Extrapolation vectorielle et applications aux équations aux dérivées partielles

Sébastien Duminil

Université Lille Nord de France
Pôle de Recherche et d'Enseignement Supérieur

ulco UNIVERSITÉ DU LITTORAL CÔTE D'OPALE

UNIVERSITÉ DU LITTORAL CÔTE D'OPALE

École doctorale : Sciences Pour l'Ingénieur n° 72

Unité de recherche : Laboratoire de mathématiques pures et appliquées

Thèse présentée par : DUMINIL Sébastien

Soutenue le 6 juillet 2012

Pour obtenir le grade de docteur de l'Université du Littoral Côte d'Opale

Discipline/Spécialité : Mathématiques Appliquées

Titre de la thèse :

Extrapolation Vectorielle et Applications aux Équations aux Dérivées Partielles

Thèse dirigée par :
SADOK Hassane

Rapporteurs :

ERHEL	Jocelyne	INRIA Rennes Bretagne Atlantique
GIRAUD	Luc	INRIA Bordeaux Sud-Ouest

Jury :

JBILOU	Khalide	Université du Littoral	Président du jury
ERHEL	Jocelyne	INRIA Rennes Betagne Atlantique	Rapporteur
GIRAUD	Luc	INRIA Bordeaux Sud-Ouest	Rapporteur
ROSIER	Carole	Université du Littoral	Examinateur
MEURANT	Gérard	Directeur de recherche	Examinateur

Université Lille Nord de France
Pôle de Recherche et d'Enseignement Supérieur

ULCO UNIVERSITÉ DU LITTORAL CÔTE D'OPALE

UNIVERSITÉ DU LITTORAL CÔTE D'OPALE

Doctoral school: Sciences Pour l'Ingénieur nº 72

University department: Laboratoire de mathématiques pures et appliquées

Thesis defended by: DUMINIL Sébastien

Defended on Friday 6th July, 2012

In order to become Doctor from Université du Littoral Côte d'Opale

Subject/Speciality: Applied Mathematics

Thesis title:

Vector Extrapolation and Applications to Partial Differential Equations

Ph. D. thesis supervised by:

SADOK Hassane

Recorders:

ERHEL	Jocelyne	INRIA Rennes Bretagne Atlantique
GIRAUD	Luc	INRIA Bordeaux Sud-Ouest

Dissertation committee:

JBILOU	Khalide	Université du Littoral	President of the jury
ERHEL	Jocelyne	INRIA Rennes Betagne Atlantique	Recorder
GIRAUD	Luc	INRIA Bordeaux Sud-Ouest	Recorder
ROSIER	Carole	Université du Littoral	Examiner
MEURANT	Gérard	Directeur de recherche	Examiner

Mots clés : Extrapolation vectorielle, RRE, MPE, MMPE, Systèmes linéaires, Méthodes de Krylov, CMRH, Implémentation parallèle, CMRH préconditionnée, Systèmes non linéaires, Équations de Navier-Stokes, Équations de Schrödinger, Méthodes multigrilles

Keywords: Vector extrapolation, Reduced Rank Extrapolation, Minimal Polynomial Extrapolation, Modified Minimal Polynomial Extrapolation, Linear systems, Krylov method, CMRH, Parallel implementation, Preconditioned CMRH, Nonlinear systems, Navier-Stokes problem, Schrodinger equation, Multigrid method

Cette thèse a été préparée au

laboratoire de mathématiques pures et appliquées
Centre Universitaire de la Mi-Voix
Maison de la Recherche Blaise Pascal
50 rue Ferdinand Buisson
BP 699
62228 Calais Cédex
☎ 03 21 46 55 90
FAX 03 21 46 55 86
✉ lmpa@univ-littoral.fr
Site http://www-lmpa.univ-littoral.fr

Remerciements

Je tiens à remercier en tout premier lieu Hassane SADOK qui a dirigé cette thèse. Tout au long de ces trois années, il a su me guider, m'encourager et me conseiller. Grâce à ses compétences et à sa confiance, Il m'a permis d'être à ce niveau aujourd'hui.

Je remercie les rapporteurs de cette thèse Jocelyne ERHEL, directrice de recherche de l'INRIA Rennes et Luc GIRAUD, directeur de recherche de l'INRIA Bordeaux Sud-Ouest pour le temps qu'ils ont consacré à la lecture de mon manuscrit et l'intérêt qu'ils ont porté à mon travail. Je les remercie aussi pour les corrections et les suggestions apportées à ce manuscrit.

Merci également à Khalide JBILOU, professeur à l'université du Littoral Côte d'Opale pour avoir accepté d'être président du jury, à Carole ROSIER, professeur à l'université du Littoral Côte d'Opale et Gérard MEURANT, ancien directeur de recherche du commissariat à l'énergie atomique d'avoir bien voulu juger ce travail.

Ce travail a été réalisé au laboratoire de mathématiques pures et appliquées Joseph Liouville de Calais. Je remercie mes collègues du LMPA pour leur accueil et leur soutien pendant ces trois années. En particulier, je tiens à remercier Philippe MARION, ingénieur d'études au LMPA pour son aide précieuse et le temps qu'il m'a consacré. Je remercie aussi mes collègues et amis Jean François et Baptiste pour leur bonne humeur, leur soutien et les bons moments passés.

Je voudrais finalement exprimer ma profonde gratitude à ma femme, Marjorie ainsi que ma famille et mes amis dont le soutien a été sans faille.

Que tous ceux qui m'ont aidé de près ou de loin dans l'élaboration de ce travail trouvent ici l'expression de ma sincère gratitude.

Résumé

Nous nous intéressons, dans cette thèse, à l'étude des méthodes d'extrapolation polynômiales et à l'application de ces méthodes dans l'accélération de méthodes de points fixes pour des problèmes donnés. L'avantage de ces méthodes d'extrapolation est qu'elles utilisent uniquement une suite de vecteurs qui n'est pas forcément convergente, ou qui converge très lentement pour créer une nouvelle suite pouvant admettre une convergence quadratique. Le développement de méthodes cycliques permet, de plus, de limiter le coût de calculs et de stockage.

Nous appliquons ces méthodes à la résolution des équations de Navier-Stokes stationnaires et incompressibles, à la résolution de la formulation Kohn-Sham de l'équation de Schrödinger et à la résolution d'équations elliptiques utilisant des méthodes multigrilles. Dans tous les cas, l'efficacité des méthodes d'extrapolation a été montrée.

Nous montrons que lorsqu'elles sont appliquées à la résolution de systèmes linéaires, les méthodes d'extrapolation sont comparables aux méthodes de sous espaces de Krylov. En particulier, nous montrons l'équivalence entre la méthode MMPE et CMRH. Nous nous intéressons, enfin, à la parallélisation de la méthode CMRH sur des processeurs à mémoire distribuée et à la recherche de préconditionneurs efficaces pour cette même méthode.

Abstract

In this thesis, we study polynomial extrapolation methods. We discuss the design and implementation of these methods for computing solutions of fixed point methods. Extrapolation methods transform the original sequence into another sequence that converges to the same limit faster than the original one without having explicit knowledge of the sequence generator. Restarted methods permit to keep the storage requirement and the average of computational cost low. We apply these methods for computing steady state solutions of incompressible flow problems modelled by the Navier-Stokes equations, for solving the Schrödinger equation using the Kohn-Sham formulation and for solving elliptic equations using multigrid methods. In all cases, vector extrapolation methods have a useful role to play.

We show that, when applied to linearly generated vector sequences, extrapolation methods are related to Krylov subspace methods. For example, we show that the MMPE approach is mathematically equivalent to CMRH method. We present an implementation of the CMRH iterative method suitable for parallel architectures with distributed memory. Finally, we present a preconditioned CMRH method.

Table des matières

Liste des tableaux

Table des figures

Liste des Algorithmes

Introduction Générale

En analyse numérique, on est souvent amené à résoudre un système d'équations. La technique de résolution dépend de ce système. Tout d'abord, ce système peut être linéaire. Dans ce cas, l'utilisation de méthodes dites de sous espaces de Krylov est considérée dans la plupart des cas. Mais là encore, la méthode choisie dépend de la matrice. La matrice peut être dense ou creuse, symétrique, hermitienne, etc... Si la matrice est symétrique définie positive, l'utilisation de la méthode du gradient conjugué sera le meilleur choix. Si elle n'est pas symétrique mais creuse, l'utilisation de la méthode GMRES est l'un des meilleurs choix possibles. Par contre, si la matrice est dense, l'utilisation de la méthode GMRES peut devenir problématique surtout lorsque la matrice est de grande taille et que la méthode converge lentement. En effet, la méthode GMRES requiert de construire deux matrices supplémentaires dont la taille croît en fonction du nombre d'itérations ce qui peut poser un problème d'espace mémoire. Dans ce cas, une méthode redémarrée a été introduite mais la convergence devient plus lente et non garantie. La méthode CMRH a été développée dans le but d'être un équivalent à la méthode GMRES. L'avantage de cette nouvelle méthode est le gain en espace mémoire et en nombre d'opérations arithmétiques. Les résultats ont montré que la convergence de cette méthode était similaire à la méthode GMRES.

La recherche actuelle en matière de résolution de systèmes linéaires est de réduire le temps d'exécution des programmes. Une partie s'intéresse à la recherche de préconditionneurs pour réduire le nombre d'itérations. On s'est donc intéressé à vérifier que la méthode CMRH s'adaptait à cette recherche de préconditionneurs.

Les avancées faites en informatique nous permettent aujourd'hui de pouvoir résoudre des systèmes de plus en plus grands ce qui rend le temps d'exécution très important. En effet, la place mémoire est de plus en plus conséquente mais la vitesse d'exécution des processeurs atteint ses limites. On s'intéresse donc de plus en plus à la parallélisation de nos algorithmes dans le but de réduire ce temps. Dans cette thèse, on s'est intéressé à la parallélisation de la méthode CMRH. En effet, on montre qu'avec cette méthode parallélisée, on obtient des résultats équivalents voir meilleurs qu'avec l'utilisation de la méthode GMRES parallélisée.

Prenons maintenant le cas où le système est non linéaire. On s'intéresse à la résolution d'équations du type $F(x) = 0$. La méthode la plus efficace est la méthode

de Newton. Il a été montré que cette méthode est à convergence quadratique locale sous certaines hypothèses. Elle est définie comme suit : Étant donné un vecteur initial x_0, on construit une suite $\{x_k\}_{k\in\mathbb{N}}$ par $x_{k+1} = x_k - J(x_k)^{-1}F(x_k)$, où $J(x)$ est la matrice Jacobienne de la fonction F au point x. Cette méthode possède néanmoins beaucoup d'inconvénients. Tout d'abord, la convergence n'est que locale dans le sens où il faut que le point initial soit proche de la solution. Ensuite, le calcul de la Jacobienne et la résolution du système linéaire en chaque point peut vite devenir très long et très coûteux si la taille du système augmente. Enfin, dans certains cas, on ne connaît pas explicitement la fonction F. Dans ce cas, le calcul de la matrice Jacobienne est impossible. Plusieurs pistes sont développées. La première consiste à approcher la matrice Jacobienne. Pour cela, les méthodes quasi-Newton ont été introduites. Elles comprennent, par exemple, la méthode de la corde, la méthode de Shamanskii ou encore la méthode Broyden. D'autres méthodes s'intéressent à la résolution du système linéaire $J(x_k)^{-1}F(x_k)$, ce sont les méthodes de Newton inexactes qui utilisent les méthodes de Krylov. Une troisième classe de méthodes s'intéresse directement à l'approximation de $J(x_k)^{-1}F(x_k)$. Les méthodes de type sécante et les méthodes d'extrapolation font partie de cette catégorie. On s'intéresse dans cette thèse aux méthodes d'extrapolation. Ces méthodes peuvent être décomposées en trois catégories : les méthodes d'extrapolation polynomiales, les méthodes dites d'ϵ-algorithmes et les méthodes dites hybrides. On s'intéressera uniquement aux méthodes polynomiales.

Il existe de nombreuses méthodes d'extrapolation polynomiales. Dans le cas scalaire, on peut citer la méthode de Lusternik et la méthode d'Aitken. Dans le cas vectoriel, les méthodes de Aitken, RRE, MPE et MMPE sont les plus connues. Les méthodes que nous venons de citer utilisent une suite de vecteurs, généralement issue d'une méthode de points fixes, pour créer une nouvelle suite possédant une meilleure convergence. Ces méthodes s'utilisent aussi bien dans la résolution de systèmes d'équations non linéaires que linéaires. Elles sont toutefois plus efficaces lorsqu'elles sont appliquées à la résolution de systèmes d'équations non linéaires.

Cette thèse se décompose de la façon suivante. Le premier chapitre sera consacré aux rappels de plusieurs définitions, théorèmes et méthodes qui seront utilisés tout au long de cette thèse.

Le deuxième chapitre portera sur les méthodes d'extrapolation. Nous définirons ces méthodes dans le cas scalaire puis dans le cas vectoriel. Pour le cas vectoriel, nous définirons les méthodes RRE, MPE et MMPE et détaillerons leur algorithme. Nous nous intéresserons à leurs applications, tout d'abord, dans la résolution de systèmes d'équations non linéaires où nous prouverons un résultat de convergence, puis dans la résolution de systèmes d'équations linéaires où nous donnerons le lien avec les méthodes de sous espaces de Krylov.

