par l'ajout de multiplicateurs de Lagrange. Ainsi l'approche GGL (Gear-Gupta-Leimkuhler) [63] permet une stabilisation des contraintes dans le cas des systèmes mécaniques multi-corps rigides.

### C.1.2 Recherche d'une solution initiale

Les méthodes les plus simples d'initialisation sont basées sur le fait qu'une solution stationnaire est consistante. On peut donc faire une première intégration en temps, sans se soucier des résidus intermédiaires, jusqu'à obtenir une solution stationnaire. En pratique, il est possible de ne faire cette intégration que sur un modèle réduit grossier, ou tout simplement d'ajouter de la diffusion. Il est important de noter que cette méthode est empirique, dans la mesure où la convergence est prouvée en faisant une série d'hypothèses difficilement vérifiables en pratique.

Dans ce qui suit, nous allons voir trois types de méthodes : le premier type consiste à effectuer plusieurs fois le (ou les) premier pas de temps, en diminuant $h$. Le second type de méthode consiste à résoudre le système d'EDA simultanément avec les contraintes algébriques cachées. Les méthodes les plus récentes sont basées sur l'attractivité de la variété lente que constitue l'ensemble des contraintes algébriques.

### C.1.2.1 Méthodes basées sur l'intégration en temps

Sans aller jusqu'à la solution stationnaire, il est possible de faire quelques pas de temps, sans contrôle d'erreur, pour atteindre une solution consistante. Cette idée remonte probablement à [113] qui suggère qu'une solution consistante peut être atteinte après $\nu$ pas de temps, où $\nu$ est l'indice de l'EDA. Le problème est qu'il devient difficile d'imposer des valeurs initiales, puisque ces valeurs peuvent être changées par l'intégration.

Une variation de cette technique est utilisée dans DASSL, où l'algorithme s'arrête uniquement si les valeurs imposées par l'utilisateur à $t_0$ sont toujours respectées à $t_1$.

**Algorithme 19** Pas d'Euler pour les EDO

**Require:** une tolérance $\epsilon$, $h_{max}$

1: $h_0 = min\{h_{max}, 1/(2||\dot{x}_0||)\}$
2: **repeat**
3: Prédiction : $x_1^{(0)} = x_0 + h_0\dot{x}_0$, $\dot{x}_1^{(0)} = \dot{x}_0$
4: Correction : Damped-Newton donne $\Delta x$ :
$x_1^{(k+1)} = x_1^k - \delta\Delta x$, $\dot{x}_1^{(k+1)} = \dot{x}_1^{(k+1)} - \frac{1}{h_0}\delta\Delta x$
5: $h_0 = h_0/10$
6: **until** $||x_0 - x_1|| < \epsilon$

**Adaptation aux EDA** Bien que l'algorithme précédent ait été utilisé pour initialiser des EDA, une adaptation proposée par KRONER, MARQUARDT et GILLES [83] est plus couramment utilisée. Elle consiste à adapter la taille du pas de Newton $\delta$, et le contrôle de l'erreur.

**Algorithme 20** Pas d'Euler pour les EDA

**Require:** des tolérances $\epsilon$ et $\xi$, $h_{max}$

1: $h_0 = min\{h_{max}, 1/(2||\dot{x}_0||)\}$
2: **repeat**
3: Prédiction : $x_1^{(0)} = x_0 + h_0\dot{x}_0$, $\dot{x}_1^{(0)} = \dot{x}_0$
4: Correction : Damped-Newton donne $\Delta x$ :
$x_1^{(k+1)} = x_1^k - \delta^k\Delta x$, $\dot{x}_1^{(k+1)} = \dot{x}_1^{(k+1)} - \frac{1}{h_0}\delta^k\Delta x$
5: **until** $||x_1^{(*)} - x_1^{(0)}|| < \epsilon$ et $||F(x_1^{(*)}, \dot{x}_1^{(*)}, t_1)||_2 < \xi$

Plusieurs méthodes ont été proposées pour revenir à $t_0$ :

- On peut effectuer l'intégration arrière par un schéma d'Euler explicite pour trouver $x_0$ étant donné $x_1$. Cette méthode a été initialement proposée dans [113].
- Il est également possible de trouver $x_0$ et $\dot{x_0}$ par extrapolation arrière [83] des valeurs obtenues au cours des pas de temps successifs.
- On peut également effectuer un pas projectif [62] : après $k$ itérations on calcule $x_0^p = k\ x_{k-1} - (k-1)x_k$.

Notons également que des théorèmes sur la convergence sont disponibles pour certaines EDA uniquement.