Les trois chapitres suivants présentent les résultats de l'application des méthodes d'extrapolation lors de la résolution d'équations aux dérivées partielles linéaires et

non linéaires. Dans le chapitre 3, nous nous intéresserons à la résolution numérique des équations de Navier-Stokes stationnaires dans le cas de fluides incompressibles. Le principe et les méthodes de résolution seront rappelés. Le chapitre 4 porte sur la résolution de la formulation Kohn-Sham de l'équation de Schrödinger. Pour cette équation, des méthodes quasi-Newton sont actuellement utilisées. Nous comparons ces méthodes à nos méthodes d'extrapolation. La résolution d'équations aux dérivées partielles linéaires est étudiée dans le chapitre 5. Pour résoudre ces équations, les méthodes multi grilles ont été définies. Ces méthodes possèdent, dans certains cas, une convergence très lente. Pour ces cas, l'utilisation des méthodes d'extrapolation permettra d'améliorer cette convergence.
Enfin, dans le dernier chapitre, nous nous intéresserons à la résolution de systèmes linéaires par la méthode CMRH. La recherche de préconditionneurs et l'utilisation en parallèle de cet algorithme seront considérées. Des comparaisons avec la méthode GMRES seront aussi présentées.

Chapitre 1

Rappels et Définitions

1.1 Introduction

Dans ce chapitre, nous introduisons les notations et les définitions qui seront utilisées tout au long de cette thèse. Nous donnons quelques méthodes usuelles de factorisation de matrices, puis déduisons de celles-ci des méthodes de résolutions de systèmes linéaires comme la méthode GMRES. Nous rappelons ensuite les définitions concernant les méthodes préconditionnées. Enfin, nous donnons les définitions et quelques applications des méthodes de résolution de systèmes non linéaires comme la méthode de Newton et ses variantes.

1.2 Notations et Définitions

1.2.1 Espaces vectoriels et vecteurs

On note par $\mathbb{R}^N$ (respectivement $\mathbb{C}^N$) l'espace vectoriel de dimension N sur le corps des nombres réels $\mathbb{R}$ (respectivement des nombres complexes $\mathbb{C}$).
La base canonique de $\mathbb{R}^N$ sera notée $(e_1^{(N)}, e_2^{(N)}, \cdots, e_N^{(N)})$. Lorsque que la dimension sera évidente, on notera $(e_1, e_2, \cdots, e_N)$.
Un vecteur $v \in \mathbb{R}^N$ de composantes $(v)_1, \cdots, (v)_N$ est représenté par le vecteur colonne

$$v = \begin{pmatrix} (v)_1 \\ (v)_2 \\ \vdots \\ (v)_N \end{pmatrix}.$$

Le vecteur transposé de v est le vecteur ligne $v^T = ((v)_1, (v)_2, \cdots, (v)_N)$.
Pour deux vecteurs réels u et v, le produit scalaire euclidien de u et v noté (u, v) est

défini par :

$$(u,v) = v^T u = \sum_{i=1}^{N} (v)_i (u)_i.$$

La norme euclidienne associée à ce produit est définie par

$$\|v\| = (v,v)^{\frac{1}{2}}.$$

On définit aussi la norme infinie d'un vecteur v par

$$\|v\|_\infty = \max_{i=1,\cdots,N} |(v)_i|.$$

1.2.2 Suite de vecteurs : définitions

Soit $\{s_k\}_{k\in\mathbb{N}}$ une suite de vecteurs de $\mathbb{R}^N$.

Différences finies

Définition 1.1

On définit les différences finies d'ordre i comme suit :

$$\Delta s_k = s_{k+1} - s_k, \quad k = 0,1,2,\cdots \text{ si } i = 1,$$

$$\Delta^i s_k = \Delta^{i-1} s_{k+1} - \Delta^{i-1} s_k \quad k = 0,1,2,\cdots, \text{ si } i > 1.$$

Convergence

Dans cette thèse, nous allons étudier des méthodes itératives. Il est donc intéressant d'étudier l'ordre de convergence de la suite générée par ces méthodes. Pour cela, nous aurons besoin des définitions suivantes, voir [17, 49].

Définition 1.2

Soit $s_* \in \mathbb{R}^N$, $s_k \in \mathbb{R}^N$, $k = 0,1,2,\cdots$. Alors la suite s_k converge vers s_* si

$$\lim_{k\to\infty} \|s_k - s_*\| = 0.$$

Si de plus, il existe une constante $K \in [0,1)$ et un entier $k_0 \geq 0$ tels que pour tout $k \geq k_0$,

$$\|s_{k+1} - s_*\| \leq K \|s_k - s_*\|,$$

alors la suite s_k converge linéairement vers s_*. Si pour une suite $\{c_k\}$ qui tend vers 0*,*

$$\|s_{k+1} - s_*\| \leq c_k \|s_k - s_*\|,$$

pour tout k, alors la suite s_k converge super-linéairement vers s_*. S'il existe des constantes $p > 1$, $K \geq 0$ et $k_0 \geq 0$ telles que s_k converge vers s_* pour tout $k \geq k_0$ et

$$\|s_{k+1} - s_*\| \leq K \|s_k - s_*\|^p,$$

alors la suite s_k converge vers s_* avec au moins un ordre p. Si $p = 2$ ou $p = 3$, alors la convergence est dite respectivement quadratique ou cubique.

1.2.3 Matrice

Soit $A \in \mathbb{R}^{M \times N}$ une matrice qui a M lignes et N colonnes. Les éléments de A seront notés $a_{i,j}$, $i = 1, \cdots, M$ et $j = 1, \cdots, N$. Les vecteurs colonnes de la matrice A seront notés a^j, $j = 1, \cdots, N$.
On définit la norme matricielle euclidienne par

$$\|A\|_2 = \max_{\|x\|=1} \|Ax\|$$

et la norme matricielle de Frobenius par

$$\|A\|_F = \left(\sum_{j=1}^{N} \|a^j\|^2 \right)^{\frac{1}{2}}.$$

1.2.4 Matrices particulières

Soit A une matrice carrée d'ordre N.

Définition 1.3

La matrice A est dite :

- *symétrique si $A^T = A$,*
- *diagonale si $a_{i,j} = 0$ pour $i \neq j$,*
- *triangulaire inférieure si $a_{i,j} = 0$ pour $i < j$,*
- *triangulaire supérieure si $a_{i,j} = 0$ pour $i > j$,*
- *orthogonale si $AA^T = A^T A = I_N$,*
- *Hessenberg supérieure si $A_{i,j} = 0$ pour $i > j + 1$,*

– *définie positive si* $(Ax, x) > 0$ *pour tout* $x \in \mathbb{R}^N$ *et* $x \neq 0$.

La matrice identité d'ordre N sera notée I_N.

Nous donnons maintenant une proposition permettant de définir la formule de Sherman-Morrison.

Proposition 1. *Soient B une matrice inversible d'ordre N et* $u, v \in \mathbb{R}^N$. *Alors la matrice* $B + uv^T$ *est inversible si et seulement si* $1 + v^T B^{-1} u \neq 0$. *Dans ce cas, nous avons la formule suivante :*

$$(B + uv^T)^{-1} = \left(I - \frac{(B^{-1}u)v^T}{1 + v^T B^{-1} u} \right) B^{-1}. \tag{1.1}$$

La formule généralisée, connue sous le nom de formule de Sherman-Morrison-Woodbury, est définie dans la proposition suivante.

Proposition 2. *Soient* $A \in \mathbb{R}^{N\times N}$ *et* $C \in \mathbb{R}^{M\times M}$ *deux matrices inversibles. Soient* $U \in \mathbb{R}^{N\times M}$ *et* $V \in \mathbb{R}^{M\times N}$. *Si la matrice* $\left(C^{-1} + VA^{-1}U\right)$ *est inversible, alors* $\left(A + UC^{-1}V\right)$ *l'est aussi et vérifie la formule de Sherman–Morrison–Woodbury donnée comme suit :*

$$\left(A + UC^{-1}V\right)^{-1} = A^{-1} - A^{-1}U\left(C^{-1} + VA^{-1}U\right)^{-1} VA^{-1}. \tag{1.2}$$

1.2.5 Inverse généralisé d'une matrice

Théorème 1.1

Soit une matrice $A \in \mathbb{R}^{M\times N}$. *Alors il existe une unique matrice que l'on note* $A^+ \in \mathbb{R}^{N\times M}$ *vérifiant les conditions suivantes :*

– $AA^+A = A$,
– $A^+AA^+ = A^+$,
– $(AA^+)^T = AA^+$,
– $(A^+A)^T = A^+A$.

Nous donnons la proposition suivante [7].

Proposition 3. *La matrice* A^+ *est appelée l'inverse généralisé ou le pseudo-inverse de la matrice A et est définie par :*

$A^+ = A^T(AA^T)^{-1}$ *si rang(A)=M,*

$A^+ = (A^TA)^{-1}A^T$ *si rang(A)=N.*

Lemme 1. *Soit A une matrice réelle d'ordre $M \times N$ avec $1 \leq N \leq M$ et de $rang(A) = N$. Alors*

$$\left\| A^{+} \right\| \leq \frac{\|A\|_F^{N-1}}{\sqrt{det(A^T A)}}.$$

Voir [40] pour la preuve de ce lemme.

1.3 Factorisation d'une matrice

1.3.1 Méthode de Gram-Schmidt

Théorème 1.2

Soit $A \in \mathbb{R}^{M \times N}$ une matrice de rang maximal avec $M \geq N$. Alors il existe un unique couple de matrices (Q, R) tel que $A = QR$, où $Q \in \mathbb{R}^{M \times N}$ est une matrice orthogonale et $R \in \mathbb{R}^{N \times N}$ est une matrice triangulaire supérieure.

La preuve de ce théorème se trouve dans [53]. La méthode de Gram-Schmidt permet d'obtenir la décomposition QR d'une matrice A. Soit $A = (a^1, \cdots, a^N)$ où $a^i \in \mathbb{R}^M$ sont les vecteurs colonnes de A. On note $Q = (q^1, \cdots, q^N)$ et $R = (r_{i,j})$. L'algorithme 1 présente l'implémentation classique de la méthode de Gram-Schmidt [53].

Algorithme 1: Algorithme de la méthode de Gram-Schmidt classique

$r_{1,1} = \|a^1\|_2$;
$q^1 = \frac{a^1}{r_{1,1}}$;
pour $j = 2, \cdots, n$ **faire**
  $r_{i,j} = (a^j, q^i),\ i = 1, \cdots, j-1$;
  $r_{j,j} = \|a_j - \sum_{i=1}^{j-1} r_{i,j} q^i\|_2$;
  $q^j = \frac{(a^j - \sum_{i=1}^{j-1} r_{i,j} q^i)}{r_{j,j}}$;
fin

Le problème de cet algorithme est qu'il n'est pas stable numériquement. On préférera utiliser la version modifiée (voir Algorithme 2). Il a été montré que ces deux algorithmes sont mathématiquement équivalents mais que leur comportement numérique est différent.

1.3.2 Décomposition LU

Algorithme 2: Algorithme de la méthode de Gram-Schmidt modifiée

$r_{1,1} = \|a^1\|_2$;
$q^1 = \frac{a^1}{r_{1,1}}$;
pour $j = 2, \cdots, n$ **faire**
  $w = a^j$;
  pour $i = 1, \cdots, j-1$ **faire**
    $r_{i,j} = (w, q^i)$;
    $w = w - r_{i,j} q^i$;
  fin
  $r_{j,j} = \|w\|_2$;
  $q^j = \frac{w}{r_{j,j}}$;
fin

Théorème 1.3

Soit $A \in \mathbb{R}^{N\times N}$ une matrice inversible. Alors il existe un triplet (P, L, U) tel que $A = PLU$ où $P \in \mathbb{R}^{N\times N}$ est une matrice de permutation, $L \in \mathbb{R}^{N\times N}$ est une matrice triangulaire inférieure et $U \in \mathbb{R}^{N\times N}$ est une matrice triangulaire supérieure.

L'algorithme 3 présente la décomposition *LU* sans pivotage.

Algorithme 3: Algorithme de la décomposition LU

$u_{1,1} = a_{1,1}$;
pour $j = 2, \cdots, N$ **faire**
  $u_{1,j} = a_{1,j}$;
  $l_{j,1} = \frac{a_{j,1}}{a_{1,1}}$;
fin
pour $i = 2, \cdots, N-1$ **faire**
  $u_{i,i} = a_{i,i} - \sum_{k=1}^{i-1} l_{i,k} u_{k,i}$;
  pour $j = i+1, \cdots, N$ **faire**
    $u_{i,j} = a_{i,j} - \sum_{k=1}^{i-1} l_{i,k} u_{k,j}$;
    $l_{j,i} = \frac{1}{u_{i,i}} \left(a_{j,i} - \sum_{k=1}^{i-1} l_{j,k} u_{k,i}\right)$;
  fin
fin
$u_{N,N} = a_{N,N} - \sum_{k=1}^{N-1} l_{N,k} u_{k,N}$;

1.3.3 Factorisation de Cholesky

Théorème 1.4

Soit $A \in \mathbb{R}^{N\times N}$ une matrice symétrique, définie positive. Alors il existe une matrice L telle que $A = LL^T$ où $L \in \mathbb{R}^{N\times N}$ est une matrice triangulaire inférieure. Si, de plus, on impose que les éléments diagonaux de la matrice L soient tous positifs, alors cette factorisation est unique.

L'algorithme 4 présente cette méthode.

Algorithme 4: Algorithme de la factorisation de Cholesky

$l_{1,1} = \sqrt{a_{1,1}}$;
$l_{i,1} = a_{i,1}/l_{1,1}$, $i = 2, \cdots, N$;
pour $j = 2 : N$ **faire**
  $v = a^j - \sum_{k=1}^{j-1} l_{j,k} l^k$;
  $l_{j,j} = \sqrt{(v)_j}$;
  $l_{i,j} = (v)_i / l_{j,j}$, $i = j+1, \cdots, N$;
fin

1.4 Résolution de systèmes linéaires

On considère le système d'équations linéaires suivant :

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

où $A \in \mathbb{R}^{N\times N}$, x et b des vecteurs de $\mathbb{R}^N$. On s'intéresse ici aux méthodes créant une base du sous espace de Krylov $K_k(A, v)$ défini comme étant le sous espace vectoriel engendré par $v, Av, \cdots, A^{k-1}v$:

$$K_k(A, v) = vect\{v, Av, \cdots, A^{k-1}v\}.$$

On notera par K_k la matrice de taille $N \times k$, dont les colonnes sont formées par $[v, Av, \cdots, A^{k-1}v]$.

1.4.1 Processus d'Arnoldi

La méthode d'Arnoldi [3] transforme une matrice A non hermitienne en une matrice H, Hessenberg supérieure. Elle utilise le processus d'Arnoldi décrit dans l'algorithme 5. Ce processus construit une base orthonormale $\{v^1, \cdots, v^k\}$ du sous espace de Krylov $K_k(A, v)$. Il utilise le procédé de Gram-Schmidt modifié.

Théorème 1.5

L'algorithme de Arnoldi s'arrête à l'itération m si et seulement si le degré du polynôme minimal de A pour le vecteur v est égal à m.

La preuve de ce théorème se trouve dans [53].

Algorithme 5: Algorithme du processus d'Arnoldi

Entrée : $v \in \mathbb{R}^N$ et k ;
$v^1 = \frac{v}{\|v\|_2}$;
pour $j = 1, \cdots, k$ **faire**
  $w = Av^j$;
  pour $i = 1, \cdots, j$ **faire**
    $h_{i,j} = (w, v^i)$;
    $w = w - h_{i,j} v^i$;
  fin
  $h_{j+1,j} = \|w\|_2$;
  Si $h_{j+1,j} = 0$, Stop ;
  Sinon $v^{j+1} = \frac{w}{h_{j+1,j}}$;
fin

1.4.2 Méthode GMRES

Considérons le système (1.3). Soient x_0 un vecteur initial et $r_0 = b - Ax_0$ le résidu correspondant. La méthode GMRES [56] consiste à construire une suite d'approximations $\{x_k\}$ telle que les deux conditions suivantes soient vérifiées :

$$x_k \in x_0 + K_k(A, r_0) \tag{1.4}$$

$$r_k = b - Ax_k \perp AK_k(A, r_0). \tag{1.5}$$

La méthode GMRES utilise le processus d'Arnoldi pour construire une base orthonormale $V_k = [v^1, \cdots, v^k]$ de $K_k(A, r_0)$.

L'algorithme 6 résume la méthode GMRES.

1.4.3 Processus de Hessenberg

Le processus de Hessenberg [34] est décrit comme un algorithme servant à calculer le polynôme caractéristique d'une matrice A donnée. Mais il peut aussi être utilisé

Algorithme 6: Méthode GMRES

Soit $x_0 \in \mathbb{R}^N$;
Calculer $r_0 = b - Ax_0$;
$\beta = \|r_0\|$;
Processus d'Arnoldi :
$v^1 = r_0/\beta$;
pour $k = 1, \cdots,$ *jusqu'à convergence* **faire**
    $u = Av^k$;
    pour $j = 1, \cdots, k$ **faire**
        $h_{j,k} = {v^j}^T u$;
        $u = u - h_{j,k} v^j$;
    fin
    $h_{k+1,k} = \|u\|$;
    $v^{k+1} = u/h_{k+1,k}$;
    Mettre à jour la factorisation QR de H_k ;
    Appliquer les rotations de Givens à H_k et βe_1 ;
fin
Résoudre $H_k d_k = \beta e_1$;
Mettre à jour $x_k = x_0 + V_k d_k$;

pour transformer cette même matrice A en une matrice de Hessenberg supérieure. La méthode de Hessenberg présentée dans l'algorithme 7 construit une base $\{l^1, \cdots, l^k\}$ du sous espace de Krylov $K_k(A, v)$. L'algorithme 7 commence par calculer $l^1 = \frac{v}{\alpha}$ avec $\alpha = (v)_1$. À l'étape $j+1$, on commence par calculer $w = Al^j$, puis on soustrait successivement un multiple de l^1 pour annuler $(w)_1$, un multiple de l^2 pour annuler $(w)_2$, jusqu'à soustraire un multiple de l^j pour annuler $(w)_j$. Le vecteur obtenu est alors normalisé pour que l'élément $(l^{j+1})_{j+1}$ soit égal à un .

Algorithme 7: Algorithme du processus de Hessenberg

Entrée : $v \in \mathbb{R}^N$ et k;
$\beta = ((e)_1, v)$ et $l^1 = \frac{v}{\beta}$;
pour $j = 1, \cdots, k$ **faire**
  $u = Al^j$;
  pour $i = 1, \cdots, j$ **faire**
    $h_{i,j} = (u)_i$;
    $u = u - h_{i,j} l^i$;
  fin
  $h_{j+1,j} = (u)_{j+1}$;
  $l^{j+1} = u / h_{j+1,j}$;
fin

L'intérêt de cet algorithme est la réduction du coût du nombre d'opérations arithmétiques et le coût de stockage par rapport au processus d'Arnoldi. Nous reviendrons sur ces détails dans le chapitre suivant.

1.4.4 Préconditionnement

Considérons le système (1.3). Étant donnée une matrice M inversible d'ordre N, le système

$$M^{-1}Ax = M^{-1}b, \tag{1.6}$$

admet la même solution que (1.3). On dira que M est la matrice de préconditionnement associée au système préconditionné (1.6).

La convergence d'une méthode itérative dépendra beaucoup des propriétés spectrales de la nouvelle matrice $M^{-1}A$. Dans la pratique, on ne calcule pas la matrice inverse M^{-1}. En fait, dans les méthodes itératives de sous espaces de Krylov, on a seulement besoin de connaître le produit matrice-vecteur. Lorsque le système est préconditionné, le produit matrice-vecteur correspond à

$$w = M^{-1}Av \Longleftrightarrow Mw = Av. \tag{1.7}$$

Par conséquent, il faut pouvoir résoudre facilement le système

$$Mw = c. \tag{1.8}$$

Le principal problème est le choix de la matrice M. Ce qui est une question fort délicate qui n'a pas encore de solution universelle. Nous pouvons définir quelques propriétés qui peuvent être utiles lors de la construction d'un préconditionneur.

Remarque 1. *M doit être le plus près possible de A, c'est-à-dire :*

- $Sp(M^{-1}A) \cong Sp(I)$,
- $\|M^{-1}A - I\|_2 < \epsilon$.

Remarque 2. *On doit respecter la structure de la matrice pour certains problèmes.*

Remarque 3. *D'après l'étude de la convergence des méthodes de sous espaces de Krylov, les valeurs propres de $M^{-1}A$ doivent être regroupées et être éloignées de* 0.

Préconditionnement à droite

Définition 1.4

On dira que M est une matrice de préconditionnement à droite si le système (1.3) est transformé sous la forme :

$$AM^{-1}y = b. \tag{1.9}$$

Dans ce cas, la solution s'obtient en posant :

$$x = M^{-1}y. \tag{1.10}$$

Préconditionnement à gauche

Définition 1.5

On dira que M est une matrice de préconditionnement à gauche si le système (1.3) est transformé sous la forme :

$$M^{-1}Ax = M^{-1}b. \tag{1.11}$$

C'est ce préconditionnement qui sera utilisé dans la suite de cette thèse lors de l'étude de la méthode CMRH (cf. Chapitre 6).

Préconditionnement à droite et à gauche

On peut aussi combiner ces deux types de préconditionnement.

Définition 1.6

On dira que le système (1.3) est préconditionné à gauche et à droite si on choisit deux matrices M_G et M_D telles que

$$M_G^{-1} A M_D^{-1} y = M_G^{-1} b. \tag{1.12}$$

Dans ce cas, la solution s'obtient en posant :

$$x = M_D^{-1} y. \tag{1.13}$$

Ce type de préconditionnement est surtout utilisé lorsque la matrice A est symétrique. En effet, on choisira les matrices M_G et M_D de telle sorte que la matrice $M_G^{-1} A M_D^{-1}$ reste symétrique.

1.5 Résolution de systèmes non linéaires

Nous nous intéressons maintenant à la résolution de systèmes non linéaires de la forme

$$F(s) = 0 \tag{1.14}$$

où F est une fonction de $\mathbb{R}^N$ dans $\mathbb{R}^N$. On pose s_* une solution exacte de (1.14). La méthode la plus connue pour résoudre ce problème est la méthode de Newton. Nous commencerons par la définir. Nous verrons aussi que cette méthode possède beaucoup d'inconvénients. Pour pallier cela, des variantes de cette méthode ont été introduites. Nous détaillerons quelques-unes de celles-ci.

1.5.1 Méthode de Newton

La méthode de Newton est définie par

$$s_{k+1} = s_k - J(s_k)^{-1} F(s_k),\ k = 0, 1, \cdots \tag{1.15}$$

où $J(s_k)$ définit la matrice jacobienne de la fonction F au point s_k. L'algorithme 8 présente la méthode de Newton. Chaque itération de la méthode de Newton nécessite une évaluation de la fonction F et une évaluation de la jacobienne J ainsi que la résolution d'un système linéaire. Cette méthode devient donc très coûteuse lorsque la taille du problème est grande. De plus, la résolution du système linéaire implique le

Algorithme 8: Méthode de Newton

Entrée : $s_0 \in \mathbb{R}^N$ et tol;
tant que $\|F(s_k)\| > tol$ **faire**
  Résoudre le système $J(s_k)d_k = -F(s_k)$;
  Mettre à jour $s_{k+1} = s_k + d_k$;
fin

calcul de la matrice jacobienne qui peut être singulière ou mal-conditionnée. Un point important de la méthode de Newton est qu'elle possède une convergence quadratique locale. Nous définissons $V(s_*, r)$ le voisinage ouvert de rayon r autour de s_* :

$$V(s_*, r) = \{y \in \mathbb{R}^N / \|y - s_*\| < r\}. \tag{1.16}$$

Le théorème 1.6 donne des conditions pour la convergence de la méthode de Newton, voir [17, 41, 49].

Théorème 1.6

Soit $F : \mathbb{R}^N \to \mathbb{R}^N$ une fonction continûment différentiable dans un ouvert convexe $\mathcal{D} \subset \mathbb{R}^N$. Supposons qu'il existe $s_* \in \mathbb{R}^N$ et $r, \beta > 0$ tels que $V(s_*, r) \subset \mathcal{D}$, $F(s_*) = 0$, $J(s_*)^{-1}$ existe avec $\|J(s_*)^{-1}\| \leq \beta$ et J vérifie

$$\exists K \geq 0, \ \ \forall x, y \in D \ \ \|J(x) - J(y)\| \leq K\|x - y\|. \tag{1.17}$$

Alors il existe $\epsilon > 0$ tel que pour tout $s_0 \in V(s_*, \epsilon)$, la suite $\{s_k\}$ générée par (1.15) est bien définie, converge vers s_* et vérifie

$$\|s_{k+1} - s_*\| \leq \beta K \|s_k - s_*\|^2. \tag{1.18}$$

Résultats numériques

Nous allons montrer, sur un exemple, que le temps d'exécution du problème augmente fortement lorsque la taille du système augmente.

Exemple 1. *Dans cet exemple, on résolve l'équation de Chandrasekhar [42], qui est l'équation intégrale obtenue de la théorie de transfert radiatif :*

$$F(H)(\mu) = H(\mu) - \left(1 - \frac{c}{2}\int_0^1 \frac{\mu H(\mu)}{\mu + \nu} d\nu\right)^{-1} = 0. \tag{1.19}$$

Nombre de points	100	500	1000	2000
Temps d'exécution	0.03	2.68	22.31	233.52

TABLE 1.1 – Temps d'exécution (en secondes) de la méthode de Newton pour le problème de Chandrasekhar

Cette équation admet exactement deux solutions pour $0 < c < 1$. *L'intégrale est approchée par la règle du point milieu sur une grille uniforme* $\{\mu_i\}$,

$$\int_0^1 H(\mu)d\mu \approx \frac{1}{N}\sum_{j=1}^{N} H(\mu_j), \tag{1.20}$$

avec $\mu_i = (i-1/2)/N$ *pour* $i = 1, \cdots, N$. *Le problème discrétisé résultant est*

$$F_i(H) = (H)_i - \left(1 - \frac{c}{2N}\sum_{j=1}^{N}\frac{\mu_i (H)_j}{\mu_i + \mu_j}\right)^{-1}. \tag{1.21}$$

Nous exprimons (1.21) sous une forme plus compacte. On pose

$$A_{i,j} = \frac{c\mu_i}{2N(\mu_i + \mu_j)}. \tag{1.22}$$

Avec cette matrice, $F(x)$ *peut être rapidement évaluée par*

$$(F(x))_i = (x)_i - (1 - (Ax)_i)^{-1}. \tag{1.23}$$

La matrice Jacobienne est obtenue en posant

$$(F'(x))_{ij} = \delta_{ij} - \frac{A_{ij}}{(1-(Ax)_i)^2}. \tag{1.24}$$

Nous résolvons cette équation avec une estimation initiale $x_0 = (1, \cdots, 1)^T$ et $c = 0.1$. Nous arrêtons la méthode lorsque $\|F(x)\| \leq 10^{-7}$. La méthode de Newton converge en deux itérations. On fait varier le nombre de points sur la grille. Le tableau 1.1 montre le temps d'exécution de la méthode de Newton en fonction du nombre de points sur la grille. On observe bien, sur cet exemple, que le temps d'exécution augmente très rapidement même pour un nombre de points relativement petit.

Pour pallier ce problème de temps, deux classes de méthodes ont été introduites. L'une consiste à résoudre approximativement le système linéaire, on obtient les méthodes de Newton inexactes. L'autre utilise une approximation de la jacobienne, on aboutit aux méthodes quasi-Newton. Nous donnons maintenant les détails de ces deux classes.