#### C.1.2.2 Initialisation à l'aide des variétés lentes

L'idée commune aux méthodes présentées ici est de voir la solution stationnaire comme un point fixe : on peut considérer que l'intégration sur $m$ pas de temps à partir de la solution $x^k$ est une fonction de $\phi(x^k) = x^{k+m}$. Dans ce cas, la solution stationnaire, qui est donc la solution consistante, vérifie $\phi(x) = x$, et on peut donc appliquer une méthode de Newton-Krylov directement sur cette fonction.

Les méthodes d'initialisation qui utilisent l'attractivité des variétés lentes reposent donc sur le même principe que les méthodes présentées dans la sous-section précédente : il s'agit de faire quelques pas d'intégration en temps et de revenir à $t_0$. Il y a cependant deux différences majeures : la façon d'imposer certaines valeurs à $t_0$ est différente, et le pas de temps n'est pas diminué par les itérations.

En réalité, les méthodes présentées ici sont basées sur la réduction de modèle : on considère que l'on peut diviser les variables du système en deux catégories : $u$ sont les variables observables, qui sont les variables d'intérêt permettant de définir le modèle grossier. On considérera en général que ce sont également les variables dont on connaît la valeur à $t_0$, appelée $u_0$. Les variables $v$ contiendront le reste des variables du système complet.

On dira qu'une solution initiale est consistante si elle satisfait :

$$\frac{d^{m+1}v}{d^{m+1}t} = 0 \tag{C.3}$$

où $m$ est l'ordre de précision de la solution. Une démonstration est donnée dans [61] dont voici les principales étapes :

- On réécrit d'abord le problème sous la forme :

$$\begin{cases} \dot{x} &= f(x,y) \\ \epsilon\dot{y} &= g(x,y) \end{cases} \tag{C.4}$$

- Puis on écrit le développement en perturbations singulières

$$\left\{\begin{array}{rcl} x(t,\epsilon) & = & \displaystyle\sum_{n=0}^{\infty}\epsilon^n X_n(t) + \epsilon\sum_{n=0}^{\infty}\epsilon^n \xi_n(t/\epsilon) \\ y(t,\epsilon) & = & \displaystyle\sum_{n=0}^{\infty}\epsilon^n Y_n(t) + \sum_{n=0}^{\infty}\epsilon^n \eta_n(t/\epsilon) \end{array}\right. \tag{C.5}$$

- On remplace ensuite $y$ dans

$$\frac{d^{m+1}v}{dt^{m+1}} = v_y\frac{d^{m+1}y}{dt^{m+1}} + \cdots = \epsilon^{-(m+1)}v_y\frac{d^{m+1}\eta_0}{d\tau^{m+1}} + \cdots + O(\epsilon^{-m}). \tag{C.6}$$

L'idée sous-jacente est que la dérivation en temps amplifie plus les composantes rapides que les composantes lentes.

#### C.1.2.3 La méthode *Constrained Runs*

En introduction, par soucis de simplicité, nous avons considéré que $u$ et $v$ étaient des partitions des inconnues du système. La méthode est en fait plus générale. Considérons que $x$ sont les variables différentielles, et $y$ les variables algébriques et que nous cherchons à résoudre

$$\begin{array}{rcl} \dot{x} & = & f(x,y) \\ 0 & = & g(x,y) \end{array}. \tag{C.7}$$

Dans ce cas, la méthode peut s'appliquer quelles que soient les transformations $u$ et $v$ des inconnues :

$$\begin{array}{l} u{=}u(x,y) \\ v{=}v(x,y). \end{array} \tag{C.8}$$

Par la suite, on se limitera au cas où $(u,v)$ est un réordonnancement des inconnues.

Initialement, GEAR et KEVREKIDIS ont proposé de calculer le nouveau $v_0$ par une extrapolation arrière des $v_j$ obtenus. Cette méthode est issue de [62] où l'extrapolation permettait de calculer des valeurs antérieures, en utilisant uniquement un schéma d'intégration avant. L'algorithme de la méthode est donné en algorithme 21, où $\Delta^{m+1}(v(t))$ est la formule d'approximation des dérivés par différences avants.

$$\begin{array}{rcl} \Delta^{m+1}(v(t)) & = & \Delta^m v(t+h) - \Delta^m v(t) \\ & \underbrace{=}_{\Delta^0 v(t)=v(t)} & h^{m+1}\dfrac{d^{m+1}v}{dt^{m+1}} + O(h^{m+2}) \end{array} \tag{C.9}$$

Dans cet algorithme, l'étape $v_0 = v_0 + \delta$ doit être vue comme la création d'une interpolation linéaire dans le plan $(t-v)$ de $v_1, \ldots v_{m+1}$. Le nouveau $v_0$ étant le point de cette courbe passant par $t_0$, $v_0$ est donc obtenue par extrapolation. Un exemple permet de mettre en évidence le lien entre la formule (C.9) et l'interpolation.