1.5.2 Méthodes de Newton inexactes

Les méthodes de Newton inexactes utilisent l'idée de résoudre approximativement l'équation linéaire pour l'itération de Newton. Une méthode de Newton inexacte [16] est une méthode qui génère une suite $\{s_k\}$ comme dans l'algorithme 9. Le paramètre

Algorithme 9: Méthode de Newton inexacte

Entrée : $s_0 \in \mathbb{R}^N$ et tol;
tant que $\|F(s_k)\| > tol$ **faire**
  Résoudre le système $J(s_k)d_k = -F(s_k) + r_k$;
  avec $\|r_k\| \leq \eta_k \|F(s_k)\|$;
  Mettre à jour $s_{k+1} = s_k + d_k$;
fin

η_k dans l'algorithme 9 s'appelle terme forçant. Nous donnons un théorème donnant des résultats sur la convergence des méthodes de Newton inexactes en fonction du terme forçant. La preuve de ce théorème se trouve dans [66].

Théorème 1.7

Soit $F : \mathbb{R}^N \to \mathbb{R}^N$ une fonction continûment différentiable dans un domaine ouvert convexe $\mathscr{D} \subset \mathbb{R}^N$ contenant s_*. On suppose que $F(s_*) = 0$ et que $J(s_*)$ est non singulière et K-lipschitzienne dans D. Supposons que la suite $\{\eta_k\}$ satisfait $0 \leq \eta_k \leq \eta < 1$. Alors si s_0 est suffisamment proche de s_*, la suite $\{s_k\}$ générée par la méthode de Newton inexacte converge vers s_* linéairement. De plus, si $\eta_k \to 0$, alors la suite $\{s_k\}$ converge super-linéairement vers s_*. Enfin, si $\eta_k = O(\|F(s_k)\|)$, alors la suite $\{s_k\}$ a une convergence quadratique.

Une étape très importante dans l'utilisation de cette méthode est le choix du terme forçant. Le but est d'obtenir une convergence rapide avec un coût minimal de calculs. Plusieurs choix ont été proposés, tous possédant leurs avantages et leurs inconvénients. Nous utiliserons, dans cette thèse, la méthode de Newton-GMRES [54] qui consiste à résoudre le système linéaire en utilisant la méthode GMRES.

1.5.3 Méthodes quasi-Newton

Comme nous l'avons énoncé précédemment, les méthodes quasi-Newton utilisent une approximation de la matrice jacobienne pour résoudre plus rapidement le système non linéaire. Deux raisons principales peuvent expliquer le calcul d'une approximation. D'une part, les dérivées sont coûteuses à calculer pour les systèmes de grande taille. D'autre part, on ne dispose pas toujours d'une expression analytique de la fonction F et le calcul des dérivées est alors impossible. Le principe des méthodes

quasi-Newton est de modifier l'itération de Newton en approchant la Jacobienne par des formules de récurrence ne demandant que l'évaluation de la fonction F. L'itération de la méthode de Newton devient donc

$$s_{k+1} = s_k - J_k^{-1} F(s_k), \;\; J_k \approx J(s_k). \tag{1.25}$$

Plusieurs méthodes ont été proposées pour réduire le coût de la méthode de Newton. Nous allons introduire quelques-unes d'entre elles.

Méthode de la corde

Cette méthode consiste à remplacer la matrice jacobienne $J(s_k)$ par la matrice initiale $J(s_0)$ pour toutes les itérations. L'algorithme 10 décrit le processus de la méthode de la corde. Sous des hypothèses standard, la méthode de la corde a seulement une

Algorithme 10: Méthode de la corde

Entrée : $s_0 \in \mathbb{R}^N$ et tol;
tant que $\|F(s_k)\| > tol$ **faire**
    Résoudre le système $J(s_0)d_k = -F(s_k)$;
    Mettre à jour $s_{k+1} = s_k + d_k$;
fin

convergence linéaire comme le montre ce théorème prouvé dans [41].

Théorème 1.8

Supposons que les hypothèses standard soit satisfaites. Alors il existe des constantes $K > 0$ et $r > 0$ telles que si $s_0 \in V(s_*, r)$, alors les itérations de la méthode de la corde convergent linéairement vers s_* et

$$\|s_{k+1} - s_*\| \leq K\|s_0 - s_*\|\|s_k - s_*\|. \tag{1.26}$$

Méthode de Shamanskii

La méthode de Shamanskii est une version redémarrée de la méthode de la corde. La matrice jacobienne est recalculée après chaque m itérations, où m est un entier suffisamment petit. Si les hypothèses standard sont vérifiées, la méthode de Shamanskii a une convergence super-linéaire d'ordre $m+1$. Le théorème suivant énonce ce résultat. La preuve est donnée dans [41].

Théorème 1.9

Supposons que les hypothèses standard soient satisfaites. Soit $m \geq 0$. Alors il existe

des constantes $K > 0$ et $r > 0$ telles que si $s_0 \in V(s_*, r)$, alors les itérations de la méthode de Shamanskii convergent super-linéairement vers s_* et

$$\|s_{k+1} - s_*\| \leq K\|s_k - s_*\|^{m+1}. \tag{1.27}$$

Une autre méthode quasi-Newton, très connue et très utilisée, existe sous le nom de méthode de Broyden. Cette méthode sera définie dans le chapitre 4.

Chapitre 2

Méthodes d'extrapolation vectorielle

2.1 Introduction

Très souvent en analyse numérique, on obtient une suite de vecteurs. C'est le cas avec les méthodes itératives pour résoudre des systèmes d'équations linéaires et non linéaires, dans le calcul d'éléments propres d'une matrice. La convergence de ces suites est souvent lente. Lorsqu'un processus itératif converge lentement, les méthodes d'extrapolation sont alors utilisées. Il existe de nombreuses méthodes d'extrapolation. Elles se décomposent en trois catégories : les méthodes polynomiales, les méthodes ϵ-algorithmes et enfin les méthodes hybrides. On s'intéressera uniquement aux méthodes d'extrapolation polynomiale qui sont les plus efficaces [39, 40].

Les méthodes d'extrapolation polynomiale vectorielle les plus populaires sont : la méthode MPE (Minimal Polynomial Extrapolation) de Cabay et Jackson [14], la méthode RRE (Reduced Rank Extrapolation) de Eddy [22] et Mesina [46] et la méthode MMPE (Modified Minimal Polynomial Extrapolation) de Sidi, Ford & Smith [65], Brezinsky [10] et Pugachev [51]. L'analyse de la convergence de ces méthodes est donnée dans [65]. Les algorithmes d'implémentation de ces méthodes ont été proposés dans [10, 22, 28, 38]. Les méthodes d'extrapolation vectorielle sont considérées pour être plus efficaces lorsqu'elles sont appliquées à la résolution de systèmes d'équations non linéaires [39, 40].

Le but de ce chapitre est de présenter les méthodes d'extrapolation vectorielles, de les comparer avec d'autres méthodes de résolution et d'obtenir de nouveaux résultats de convergence.

Ce chapitre se présente de la façon suivante. Nous commençons par définir les méthodes d'extrapolation dans le cas scalaire. Nous donnons la définition de l'accélération de la convergence. Nous définissons quelques exemples de méthodes qui seront généralisées dans la suite. Ces généralisations permettent de définir les méthodes vec-

torielles dont la méthode RRE, la méthode MPE et la méthode MMPE. Nous donnons les définitions, les implémentations et les algorithmes de chacune de ces méthodes dans le cas classique et dans le cas cyclique. Nous énonçons ensuite des résultats de convergence lorsque ces méthodes sont appliquées à la résolution de systèmes non linéaires. Nous nous intéressons aussi à la résolution de systèmes linéaires. Ces méthodes s'apparentent à des méthodes de sous espaces de Krylov. Il a été prouvé dans [54, 56] que les méthodes MPE et RRE sont mathématiquement équivalentes, respectivement, à la méthode d'Arnoldi et à la méthode GMRES. La méthode MMPE est quant à elle équivalente à la méthode CMRH développée par Sadok dans [59]. Nous rappelons donc cette méthode et donnons l'algorithme correspondant. Enfin, des résultats numériques seront donnés.

2.2 Méthodes d'extrapolation dans le cas scalaire

2.2.1 Introduction

Définition 2.1

Soient $\{s_k\}_{k\in\mathbb{N}}$ *et* $\{t_k\}_{k\in\mathbb{N}}$ *deux suites réelles convergentes vers la même limite* s_*. *On dira que la suite* $\{t_k\}_{k\in\mathbb{N}}$ *converge plus rapidement vers* s_* *que la suite* $\{s_k\}_{k\in\mathbb{N}}$ *ou que la suite* $\{t_k\}_{k\in\mathbb{N}}$ *accélère la convergence de la suite* $\{s_k\}_{k\in\mathbb{N}}$ *si*

$$\lim_{k\to\infty} \frac{t_k - s_*}{s_k - s_*} = 0.$$

Exemple 2. *Soit* $\{s_k\}_{k\in\mathbb{N}}$ *une suite réelle convergente vers* s_*. *Considérons la suite* $\{t_k\}_{k\in\mathbb{N}}$ *définie par*

$$t_k = \frac{s_k + s_{k+1}}{2}, \quad k = 0, 1, \cdots.$$

On montre facilement que

$$\lim_{k\to\infty} \frac{t_k - s_*}{s_k - s_*} = 0 \Leftrightarrow \lim_{k\to\infty} \frac{s_{k+1} - s_*}{s_k - s_*} = -1.$$

Ce qui montre que la suite $\{t_k\}_{k\in\mathbb{N}}$ *converge plus vite que* $\{s_k\}_{k\in\mathbb{N}}$ *lorsque* $\lim_{k\to\infty} \frac{s_{k+1}-s_*}{s_k-s_*} = -1$.

Le but de l'extrapolation est de trouver une suite $\{t_k\}_{k\in\mathbb{N}}$ à partir de la suite $\{s_k\}_{k\in\mathbb{N}}$ de telle sorte que $\{t_k\}_{k\in\mathbb{N}}$ converge vers la même limite que $\{s_k\}_{k\in\mathbb{N}}$ et de façon plus rapide.

Définition 2.2

Soit $\{s_k\}_{k\in\mathbb{N}}$ une suite réelle divergente. Si la nouvelle suite $\{t_k\}_{k\in\mathbb{N}}$ converge vers une limite s_*, on l'appellera l'anti-limite de la suite.

Corollaire 2.1. Soit s_* la limite d'une suite $\{s_k\}_{k\in\mathbb{N}}$. s_* peut s'écrire sous la forme suivante :

$$s_* = s_k - \frac{s_{k+1} - s_k}{\frac{s_{k+1}-s_*}{s_k-s_*} - 1}. \tag{2.1}$$

Si le comportement asymptotique du rapport $\frac{s_{k+1}-s_*}{s_k-s_*}$ est connu, on peut améliorer la convergence de la suite $\{s_k\}_{k\in\mathbb{N}}$.

2.2.2 Application à une suite particulière

Dans toute cette section, nous considérons la suite suivante :

$$s_k = s_* + a_1(\lambda_1)^k + a_2(\lambda_2)^k,\ k = 0, 1, \cdots \tag{2.2}$$

où $0 < |\lambda_2| < |\lambda_1| < 1$ et $a_1 a_2 \neq 0$. Alors

$$\frac{s_{k+1} - s_*}{s_k - s_*} = \frac{a_1(\lambda_1)^{k+1} + a_2(\lambda_2)^{k+1}}{a_1(\lambda_1)^k + a_2(\lambda_2)^k}. \tag{2.3}$$

Nous avons

$$\frac{s_{k+1} - s_*}{s_k - s_*} = \lambda_1 + O\left(\left(\frac{\lambda_2}{\lambda_1}\right)^k\right).$$

Méthode de Lusternik

Nous définissons une transformation linéaire qui donne de bons résultats de convergence lorsqu'elle est appliquée à la suite (2.2). Nous posons

$$t_k^{(1)} = s_k - \frac{s_{k+1} - s_k}{\lambda_1 - 1},\ k = 0, 1, 2, \cdots. \tag{2.4}$$

Proposition 4. *Soit $\{s_k\}_{k\in\mathbb{N}}$ la suite définie par (2.2). La méthode de Lusternik définie par (2.4) accélère la convergence de la suite $\{s_k\}_{k\in\mathbb{N}}$.*

Exemple 3. *On considère la suite suivante :*

$$s_k = 2 + 2(\frac{9}{10})^k + 10(\frac{1}{10})^k, k = 0, 1, \cdots.$$

Le tableau 2.1 donne les résultats des suites $\{s_k\}_{k\in\mathbb{N}}$, $\{t_k^{(1)}\}_{k\in\mathbb{N}}$ et $\{\frac{t_k^{(1)}-s_*}{s_k-s_*}\}_{k\in\mathbb{N}}$. On observe bien l'accélération de la convergence.

Itérations	s_k	$t_k^{(1)}$	$\frac{t_k^{(1)}-s_*}{s_k-s_*}$
0	14	-78	-6.6667
1	4.8	-6	-2.8571
2	3.72	1.2	-4.651210^{-1}
3	3.468	1.92	-5.449610^{-2}
4	3.3132	1.992	-6.092010^{-3}
5	3.1811	1.9992	-6.773510^{-4}
6	3.0629	1.9999	-7.526610^{-5}
7	2.9566	2	-8.363010^{-6}

TABLE 2.1 – Application de la méthode de Lusternik à la suite de l'exemple 3

Comme λ_1 est en général inconnu, on peut approcher le rapport $\frac{s_{k+1}-s_*}{s_k-s_*}$ par le rapport $\frac{\Delta s_{k+1}}{\Delta s_k}$ et on obtient la méthode :

Méthode Δ^2 de Aitken

Grâce à (2.3), on montre aussi que

$$\frac{s_{k+2}-s_{k+1}}{s_{k+1}-s_k} = \lambda_1 + O\left(\left(\frac{\lambda_2}{\lambda_1}\right)^k\right). \tag{2.5}$$

Comme dans la sous-section précédente, on définit une nouvelle transformation, connue sous le nom de méthode Δ^2 de Aitken [1], s'exprimant sous la forme suivante :

$$t_k^{(2)} = s_k - \frac{s_{k+1}-s_k}{\frac{s_{k+2}-s_{k+1}}{s_{k+1}-s_k}-1},\ k = 0,1,2,\cdots. \tag{2.6}$$

Contrairement à la méthode de Lusternik, la méthode de Aitken dépend uniquement des itérées s_k.

Proposition 5. *Soit $\{s_k\}_{k\in\mathbb{N}}$ la suite définie par (2.2). La méthode de Aitken définie par (2.6) accélère la convergence de la suite $\{s_k\}_{k\in\mathbb{N}}$.*

Exemple 4. *Considérons la suite de l'exemple précédent. Le tableau 2.2 donne les résultats des suites $\{s_k\}_{k\in\mathbb{N}}$, $\{t_k^{(2)}\}_{k\in\mathbb{N}}$ et $\{\frac{t_k^{(2)}-s_*}{s_k-s_*}\}_{k\in\mathbb{N}}$. La convergence est bien accélérée aussi pour cette méthode. Par rapport à la méthode précédente, il nous faut une itération de plus pour accéder au résultat voulu.*

Remarque 1. Si, dans la suite (2.2), $|\lambda_1| > 1$, la suite $\{s_k\}_{k\in\mathbb{N}}$ diverge, mais les deux suites $\{t_k^{(1)}\}_{k\in\mathbb{N}}$ et $\{t_k^{(2)}\}_{k\in\mathbb{N}}$ convergent vers l'anti-limite s_*.

Itérations	s_k	$t_k^{(2)}$	$\frac{t_k^{(2)}-s_*}{s_k-s_*}$
0	14	3.5764	1.313610^{-1}
1	4.8	3.3913	4.968910^{-1}
2	3.72	3.0667	6.201610^{-1}
3	3.468	2.4114	2.802610^{-1}
4	3.3132	2.0603	4.590210^{-2}
5	3.1811	2.0064	5.381910^{-3}
6	3.0629	2.0006	6.016710^{-4}
7	2.9566	2.0001	6.689810^{-5}
8	2.8609	2	7.433710^{-6}

TABLE 2.2 – Application de la méthode Δ^2 de Aitken à la suite de l'exemple 4

Maintenant, intéressons-nous plus en détail à la méthode de Aitken. Nous donnons plusieurs expressions différentes de (2.6) qui nous permettront de généraliser les résultats. En effet, la relation (2.6) peut s'exprimer de la façon suivante :

$$t_k^{(2)} = s_k - \frac{(\Delta s_k)^2}{\Delta^2 s_k} = s_k - \Delta s_k \left(\Delta^2 s_k\right)^{-1} \Delta s_k, \tag{2.7}$$

où Δs_k et $\Delta^2 s_k$ sont données par la définition 1.1. Le terme $t_k^{(2)}$ peut être exprimé comme le quotient de deux déterminants :

$$t_k^{(2)} = \frac{\begin{vmatrix} s_k & s_{k+1} \\ \Delta s_k & \Delta s_{k+1} \end{vmatrix}}{\begin{vmatrix} 1 & 1 \\ \Delta s_k & \Delta s_{k+1} \end{vmatrix}}. \tag{2.8}$$

Nous pouvons aussi exprimer les itérées $t_k^{(2)}$ comme une combinaison barycentrique :

$$t_k^{(2)} = \gamma_0 s_k + \gamma_1 s_{k+1}, \tag{2.9}$$

où $\gamma_0 = \frac{\Delta s_{k+1}}{\Delta^2 s_k}$ et $\gamma_1 = -\frac{\Delta s_k}{\Delta^2 s_k}$. Par conséquent, γ_0 et γ_1 satisfont le système d'équations linéaires suivant :

$$\begin{cases} \gamma_0 + \gamma_1 = 1 \\ \gamma_0 \Delta s_k + \gamma_1 \Delta s_{k+1} = 0. \end{cases} \tag{2.10}$$

2.2.3 Généralisation et transformation de Shanks

On peut généraliser la méthode d'Aitken en considérant une transformation utilisant $m+1$ termes de la suite $\{s_k\}_{k\in\mathbb{N}}$. Ce cas est dû à Shanks [61]. La généralisation de

l'équation (2.8) est donnée par

$$t_{k,m} = \frac{\begin{vmatrix} s_k & s_{k+1} & \dots & s_{k+m} \\ \Delta s_k & \Delta s_{k+1} & \dots & \Delta s_{k+m} \\ \vdots & \vdots & & \vdots \\ \Delta s_{k+m-1} & \Delta s_{k+m} & \dots & \Delta s_{k+2m-1} \end{vmatrix}}{\begin{vmatrix} 1 & 1 & \dots & 1 \\ \Delta s_k & \Delta s_{k+1} & \dots & \Delta s_{k+m} \\ \vdots & \vdots & & \vdots \\ \Delta s_{k+m-1} & \Delta s_{k+m} & \dots & \Delta s_{k+2m-1} \end{vmatrix}}. \tag{2.11}$$

En utilisant la formule de Cramer, on peut en déduire une définition de $t_{k,m}$ sous la forme suivante :

$$t_{k,m} = \sum_{j=0}^{m} \gamma_j s_{k+j} \tag{2.12}$$

où les γ_j sont solutions du système d'équations linéaires suivant :

$$\begin{cases} \gamma_0 & + & \gamma_1 & + & \cdots & + & \gamma_m & = 1 \\ \gamma_0 \Delta s_k & + & \gamma_1 \Delta s_{k+1} & + & \cdots & + & \gamma_m \Delta s_{k+m} & = 0 \\ \vdots & & \vdots & & \cdots & & \vdots & \\ \gamma_0 \Delta s_{k+m-1} & + & \gamma_1 \Delta s_{k+m} & + & \cdots & + & \gamma_m \Delta s_{k+2m-1} & = 0. \end{cases} \tag{2.13}$$

2.2.4 Application à la méthode du point fixe

Considérons maintenant la suite formée par la méthode de points fixes suivante :

$$s_{k+1} = G(s_k),\ k = 0, 1, 2, \cdots \tag{2.14}$$

où la fonction G est non linéaire. Supposons, de plus, que la suite formée converge vers la limite s_*. On peut écrire :

$$\frac{s_{k+1} - s_*}{s_k - s_*} = G'(s_*) + O(s_k - s_*). \tag{2.15}$$

Comme pour la méthode de Lusternik, on peut construire une méthode d'accélération linéaire de la forme suivante :

$$t_k^{(3)} = s_k - \frac{s_{k+1} - s_k}{G'(s_*) - 1}.$$

L'avantage de cette méthode est que la convergence est quadratique :

$$t_k^{(3)} - s_* = O((s_k - s_*)^2).$$

Par contre, le principal inconvénient de cette transformation est que les itérées dépendent de s_*, la limite de la suite initiale qui est inconnue. Une idée couramment utilisée est de remplacer s_* par son approximation s_k. On obtient donc la transformation non linéaire suivante :

$$t_k^{(4)} = s_k - \frac{s_{k+1} - s_k}{G'(s_k) - 1}.$$

Cette transformation est aussi à convergence quadratique :

$$t_k^{(4)} - s_* = O((s_k - s_*)^2).$$

2.2.5 Méthode redémarrée ou itérée

Introduisons maintenant la méthode redémarrée. Soit x_0 un point initial. La suite x_k est formée en posant $s_0 = x_k$ puis en calculant la transformation t_i, à partir de la suite $s_0, s_1, \cdots$. Enfin on pose :

$$x_{k+1} = t_i.$$

Nous allons appliquer cette méthode redémarrée aux transformations vues précédemment en choisissant $i = 0$. Mais d'abord, introduisons la fonction F définie par :

$$F(x) = x - G(x). \tag{2.16}$$

Méthode de Steffensen

Commençons par utiliser la méthode de Aitken redémarrée. On pose

$$x_{k+1} = t_0^{(2)} = s_0 - \frac{s_1 - s_0}{\frac{s_2 - s_1}{s_1 - s_0} - 1}.$$

On obtient la méthode de Steffensen définie comme suit :

$$x_{k+1} = x_k - \frac{F(x_k)^2}{F(x_k - F(x_k)) - F(x_k)}.$$

Méthode de Newton

Appliquons maintenant la transformation non linéaire $t_k^{(4)}$ à notre méthode redémarrée. On obtient :

$$x_{k+1} = t_0^{(4)} = s_0 - \frac{s_1 - s_0}{G'(s_0) - 1}.$$

Ce qui aboutit à la méthode de Newton :

$$x_{k+1} = x_k - \frac{F(x_k)}{F'(x_k)}.$$

Les deux dernières méthodes présentées sont à convergence quadratique. Le principal inconvénient est que cette convergence est locale dans le sens que l'estimation initiale x_0 doit être assez proche de la solution s_*.

Exemple 5. *Prenons deux exemples qui montrent la convergence des méthodes de Newton et de Steffensen. Soient $F_1 : \mathbb{R} \to \mathbb{R}$ donnée par*

$$F_1(x) = arctan(x),$$

qui a pour solution exacte $s_* = 0$ et $F_2 : \mathbb{R} \to \mathbb{R}$ donnée par

$$F_2(x) = \cos(x) - x^3.$$

Nous choisissons $x_0 = 1$. Les figures 2.1 et 2.2 montrent la convergence du résidu non-linéaire des méthodes de Newton et de Steffensen.

Dans la suite de ce chapitre, nous généraliserons le processus de redémarrage aux suites vectorielles.

2.3 Méthodes polynomiales d'extrapolation vectorielle

Nous nous intéressons maintenant aux suites vectorielles. Considérons $\{s_k\}_{k\in\mathbb{N}}$, une suite de vecteurs de $\mathbb{R}^N$.

2.3.1 Définitions

Les méthodes polynomiales d'extrapolation sont définies comme suit :

$$t_{k,q} = \sum_{j=0}^{q} \gamma_j s_{k+j} \tag{2.17}$$

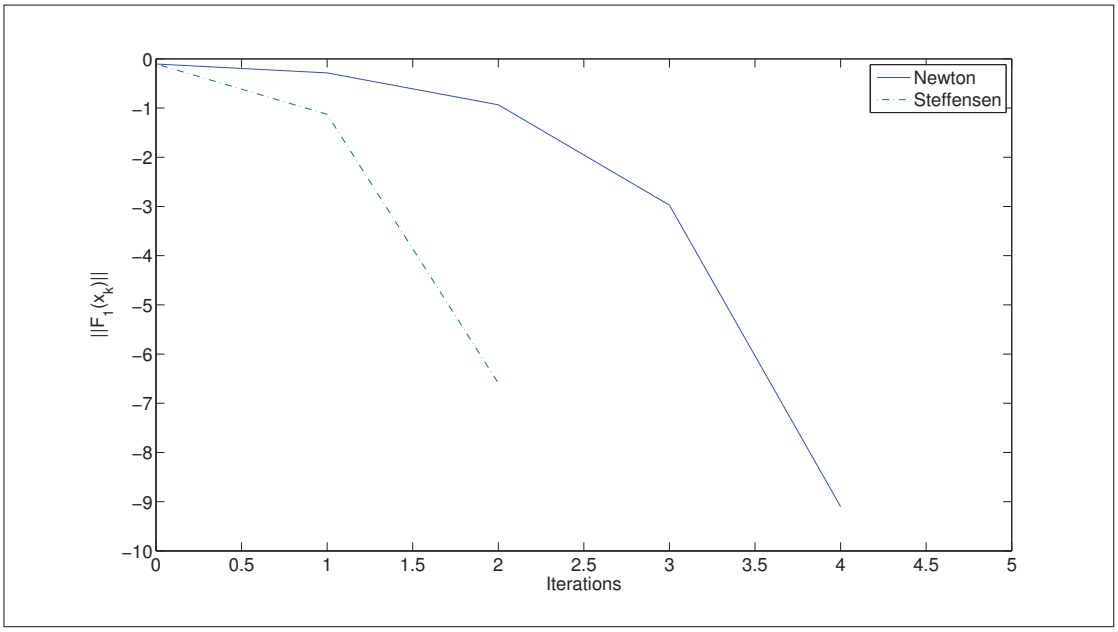


FIGURE 2.1 – Convergence des méthodes de Newton et de Steffensen pour la fonction F_1.