**Algorithme 21** Constrained Runs avec interpolation

**Require:** a tolerance $\epsilon$ and a steplength $h$
**Require:** known variables $u_0$, and an initial guess for the unknowns $v_0$
1: **repeat**
2: perform $m+1$ time steps $(u_0, v_0)$ : $(u_j, v_j)$ with $j = 1 \dots m+1$
3: $\delta = (-1)^m \Delta^{m+1} v_0$
4: $v_0 \leftarrow v_0 + \delta$
5: **until** $||\delta|| < \epsilon$

**Exemple 3.** *Prenons $m = 1$, dans ce cas :*

$$\begin{aligned} \Delta^2 v_0 &= \Delta^1 v_1 - \Delta^1 v_0 \\ &= (\Delta^0 v_2 - \Delta^0 v_1) - (\Delta^0 v_1 - \Delta^0 v_0) \\ &= (v_2 - v_1) - (v_1 - v_0) \\ &= v_2 - 2v_1 + v_0. \end{aligned} \tag{C.10}$$

*Finalement, $\delta = (-1)^1 \Delta^2 = -v_2 + 2v_1 - v_0$ et donc $v_0 = v_0 + \delta = 2v_1 - v_2$.*

La convergence de cet algorithme repose sur le fait que l'intégration en temps amortit les composantes rapides de la solution, et que cet amortissement est plus important que l'effet du retour à $t_0$ par l'extrapolation.

#### C.1.2.4 La méthode *Constrained Runs Functional*

On considère ici qu'une itération de l'algorithme 21 peut s'écrire comme l'évaluation d'une fonction $\mathcal{C}$ telle que $v_{k+1} = \mathcal{C}(u_0, v_k)$. Les deux algorithmes s'arrêtent lorsque $\delta < \epsilon$. La solution ayant convergé, $v_0^*$ est donc un point fixe de la fonction $\mathcal{C}$. VANDEKERCKHOVE, KEVREKIDIS et ROOSE ont donc proposé dans [120] de résoudre ce problème par une méthode de Newton-Krylov. On cherche donc à résoudre le problème de point fixe : $g(u_0, v) = v - \mathcal{C}(u_0, v) = 0$ par une méthode de Newton : $v_{k+1} = v_k + \delta_k$ avec $\delta_k$ solution de l'équation (C.11)

$$\left( I - \frac{\partial \mathcal{C}}{\partial v}(u_0, v_k) \right) \delta_k = -g(u_0, v_k). \tag{C.11}$$

En pratique, il n'est pas possible de calculer la jacobienne de la fonction $\mathcal{C}$ car celle-ci effectue l'intégration sur $m+1$ pas de temps. Dans l'esprit des méthodes de type *Jacobian-Free* [82], on se contentera d'estimer le produit matrice-vecteur par l'équation (C.12) :

$$\left( I - \frac{\partial \mathcal{C}}{\partial v}(u_0, v_k) \right) \delta_k \approx \delta_k - \frac{\mathcal{C}(u_0, v_k + \epsilon \delta_k) - \mathcal{C}(u_0, v_k)}{\epsilon}. \tag{C.12}$$

L'approximation du produit matrice-vecteur permet d'appliquer une méthode de Krylov. Dans [120] la méthode choisie est GMRES [110].

#### C.1.2.5 La méthode *InitMan*

Cette méthode, proposée par VANDEKERCKHOVE et al. dans [121], peut être vue comme une variante de la méthode CRF. Notons $\Phi(u, v)$ l'opération qui consiste à effectuer les quelques pas d'intégrations en temps présents dans les itérations de CRF, à partir de la solution initiale $(u, v)$. La méthode CRF avec restriction se résume donc à chercher le point fixe de $\mathcal{C}(u_0, v) = \phi(u_0, v)|_v$, c'est-à-dire à trouver $v$ tel que $v - \Phi(u_0, v)|_v = 0$. La méthode InitMan consiste à chercher $u$ tel que $u_0 - \Phi(u, v_0)|_u = 0$, la solution $v$ recherchée sera alors $v = \Phi(u, v_0)|_v$. Notons que ces méthodes CRF et InitMan ont été introduites dans le contexte des intégrateurs grossiers : $u$ représente les variables retenues dans le modèle grossier, et $v$ regroupe les variables supposées être soumises à $u$. Dans ce contexte, l'opérateur $\Phi$ peut très bien être l'intégrateur grossier, qui ne dépend que de $u$ et estime $\phi(u)|_v$ après l'intégration par une opération appelée *lifting* dans [121, 60].

#### C.1.2.6 Méthodes directes

**Cas particulier des EDA semi-explicites d'indice 1** Dans le cas particulier des EDA semi-explicites d'indice 1, une solution initiale consistante peut être trouvée par diverses manipulations analytiques. C'est notamment ce qui est implémenté dans la bibliothèque DASKP [21], et initialement proposé dans [85].

Soit une EDA semi-explicite d'indice 1 $G(t, y, \dot{y}) = 0$ mise sous la forme

$$y = \begin{bmatrix} u \\ v \end{bmatrix}, \begin{cases} f(t, u, v, \dot{u}) &= 0 \\ g(t, u, v) &= 0 \end{cases}, \ f_{\dot{u}} \text{ non singulière.} \tag{C.13}$$

Le problème d'initialisation se résume souvent à chercher $v_0$ et $\dot{u}_0$, $u_0$ étant donné.

L'idée principale est de résoudre le système non linéaire :

$$x = \begin{bmatrix} \dot{u}_0 \\ v_0 \end{bmatrix}, \ F(x) = \begin{bmatrix} f(t_0, u_0, v_0, \dot{u}_0) \\ g(t_0, u_0, v_0) \end{bmatrix} = 0 \tag{C.14}$$

directement par une méthode de Newton. Ce système est appelé *système étendu* dans la mesure où l'on va chercher à résoudre explicitement les contraintes.

La résolution de ce système non linéaire demande de calculer la matrice jacobienne

$$F'(x) = \begin{bmatrix} f_{\dot{u}} & f_v \\ 0 & g_v \end{bmatrix}. \tag{C.15}$$

Cette matrice est estimée numériquement dans DASPK en utilisant

$$J = \frac{1}{h}\frac{\partial G}{\partial \dot{y}} + \frac{\partial G}{\partial y} \tag{C.16}$$

ce qui donne :

$$\tilde{J} = J \begin{bmatrix} hI & 0 \\ 0 & I \end{bmatrix} = \begin{bmatrix} f_{\dot{u}} + hf_u & f_v \\ hg_u & g_v \end{bmatrix} = F'(x) + h \begin{bmatrix} f_u & 0 \\ hg_u & 0 \end{bmatrix}. \tag{C.17}$$

Ainsi, pour $h$ suffisamment petit on effectuera les itérations de Newton de la manière suivante : $\Delta x = -\tilde{J}^{-1}F(x)$.

Cette approche peut être généralisée au cas où les variables peuvent s'écrire sous la forme $Py = [u, v]^T$, avec $P$ une matrice de permutation, ce qui permet toujours d'imposer $u_0$ directement, ou bien généralement une matrice inversible.

Les avantages de cette méthode sont :

- Il s'agit d'une méthode algébrique : il n'y a ni calcul symbolique, ni de différentiation automatique.
- Sa mise en place est relativement simple, et elle est déjà implémentée dans le logiciel DASPK.
- La convergence théorique est au moins linéaire.

Les inconvénients :

- Cette méthode s'applique uniquement à des classes restreintes d'EDA (les EDA semi-explicites d'indice 1 ou Hessenberg d'indice 2).
- En pratique, une méthode de Newton très robuste est nécessaire. Une simple globalisation peut ne pas suffire dans le cas où l'approximation initiale de la solution est mauvaise.

**Méthode des équations de consistance** Lorsque l'EDA a un indice supérieur à 1, certaines contraintes sont cachées, ou implicites. La méthode des équations de consistance consiste à mettre à jour ces contraintes cachées par la différentiation successive des équations. Le système d'EDA est ainsi augmenté de ses contraintes cachées, puis est résolu directement. En ce sens, cette méthode peut être vue comme une généralisation de la méthode 'directe' aux EDA d'indice supérieur, à ceci près qu'aucune astuce de calcul n'est utilisée pour calculer la matrice jacobienne.

La différentiation successive d'un système d'EDA d'indice $\nu$ donne

$$G\left(t_0, y_0, y_0', \cdots, y_0^{(\nu+1)}\right) = \begin{bmatrix} F\left(t_0, y_0, y_0'\right) \\ \dfrac{dF}{dt}\left(t_0, y_0, y_0', y_0''\right) \\ \vdots \\ \dfrac{d^\nu F}{dt^\nu}\left(t_0, y_0, y_0', y_0'', .., y_0^{(\nu+1)}\right) \end{bmatrix} = 0 \qquad \text{(C.18)}$$

où les contraintes sont explicitées.