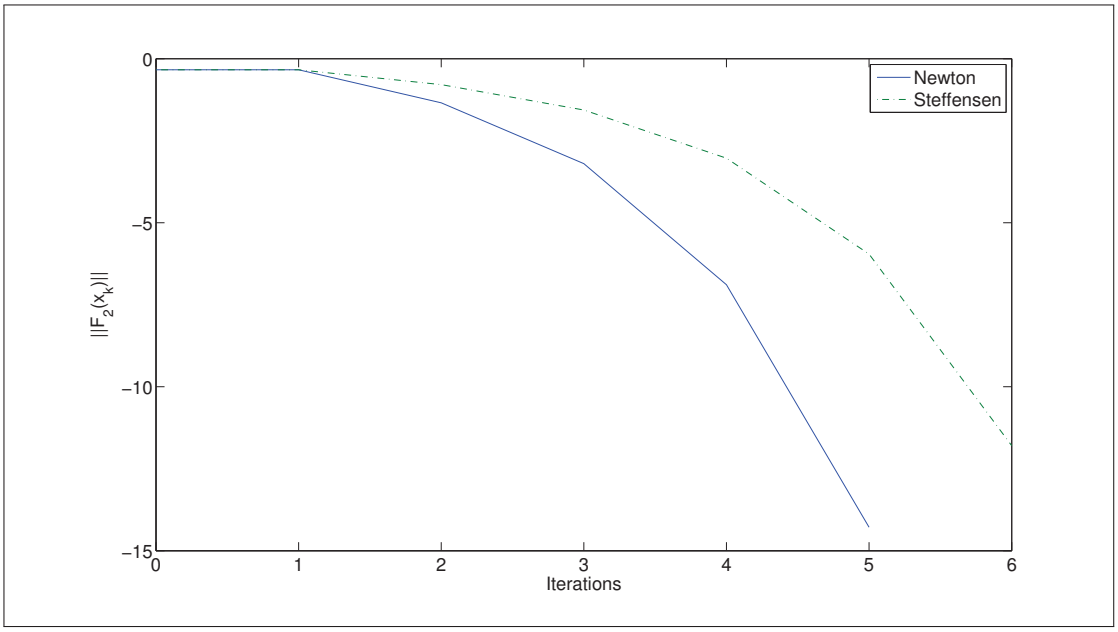


FIGURE 2.2 – Convergence des méthodes de Newton et de Steffensen pour la fonction F_2.

où k définit le premier terme de la suite et q le nombre de termes de la suite. $\gamma_j^{(q)}$ vérifie

$$\sum_{j=0}^{q} \gamma_j = 1, \ \sum_{j=0}^{q} \eta_{i,j}\gamma_j = 0, \ i = 0,1,\cdots,q-1, \tag{2.18}$$

avec les scalaires $\eta_{i,j}$ définis par $\eta_{i,j} = (y_i, \Delta s_{k+j})$ où les $y_i \in \mathbb{R}^N$ définissent la méthode utilisée.

En utilisant (2.18), la transformation (2.17) peut être exprimée sous la forme d'un quotient de deux déterminants :

$$t_{k,q} = \frac{\begin{vmatrix} s_k & s_{k+1} & \cdots & s_{k+q} \\ \eta_{0,0} & \eta_{0,1} & \cdots & \eta_{0,q} \\ \vdots & \vdots & \vdots & \vdots \\ \eta_{q-1,0} & \eta_{q-1,1} & \cdots & \eta_{q-1,q} \end{vmatrix}}{\begin{vmatrix} 1 & 1 & \cdots & 1 \\ \eta_{0,0} & \eta_{0,1} & \cdots & \eta_{0,q} \\ \vdots & \vdots & \vdots & \vdots \\ \eta_{q-1,0} & \eta_{q-1,1} & \cdots & \eta_{q-1,q} \end{vmatrix}}. \tag{2.19}$$

Notons Y_q, la matrice formée par les vecteurs $y_0,\cdots,y_{q-1}$ et $\Delta^i S_{k,(q+1)}$, les matrices définies par les vecteurs $\Delta^i s_k,\cdots,\Delta^i s_{k+q}$ pour $i = 1,2$. Si on remplace, dans le numérateur et le dénominateur de (2.19), chaque colonne j, $j = q+1,q,\cdots,2$ par la différence avec la colonne $j-1$, on obtient l'expression suivante :

$$t_{k,q} = \frac{\begin{vmatrix} s_k & \Delta S_{k,q} \\ Y_q^T \Delta s_k & Y_q^T \Delta^2 S_{k,q} \end{vmatrix}}{\left| Y_q^T \Delta^2 S_{k,q} \right|}, \tag{2.20}$$

qui, en utilisant la formule de Schur, peut s'exprimer sous la forme suivante :

$$t_{k,q} = s_k - \Delta S_{k,q} (Y_q^T \Delta^2 S_{k,q})^{-1} Y_q^T \Delta s_k. \tag{2.21}$$

On montre facilement que :

Proposition 6. *Soit $t_{k,q}$, la transformation donnée par la formule (2.21). Alors la transformation existe et est unique si et seulement si $det(Y_q^T \Delta^2 S_{k,q}) \neq 0$.*

2.3.2 Résidu généralisé

Dans cette section, nous définissons le résidu généralisé qui sera utile pour déterminer des résultats théoriques de convergence. Commençons par introduire une

nouvelle approximation

$$\tilde{t}_{k,q} = \sum_{j=0}^{q} \gamma_j s_{k+j+1}. \tag{2.22}$$

Dans [40], le résidu généralisé de $t_{k,q}$ a été défini de la façon suivante :

$$\tilde{r}(t_{k,q}) = \tilde{t}_{k,q} - t_{k,q}. \tag{2.23}$$

En utilisant (2.21), ce résidu peut être exprimé sous la forme suivante :

$$\tilde{r}(t_{k,q}) = \Delta s_k - \Delta^2 S_{k,q} (Y_q^T \Delta^2 S_{k,q})^{-1} Y_q^T \Delta s_k. \tag{2.24}$$

Si on définit $W_{k,q} = vect\{\Delta^2 s_k, \cdots, \Delta^2 s_{k+q-1}\}$ et $L_{k,q} = vect\{y_0, \cdots y_{q-1}\}$, d'après (2.24), on montre que :

Proposition 7. *Le résidu généralisé satisfait*

$$\tilde{r}(t_{k,q}) - \Delta s_k \in W_{k,q} \tag{2.25}$$

et

$$\tilde{r}(t_{k,q}) \perp L_{k,q}. \tag{2.26}$$

Ces deux dernières conditions montrent que le résidu généralisé est obtenu en projetant le vecteur Δs_k sur le sous espace $W_{k,q}$ orthogonalement à $L_{k,q}$.

2.3.3 Implémentation

Dans ce paragraphe et dans tous les sous-paragraphes implémentation suivant, on supposera que k est fixe. Sans restriction, on supposera même que $k = 0$ et nous remplacerons la notation $t_{0,q}$ par t_q. Le système linéaire (2.18) s'exprime de la manière suivante :

$$\left\{\begin{array}{lllllllll}
\gamma_0 & + & \gamma_1 & + & \ldots & + & \gamma_q & = & 1 \\
\gamma_0(y_0, \Delta s_0) & + & \gamma_1(y_0, \Delta s_1) & + & \ldots & + & \gamma_q(y_0, \Delta s_q) & = & 0 \\
\gamma_0(y_1, \Delta s_0) & + & \gamma_1(y_1, \Delta s_1) & + & \ldots & + & \gamma_q(y_1, \Delta s_q) & = & 0 \\
\vdots & & \vdots & & \vdots & & \vdots & & \vdots \\
\gamma_0(y_{q-1}, \Delta s_0) & + & \gamma_1(y_{q-1}, \Delta s_1) & + & \ldots & + & \gamma_q(y_{q-1}, \Delta s_q) & = & 0.
\end{array}\right. \tag{2.27}$$

Introduisons les scalaires β_i, pour $i = 0, \ldots, q$, définis par $\beta_i = \dfrac{\gamma_i}{\gamma_q}$. Dans ce cas, on a :

$$\gamma_i = \frac{\beta_i}{\sum_{i=0}^{q} \beta_i} \text{ pour } i = 0, \ldots, q-1 \text{ et } \beta_q = 1. \tag{2.28}$$

Avec ces nouvelles variables, le système (2.27) devient :

$$\begin{cases} \beta_0(y_0,\Delta s_0) & + & \beta_1(y_0,\Delta s_1) & + & \ldots & + & \beta_{q-1}(y_0,\Delta s_{q-1}) & = & -(y_0,\Delta s_q) \\ \vdots & & \vdots & & & & \vdots & & \vdots \\ \beta_0(y_{q-1},\Delta s_0) & + & \beta_1(y_{q-1},\Delta s_1) & + & \ldots & + & \beta_{q-1}(y_{q-1},\Delta s_{q-1}) & = & -(y_{q-1},\Delta s_q). \end{cases}$$

Ce système peut s'écrire sous la forme suivante :

$$(Y_q^T \Delta S_q)\,\beta = -Y_q^T \Delta s_q, \tag{2.29}$$

où $\beta = (\beta_0,\ldots,\beta_{q-1})^T$ et $\Delta S_q = (\Delta s_0,\ldots,\Delta s_{q-1})$.

Supposons maintenant que les coefficients $\gamma_0,\ldots,\gamma_q$ ont été calculés et introduisons les variables suivantes :

$$\begin{aligned} &\alpha_0 = 1-\gamma_0, \\ &\alpha_j = \alpha_{j-1} - \gamma_j \quad 1 \le j \le q-1. \end{aligned} \tag{2.30}$$

Dans ce cas, le vecteur t_q peut être exprimé sous la forme suivante :

$$t_q = s_0 + \sum_{j=0}^{q-1} \alpha_j \Delta s_j = s_0 + \Delta S_q\,\alpha, \tag{2.31}$$

où $\alpha = (\alpha_0,\ldots,\alpha_{q-1})^T$.

On remarque que pour déterminer les inconnues γ_i, nous devons d'abord calculer les β_i en résolvant le système d'équations linéaires (2.29).

Nous allons maintenant définir chaque méthode d'extrapolation, donner leur implémentation et leur algorithme respectif.

2.3.4 Méthode MPE

Définition

La méthode MPE (Modified Polynomial Extrapolation) a été définie par Cabay et Jackson dans [14]. Les scalaires $\eta_{i,j}$ sont définis par :

$$\eta_{i,j} = (\Delta s_{k+i}, \Delta s_{k+j}) \tag{2.32}$$

et

$$Y_q = \Delta S_{k,q}. \tag{2.33}$$

La transformation de la méthode MPE est donc définie par la relation suivante :

$$t_{k,q}^{MPE} = s_k - \Delta S_{k,q}(\Delta S_{k,q}^T \Delta^2 S_{k,q})^{-1} \Delta S_{k,q}^T \Delta s_k. \tag{2.34}$$

Implémentation

Supposons que le $rang(\Delta S_{q+1}) = q+1$. D'après le théorème 1.2, il existe une factorisation QR de la matrice ΔS_{q+1}. Posons $\Delta S_{q+1} = Q_{q+1}R_{q+1}$, où $Q_{q+1} = [q_0, q_1, \cdots, q_q] \in \mathbb{R}^{N\times(q+1)}$ est une matrice orthogonale et $R_{q+1} \in \mathbb{R}^{(q+1)\times(q+1)}$ est une matrice triangulaire supérieure dont les éléments diagonaux sont positifs. La matrice Q_{q+1} est obtenue à partir de la matrice $Q_q \in \mathbb{R}^{N\times q}$ en ajoutant le vecteur colonne q_q. De même, R_{q+1} est obtenue à partir de la matrice $R_q \in \mathbb{R}^{q\times q}$ en ajoutant une ligne et une colonne à R_q. Cette factorisation peut s'écrire sous la forme suivante :

$$\left(\Delta S_q, \Delta s_q\right) = \left(Q_q, q_q\right)\begin{pmatrix} R_q & r_q \\ 0 & \rho_q \end{pmatrix}, \tag{2.35}$$

où r_q est un vecteur de $\mathbb{R}^q$ formé par la dernière colonne de la matrice R_{q+1} où le dernier élément a été enlevé. ρ_q est un scalaire correspondant à cet élément. En développant le membre de droite de l'équation (2.35), on obtient :

$$\left(\Delta S_q, \Delta s_q\right) = \left(Q_q R_q, Q_q r_q + \rho_q q_q\right). \tag{2.36}$$

En considérant la dernière colonne dans chacun des membres, on a :

$$\Delta s_q = Q_q r_q + \rho_q q_q. \tag{2.37}$$

Comme $\Delta S_q = Q_q R_q$, en multipliant de chaque côté de l'équation (2.37) par ΔS_q^T, on obtient :

$$\Delta S_q^T \Delta s_q = R_q^T Q_q^T \left(Q_q r_q + \rho_q q_q\right). \tag{2.38}$$

Comme la matrice Q_{q+1} est orthogonale, en développant l'expression de droite, on aboutit à :

$$\Delta S_q^T \Delta s_q = R_q^T r_q. \tag{2.39}$$

Le système linéaire (2.29) peut donc se simplifier de la façon suivante :

$$\begin{aligned}\left(\Delta S_q^T \Delta S_q\right)\beta = -\Delta S_q^T \Delta s_q &\Longleftrightarrow R_q^T Q_q^T Q_q R_q \beta = -R_q^T r_q \\ &\Longleftrightarrow R_q^T R_q \beta = -R_q^T r_q \\ &\Longleftrightarrow R_q \beta = -r_q.\end{aligned}$$

La dernière équivalence est dûe au fait que R_q est non singulière. Comme la matrice du système linéaire est triangulaire supérieure, la solution est facilement calculée par une méthode de remontée. Après le calcul de β, on calcule les $\gamma_i,\ i = 0, \cdots, q$ à partir de (2.28) et les $\alpha_i,\ i = 0, \cdots, q-1$ à partir de (2.30). Enfin, on calcule l'approximation t_q^{MPE} comme suit :

$$t_q^{MPE} = s_0 + Q_q R_q \alpha. \tag{2.40}$$

Algorithme

La méthode MPE, utilisant l'implémentation présentée précédemment, est donnée dans l'algorithme 11.