On peut maintenant chercher à résoudre $G(t_0, y_0, y_0', \cdots, y_0^{(\nu+1)}) = 0$ par une méthode de Newton. Nous aurons l'unicité de $y_0$ et $y_0'$ mais pas des dérivées d'ordre supérieur.

La matrice jacobienne associée à la fonction $G$ sera de rang plein uniquement si seules les dérivations nécessaires ont été effectuées. Dans l'équation (C.18), nous avons considéré uniquement le cas où toutes les équations étaient dérivées un même nombre de fois. Il faut pouvoir dériver uniquement les équations de $F$ nécessaires pour obtenir une matrice jacobienne inversible. Intuitivement, il faut dériver chaque variable un nombre de fois correspondant à son indice.

Plusieurs méthodes ont été proposées pour dériver uniquement les équations nécessaires. Ainsi, en 1989, dans [103], PANTELIDES propose une méthode basée sur la théorie des graphes, demandant de connaître les équations analytiques. Par la suite, une méthode numérique sera proposée dans [119], elle est basée sur l'analyse de l'emplacement des éléments non nuls de $\partial F/\partial y$ et $\partial F/\partial y'$.

La construction des équations de consistance peut se faire par le calcul symbolique, mais également par le calcul numérique. Ainsi, on peut très bien approcher les équations de consistance par différences finies [86]. Par exemple, à l'ordre 1 :

$$D_h F = \frac{F(t_0 + h, y_0 + h, y'_0 + h y''_0) - F(t_0, y_0, y'_0)}{h}. \tag{C.19}$$

**Algorithme 22** Méthode des équations de consistance

1: Choix des équations à différencier
2: Différentiation numérique
3: Solution du problème non linéaire

Évidemment, la différentiation numérique peut poser des problèmes de robustesse, en particulier au niveau du conditionnement des matrices jacobiennes. Cette méthode présente l'avantage d'être générale. Dans le cas où les dérivations sont faites de manière analytique, il est démontré que cette méthode permet de trouver une solution initiale consistante d'une EDA quels que soient son indice et sa forme.

Les inconvénients de cette approche sont les suivants :

- L'imposition de contraintes, telle que l'imposition d'une valeur à une variable, conduit à un problème sur-déterminé. Dans ce cas, on pourra tout de même effectuer la résolution au sens des moindres carrés.

- Les systèmes non linéaires sont très raides, surtout dans le cas d'une approximation numérique par différences finies des équations de consistance. En particulier, la jacobienne pourra être singulière.

**Partitionnement implicite et partitionnement unitaire** Lorsqu'il n'est pas possible de calculer le nombre exact de dérivations nécessaires pour chaque équation, la jacobienne obtenue ne sera pas de rang plein. Les méthodes présentées ici permettent de contourner le problème

en utilisant une idée de [29], où une méthode d'intégration préservant les contraintes algébriques a été proposée. Le lien avec le problème d'initialisation est fait dans [28] et est résumé ici. On considère maintenant que toutes les équations sont dérivées $k \geq \nu$ fois où $\nu$ est l'indice de l'EDA considérée. L'équation (C.18) peut être réécrite sous la forme :

$$G(z) = G(y, \dot{y}, \underbrace{w}_{[y^(2),\ldots,y^{k+1}]}, t). \tag{C.20}$$

Dans la suite, on notera $G_y$ $G_{\dot{y}}$ et $G_w$ les matrices jacobiennes associées aux variables $y$, $\dot{y}$ et $w$. On considèrera également que :

- $\bar{J} = [G_{\dot{y}}\ G_w]$ est telle que $\bar{J}x = b$ détermine de manière unique la partie de $x$ correspondant à $\dot{y}$.
- $J = [G_{\dot{y}}\ G_w\ G_x]$ a toutes ses lignes linéairement indépendantes.

On partitionne maintenant nos inconnues $y = (x_1, x_2)$, où $x_2$ contient les variables connues à $t_0$. On peut donc réécrire $z$ comme $(\tilde{z}, x_2)$, c'est-à-dire que $\tilde{z}$ contient toutes les variables inconnues à $t_0$. Dans [28], il est proposé de trouver le $\tilde{z}$ qui minimise l'équation (C.21) par une méthode de Gauss-Newton.

$$C(\tilde{z}) = \frac{1}{2} G(\tilde{z}, x_2, t)^T G(\tilde{z}, x_2, t) \tag{C.21}$$

Cette méthode sera généralisée en 2004 dans [2], au cas où $(x_1, x_2)^T = Uy$, avec $U$ une matrice unitaire et non plus une permutation de la matrice identité.