Algorithme 11: Algorithme de la méthode MPE

Étape 0. Entrée : les vecteurs $s_k, s_{k+1}, \cdots, s_{k+q+1}$;
Étape 1. Calculer $u_i = \Delta s_i = s_{i+1} - s_i, \quad i = k, k+1, \cdots, k+q$;
Poser $U_j = \left[u_k \,|\, u_{k+1} \,|\, \cdots \,|\, u_{k+j} \right], \ j = 0, 1, \cdots$;
Calculer la factorisation QR de U_{q+1}, notons $U_{q+1} = Q_{q+1} R_{q+1}$;
Étape 2. Résoudre le système linéaire triangulaire supérieur
$R_q d = -r_q$; $r_q = \left[r_{0,q}, r_{1,q}, \cdots, r_{q-1,q} \right]^T$, $d = \left[d_0, d_1, \cdots, d_{q-1} \right]^T$;
Poser $d_q = 1$ et calculer $\lambda = \sum_{i=0}^{q} d_i$;
Poser $\gamma_i = (1/\lambda)\, d_i$ pour $i = 0, \cdots, q$;
Étape 3. Calculer $\alpha = \left[\alpha_0, \alpha_1, \cdots, \alpha_{q-1} \right]^T$ via
$\alpha_0 = 1 - \gamma_0$; $\alpha_j = \alpha_{j-1} - \gamma_j, \ j = 1, \cdots, q-1$;
Calculer $t_{k,q}^{MPE}$ via $t_{k,q}^{MPE} = s_k + Q_q (R_q \alpha)$;

2.3.5 Méthode RRE

Définition

La méthode RRE (Reduced Rank Extrapolation) est définie par Eddy [22] et Mesina [46] dans les années 1970. Cette méthode est obtenue en posant :

$$\eta_{i,j} = (\Delta^2 s_{k+i}, \Delta s_{k+j}). \tag{2.41}$$

Y_q est donc définie par

$$Y_q = \Delta^2 S_{k,q} \tag{2.42}$$

et la transformation de la méthode RRE est obtenue par l'expression suivante :

$$t_{k,q}^{RRE} = s_k - \Delta S_{k,q} \Delta^2 S_{k,q}^{+} \Delta s_k. \tag{2.43}$$

Implémentation

Nous résumons l'implémentation utilisée dans [63]. Utilisons la formulation de l'équation (2.17) en supposant cette fois que les $\gamma_i, \ i = 0, \cdots, q$ sont obtenues en résolvant le système de moindres carrées suivant :

$$\Delta S_{q+1} \gamma = 0, \tag{2.44}$$

sous la contrainte

$$\sum_{j=0}^{q} \gamma_i = 1. \tag{2.45}$$

Cela mène à minimiser la forme quadratique définie positive suivante :

$$\gamma^T \Delta S_{q+1}^T \Delta S_{q+1} \gamma, \tag{2.46}$$

sous la contrainte (2.45). Il a été montré dans [63] que les $\gamma_i,\ i = 0, \cdots, q$ peuvent être obtenues en résolvant le système de $q+2$ équations linéaires suivant :

$$\Delta S_{q+1}^T \Delta S_{q+1} \gamma = \lambda e \qquad\qquad \sum_{j=0}^{q} \gamma_j = 1 \tag{2.47}$$

où $e = (1, \cdots, 1)^T \in \mathbb{R}^{q+1}$ et λ, un scalaire strictement positif vérifiant

$$\lambda = \gamma^T \Delta S_{q+1}^T \Delta S_{q+1} \gamma. \tag{2.48}$$

Le calcul de $\gamma^{(q)}$ se fait donc de la manière suivante :

– on résout

$$\Delta S_{q+1}^T \Delta S_{q+1} d = e, \tag{2.49}$$

avec $d = (d_0, \cdots, d_q)^T$,

– on pose

$$\lambda = \left(\sum_{j=0}^{q} d_j \right)^{-1}, \tag{2.50}$$

– on obtient

$$\gamma = \lambda d. \tag{2.51}$$

Maintenant, supposons que la matrice ΔS_{q+1} soit de rang maximal, c'est-à-dire que $rang(\Delta S_{q+1}) = q+1$. Alors, comme pour la méthode MPE, nous pouvons déterminer la factorisation QR de la matrice $\Delta S_{q+1} = Q_{q+1} R_{q+1}$ L'équation (2.49) peut donc s'écrire de la forme suivante :

$$R_{q+1}^T R_{q+1} d = e. \tag{2.52}$$

Ce système peut être résolu en utilisant une méthode de descente, suivi d'une méthode de remontée. La fin du calcul de t_q^{RRE} est similaire à la méthode MPE.

Algorithme 12: Algorithme de la méthode RRE

Étape 0. Entrée : les vecteurs $s_k, s_{k+1}, \cdots, s_{k+q+1}$;
Étape 1. Calculer $u_i = \Delta s_i = s_{i+1} - s_i, \quad i = k, k+1, \cdots, k+q$;
Poser $U_j = \left[u_k \,|\, u_{k+1} \,|\, \cdots \,|\, u_{k+j} \right], \ j = 0, 1, \cdots$;
Calculer la factorisation QR de U_{q+1}, notons $U_{q+1} = Q_{q+1} R_{q+1}$;
Étape 2. Résoudre le système linéaire
$R_{q+1}^T R_{q+1} d = e$; $d = \left[d_0, d_1, \cdots, d_q \right]^T$, $e = [1, 1, \cdots, 1]^T$;
Poser $\lambda = \sum_{i=0}^{q} d_i$;
Poser $\gamma_i = (1/\lambda)\, d_i$ pour $i = 0, \cdots, q$;
Étape 3. Calculer $\alpha = \left[\alpha_0, \alpha_1, \cdots, \alpha_{q-1} \right]^T$ via
$\alpha_0 = 1 - \gamma_0$; $\alpha_j = \alpha_{j-1} - \gamma_j$, $j = 1, \cdots, q-1$;
Calculer $t_{k,q}^{RRE}$ via $t_{k,q}^{RRE} = s_k + Q_q (R_q \alpha)$;

Algorithme

Des algorithmes rapides, stables et utilisant peu de ressources mémoires sont décrits dans [37, 65, 63]. L'implémentation de la méthode RRE est résumée dans l'algorithme 12.

2.3.6 Méthode MMPE

Définition

La méthode MMPE (Modified Minimal Polynomial Extrapolation) a été définie par Brezinski [10], Pugatchev [51] et Sidi, Ford et Smith [65]. Cette fois-ci, on choisit Y_q de telle sorte que les colonnes $\{y_0, \cdots, y_{q-1}\}$ forment un ensemble de vecteurs linéairement indépendants de $\mathbb{R}^N$. Pour cela, on préfèrera ne pas utiliser la décomposition QR de la matrice Y_q, mais la décomposition PLU.

Implémentation

Nous résumons la méthode décrite dans [38]. Supposons que $rang(\Delta S_{q+1}) = q+1$. D'après le théorème 1.3, il existe une matrice de permutation $P \in \mathbb{R}^{N \times N}$, une matrice trapézoïdale inférieure unitaire $L_{q+1} \in \mathbb{R}^{N \times (q+1)}$ et une matrice triangulaire supérieure inversible $R_{q+1} \in \mathbb{R}^{(q+1) \times (q+1)}$ telles que :

$$P \Delta S_{q+1} = L_{q+1} R_{q+1}. \tag{2.53}$$

Posons P_{q+1}, la matrice formée par les $q+1$ premières lignes de P et L'_{q+1}, la matrice formée par les $q+1$ premières lignes de L_{q+1}. Alors, à partir de (2.53), nous obtenons

la factorisation suivante :

$$P_{q+1}\Delta S_{q+1} = L'_{q+1}R_{q+1}, \tag{2.54}$$

qui peut aussi s'écrire comme suit :

$$P_{q+1}\left(\Delta S_q, \Delta s_q\right) = \begin{pmatrix} L'_q & 0 \\ v_q^T & 1 \end{pmatrix}\begin{pmatrix} R_q & r_q \\ 0 & \rho_q \end{pmatrix}, \tag{2.55}$$

où r_q et v_q sont deux vecteurs de $\mathbb{R}^q$ et ρ_q, un scalaire. r_q correspond aux q premiers éléments de la dernière colonne de R_{q+1}. Développons maintenant la dernière équation, nous obtenons :

$$\left(P_{q+1}\Delta S_q, P_{q+1}\Delta s_q\right) = \begin{pmatrix} L'_qR_q & L'_qr_q \\ v_q^TR_q & v_q^Tr_q + \rho_q \end{pmatrix}. \tag{2.56}$$

En comparant les dernières colonnes des deux matrices de (2.56), il suit que :

$$\left(P_{q+1}\Delta s_q\right) = \begin{pmatrix} L'_qr_q \\ v_q^Tr_q + \rho_q \end{pmatrix}. \tag{2.57}$$

Mais, comme P_q est la sous matrice de P_{q+1} de taille $q \times N$ dont les lignes sont formées par les q premières lignes de P_{q+1}, nous obtenons

$$P_q\Delta s_q = L'_qr_q. \tag{2.58}$$

Maintenant, en posant $Y_q = P_q^T$ et en utilisant la factorisation $P_q\Delta S_q = L'_qR_q$, le système linéaire (2.29) peut s'écrire comme suit :

$$L'_qR_q\beta = -P_q\Delta s_q. \tag{2.59}$$

En utilisant (2.58) et le fait que la matrice L'_q est non singulière, (2.59) devient :

$$R_q\beta = -r_q. \tag{2.60}$$

Comme la matrice du système linéaire est triangulaire supérieure, la solution est facilement calculable par une méthode de remontée. Après le calcul de β, on calcule γ à partir de (2.28) et α à partir de (2.30). Enfin, on calcule l'approximation t_q^{MMPE} comme suit :

$$t_q^{MMPE} = s_0 + P_q^TL'_qR_q\alpha. \tag{2.61}$$

Algorithme 13: Algorithme de la méthode MMPE

Étape 0. Entrée : les vecteurs $s_k, s_{k+1}, \cdots, s_{k+q+1}$, $p = [1, \cdots, N]$;

Étape 1. Poser $l^1 = \Delta s_0$;

Trouver i_0 tel que $|(l^1)_{i_0}| = \|l^1\|_\infty$;

$p(i) \leftrightarrow p(1)$;

$r_{11} = (l^1)_{i_0}$; $l^1 = \frac{l^1}{r_{11}}$;

Étape 2.

pour $j = 1, \cdots, q$ **faire**

Poser $u = \Delta s_j$;

pour $l = 1, \cdots, j$ **faire**

$c = (u)_{p(l)}$; $r_{lj+1} = c$; $(u)_{p(l)} = 0$;

$(u)_{p(l+1:N)} = (u)_{p(l+1:N)} - c \times (l^j)_{p(l+1:N)}$;

fin

Déterminer i_0 tel que $|(u)_{i_0}| = \|u\|_\infty$;

$p(j+1) \leftrightarrow p(i_0)$; $r_{j+1j+1} = (u)_{i_0}$; $l^{j+1} = u/(u_{i_0})$;

fin

Étape 3. Résoudre le système linéaire triangulaire supérieur

$R_q d = -r_q$; $r_q = \left[r_{0,q}, r_{1,q}, \cdots, r_{q-1,q}\right]^T$, $d = \left[d_0, d_1, \cdots, d_{q-1}\right]^T$;

Poser $d_q = 1$ et calculer $\lambda = \sum_{i=0}^{q} d_i$;

Poser $\gamma_i = (1/\lambda)\, d_i$ pour $i = 0, \cdots, q$;

Étape 4. Calculer $\alpha = \left[\alpha_0, \alpha_1, \cdots, \alpha_{q-1}\right]^T$ via

$\alpha_0 = 1 - \gamma_0$; $\alpha_j = \alpha_{j-1} - \gamma_j$, $j = 1, \cdots, q-1$;

Calculer $t_{k,q}^{MMPE}$ via $t_{k,q}^{MMPE} = s_k + P_q^T L_q'(R_q \alpha)$;

Algorithme

L'implémentation de la méthode MMPE est donnée dans l'algorithme 13. On peut noter que les éléments nuls de $(l_j)_{p(l)}$ peuvent être remplacés par $r_{l,j}$. Du coup, la méthode MMPE requiert moins de stockage que les méthodes RRE et MPE. Dans l'algorithme, le symbole $\leftrightarrow$ désigne l'échange des données : $x \leftrightarrow y \Leftrightarrow t = x,\ x = y,\ y = t$.

2.3.7 Exemple

Exemple 6. *Nous considérons dans cet exemple, l'équation de Bratu [12] définie par :*

$$-\nabla u + \alpha \partial_x u + \lambda e^u = \phi \tag{2.62}$$

sur le carré unité de $\mathbb{R}^2$ avec des conditions aux bords de Dirichlet et avec α et λ des entiers positifs. Ce problème est discrétisé par une méthode de différences finies sur un maillage uniforme. L'équation (2.62) discrétisée est donnée sous la forme suivante :

$$AX + \lambda e^X - b = 0. \tag{2.63}$$

Le vecteur b est choisi de telle sorte que la solution de (2.63) soit égale au vecteur unité. La suite $\{s_k\}$ est obtenue en utilisant la méthode SSOR non linéaire. On décompose la matrice A sous la forme $D - L - U$ avec D, L et U définies par une méthode classique de décomposition. On pose $s_{k+1} = G(s_k)$, où

$$G(X) = B_\omega X + \omega(2-\omega)(D-\omega U)^{-1} D (D-\omega L)^{-1}(b - \lambda e^X), \tag{2.64}$$

et la matrice B_ω est donnée par :

$$B_\omega = (D-\omega U)^{-1}(\omega L + (1-\omega)D)(D-\omega L)^{-1}(\omega U + (1-\omega)D). \tag{2.65}$$

Dans les tests qui suivent, on compare les trois méthodes d'extrapolation. Le critère d'arrêt est défini par

$$\|F(s_k)\| < 10^{-7} \tag{2.66}$$

ou si le nombre d'itérations atteint 150. *On choisit $N = 900$, $\alpha = 10$ et $\omega = 1$. Nous allons comparer les résultats en faisant varier λ.*

Les tableaux 2.3, 2.4 et 2.5 montrent le nombre d'itérations, la norme du résidu et le temps CPU de chaque programme pour λ variant de 1 à 3. On remarque que les méthodes MPE et RRE convergent avec le même nombre d'itérations et un temps d'exécution comparable. La méthode MMPE est pratiquement similaire. À partir de $\lambda = 3$, les méthodes ne convergent plus aussi rapidement. On remarque aussi que le temps d'exécution peut vite augmenter en fonction du nombre d'itérations. On définit donc des méthodes redémarrées ou cycliques.

	MPE	RRE	MMPE
Nombre d'itérations	11	11	12
Norme du résidu	7.04210^{-8}	6.82910^{-8}	2.56910^{-8}
Temps CPU	0.3	0.24	0.33

TABLE 2.3 – Méthodes d'extrapolation appliquées à l'équation de Bratu ($\lambda = 1$)

	MPE	RRE	MMPE
Nombre d'itérations	16	16	18
Norme du résidu	6.84110^{-8}	6.64210^{-8}	3.42210^{-8}
Temps CPU	0.36	0.35	0.5

TABLE 2.4 – Méthodes d'extrapolation appliquées à l'équation de Bratu ($\lambda = 2$)

2.3.8 Méthodes redémarrées ou Méthodes cycliques

Les méthodes données dans les algorithmes 11, 12 et 13 deviennent très coûteuses lorsque q augmente. Pour les méthodes MPE et RRE, le nombre de calculs requis augmente de façon quadratique avec le nombre d'itérations q et le coût de stockage augmente linéairement. Une bonne méthode pour garder le coût de stockage et le coût des calculs les plus bas possibles est de redémarrer ces algorithmes périodiquement. L'algorithme 14 décrit la méthode redémarrée.

Exemple 7. *Reprenons l'équation de l'exemple précédent. On a vu que lorsque le paramètre λ augmentait, le nombre d'itérations nécessaire pour résoudre cette équation augmentait aussi. Nous allons donc utiliser le processus redémarré. Nous appliquons nos méthodes d'extrapolation dans le cas où $\lambda = 3$, et nous faisons varier le nombre d'étapes pour redémarrer nos méthodes. Le tableau 2.6 présente les résultats de convergence.*

On remarque que dans cet exemple, les méthodes cycliques permettent de réduire le nombre total d'itérations et le temps d'exécution des méthodes. On voit aussi que le comportement de la convergence est différent selon les méthodes. En effet, la méthode MPE converge le plus rapidement dans le cas du redémarrage avec $q = 12$ en 50 itérations. La méthode RRE converge le mieux lorsque $q = 11$ en 53 itérations. Enfin, la méthode MMPE converge en 55 itérations avec $q = 6$.

	MPE	RRE	MMPE
Nombre d'itérations	150	150	150
Norme du résidu	2.87410^{-6}	4.1410^{-6}	1.04310^{-5}
Temps CPU	7.36	7.42	10.26

TABLE 2.5 – Méthodes d'extrapolation appliquées à l'équation de Bratu ($\lambda = 3$)

Algorithme 14: Méthode redémarrée toutes les q étapes (Procédé cyclique)

Étape 0. Entrée : Poser $k = 0$, choisir un entier q et un vecteur s_0;
Étape 1. Former la suite $s_1, \cdots, s_{q+1}$;
Étape 2. Calculer l'approximation t_q en utilisant les algorithmes correspondants;
Étape 3. Si t_q est satisfaisant, stop ;
Sinon, poser $s_0 = t_q$, $k = k+1$ et retourner à l'étape 1;

q		MPE	RRE	MMPE
6	Nombre de cycles	13	28	10
	Nombre d'itérations	73	164	*55*
	Temps CPU	1.99	4.29	1.61
7	Nombre de cycles	12	30	9
	Nombre d'itérations	78	205	61
	Temps CPU	2.03	5.2	1.78
8	Nombre de cycles	8	8	11
	Nombre d'itérations	64	63	88
	Temps CPU	1.29	1.37	2.41
9	Nombre de cycles	7	8	8
	Nombre d'itérations	58	67	71
	Temps CPU	1.33	1.56	1.88
10	Nombre de cycles	5	7	11
	Nombre d'itérations	50	67	103
	Temps CPU	1.13	1.53	2.74
11	Nombre de cycles	5	5	7
	Nombre d'itérations	52	*53*	72
	Temps CPU	1.13	1.22	1.71
12	Nombre de cycles	5	6	9
	Nombre d'itérations	*50*	61	106
	Temps CPU	1.10	1.34	2.61
13	Nombre de cycles	5	7	8
	Nombre d'itérations	58	90	95
	Temps CPU	1.22	1.86	2.35
14	Nombre de cycles	5	7	5
	Nombre d'itérations	65	98	67
	Temps CPU	1.35	1.95	1.65

TABLE 2.6 – Méthodes d'extrapolation redémarrées appliquées à l'équation de Bratu ($\lambda = 3$)

Méthodes	MPE	RRE	MMPE
Multiplications et Additions	nq^2	nq^2	nq
Stockage	$q+1$	$q+1$	$q+1$

TABLE 2.7 – Coût des différentes méthodes d'extrapolation

2.3.9 Coût des méthodes

Dans cette section, nous donnons le nombre d'opérations et le stockage nécessaire à la réalisation de nos méthodes d'extrapolation. Tous ces coûts sont donnés dans le tableau 2.7 On ne met dans ce tableau que les nombres d'opérations principaux sachant que n est très grand par rapport à q. On remarque que la méthode qui nécessite le moins d'opérations est la méthode MMPE. En effet cette méthode nécessite nq^2 opérations due à l'utilisation de la décomposition LU. Les méthodes MPE et RRE sont comparable.

Le stockage nécessaire aux trois méthodes est identique. On ne doit stocker que $q+1$ vecteurs de tailles n correspond aux vecteurs s_0 et aux q premières colonnes de la matrice Q_{q+1}. Le nombre d'évaluations de fonctions nécessaires aux trois méthodes est de $q+1$.

Dans la suite de ce chapitre, nous regardons plus en détails comment se comportent les méthodes d'extrapolation lorsqu'elles sont appliquées à la résolution de systèmes non linéaires et linéaires. Commençons par le premier le cas.

2.4 Application à la résolution de systèmes non linéaires

Considérons le système d'équations non linéaires suivant :

$$F(x) = 0, \tag{2.67}$$

défini par la fonction non linéaire $F : \mathbb{R}^N \to \mathbb{R}^N$. On note s_*, la solution de ce système. Soit s_0, une approximation initiale de s_*. On génère la suite $s_1, s_2, \cdots$ par une méthode de point fixe, disons

$$s_{k+1} = G(s_k),\ k = 0, 1, 2, \cdots, \tag{2.68}$$

où $G : \mathbb{R}^N \to \mathbb{R}^N$, vérifiant $x - G(x) = 0$, peut être une forme préconditionnée de (2.67) possédant comme solution s_* et de sorte que $\lim_{k\to\infty} s_k = s_*$.

Dans cette section, nous allons donner des résultats de convergence de la méthode RRE, puis nous donnerons des exemples de résolution de systèmes non linéaires en comparant les méthodes d'extrapolation aux méthodes de Newton et Newton inexactes.

2.4.1 Théorèmes de convergence

Posons $x_0 = s_0$ et définissons la suite $\{x_k\}$ en considérant l'algorithme 14 appliqué à la méthode RRE avec $q = q_k$, en posant à l'itération k, $x_{k+1} = t_{q_k}^{RRE}$. On définit q_k comme étant le degré du polynôme minimal de $G'(s_*)$ pour le vecteur $x_k - s_*$, c'est-à-dire, le degré du polynôme P de degré minimal tel que $P(G'(s_*))(x_k - s_*) = 0$. Dans [39], Jbilou et Sadok ont démontré la convergence quadratique de la méthode RRE. Nous rappelons ce théorème. Commençons par définir quelques notations utiles :

$$\alpha_{q_k}(x) = \sqrt{det(H_{q_k}^*(x) H_{q_k}(x))}, \tag{2.69}$$

où

$$H_{q_k}(x) = \left(\frac{G(x) - x}{\|G(x) - x\|}, \cdots, \frac{G^{q_k}(x) - G^{q_k-1}(x)}{\|G^{q_k}(x) - G^{q_k-1}(x)\|} \right). \tag{2.70}$$

Théorème 2.1

Soit $J = G'(s_*)$. Supposons que la matrice $J-I$ soit régulière. Posons $M = \|(J-I)^{-1}\|$. Supposons que la dérivée de Fréchet G' satisfasse les conditions de Lipschitz

$$\|G'(x) - G'(y)\| \leq L\|x - y\|, \ \forall x, y \in \mathscr{D}, \tag{2.71}$$

où $\mathscr{D}$ est un sous espace ouvert et convexe de $\mathbb{C}^p$. Si

$$\exists \alpha > 0, \ \exists K, \ \forall k \geq K \ : \ \alpha_{q_k}(x_k) > \alpha, \tag{2.72}$$

alors il existe un voisinage U de s_* tel que, pour tout $s_0 \in U$,

$$\|x_{k+1} - s_*\| \leq K\|x_k - s_*\|^2. \tag{2.73}$$

Dans ce théorème, q_k dépend de s_*. Ce qui, dans la pratique, est très gênant. Pour remédier à ce problème, on propose de remplacer q_k par un entier fixé et petit. Une autre possibilité est de remplacer q_k par l_k, où l_k vérifie

$$\|\tilde{r}(t_{l_k}^{RRE})\| = 0. \tag{2.74}$$

Ce changement permet d'obtenir la même convergence quadratique et est plus intéressant dans la pratique. On considère l'algorithme 14 avec $q = l_k$, où l_k vérifie (2.74).

Théorème 2.2

Sous les mêmes hypothèses que le théorème 2.1. Si

$$\exists \alpha > 0, \ \exists K, \ \forall k \geq K \ : \ \alpha_{l_k}(x_k) > \alpha, \tag{2.75}$$

alors il existe un voisinage U de s_* tel que, pour tout $s_0 \in U$,

$$\|x_{k+1} - s_*\| \leq K\|x_k - s_*\|^2. \tag{2.76}$$

Preuve 2.1. *Posons $F(x) = G(x) - x$ et $\phi(x) = F(x) - F'(s_*)(x - s_*)$. Nous avons*

$$\|\phi(x)\| \leq \frac{1}{2}L\|x - s_*\|^2, \tag{2.77}$$

$$\|\phi(x) - \phi(y)\| \leq L\|x - y\| \max\{\|x - s_*\|, \|y - s_*\|\}, \tag{2.78}$$

$\forall\, x, y \in \mathscr{D}$. La preuve de ces deux inégalités se trouve dans [49].

Nous avons $\Delta s_0 = G(x_k) - x_k$, $\Delta s_1 = G(G(x_k)) - G(x_k)$ et $\Delta^2 s_0 = F(G(x_k)) - F(x_k)$. Ainsi, pour $i = 0, \cdots, l_k$, nous obtenons

$$\Delta s_i = G^{i+1}(x_k) - G^i(x_k)$$

$$\Delta^2 s_i = F(G^{i+1}(x_k)) - F(G^i(x_k))$$

Soit

$$C_{l_k} = \Delta^2 S_{l_k} - F'(s_*)\Delta S_{l_k}. \tag{2.79}$$

Alors C_{l_k} peut être écrite comme suit :

$$C_{l_k} = \left(\phi(G(x_k)) - \phi(x_k), \cdots, \phi(G^{l_k}(x_k)) - \phi(G^{l_k-1}(x_k))\right).$$

Si nous définissons

$$\tilde{C}_{l_k} = \left(\frac{\phi(G(x_k)) - \phi(x_k)}{\|G(x_k) - x_k\|}, \cdots, \frac{\phi(G^{l_k}(x_k)) - \phi(G^{l_k-1}(x_k))}{\|G^{l_k}(x_k) - G^{l_k-1}(x_k)\|}\right)$$

et

$$B_{l_k} = F'(s_*)^{-1} C_{l_k} \Delta S^+_{l_k}, \tag{2.80}$$

nous obtenons

$$B_{l_k} = F'(s_*)^{-1} \tilde{C}_{l_k} H^+_{l_k}(x_k).$$

Cela implique que

$$\|B_{l_k}\| \leq M\|\tilde{C}_{l_k}\|_F \left\|H^+_{l_k}(x_k)\right\|.$$

En utilisant le lemme 1, nous avons

$$\left\| H_{l_k}^{+}(x_k) \right\| \leq \frac{\left\| H_{l_k}^{+}(x_k) \right\|_F^{l_k-1}}{\alpha_{l_k}(x_k)}. \tag{2.81}$$

Maintenant, comme

$$\left\| \tilde{C}_{l_k} \right\|_F^2 = \sum_{i=1}^{l_k} \left\| \frac{\phi(G^i(x_k)) - \phi(G^{i-1}(x_k))}{\left\| G^i(x_k) - G^{i-1}(x_k) \right\|} \right\|^2,$$

il suit d'après (2.78) que

$$\left\| \tilde{C}_{l_k} \right\|_F \leq \sqrt{l_k} L \max_{0 \leq i \leq l_k} \left\| G^i(x_k) - s_* \right\|. \tag{2.82}$$

À partir de (2.81) et (2.82), nous obtenons l'inégalité suivante :

$$\left\| B_{l_k} \right\| \leq \tau_{l_k}(x_k),$$

où $\tau_{l_k}(x) = \frac{l_k^{l/2}}{\alpha_{l_k}(x)} ML \max_{0 \leq i \leq l_k} \left\| G^i(x) - s_* \right\|$. *Il existe donc un voisinage de* s_* *tel que* $\left\| B_{l_k} \right\| \leq 1$. *Dans ce cas,* $I + B_{l_k}$ *est régulière. Posons* $D_{l_k} = I - (I + B_{l_k})^{-1}$. *Revenons à l'algorithme 14 :*

$$x_{k+1} = x_k - \Delta S_{l_k} \Delta^2 S_{l_k}^{+} \Delta s_0.$$

Maintenant, à partir de (2.79) et (2.80), nous en déduisons que :

$$\Delta S_{l_k} = (I + B_{l_k})^{-1} F'(s_*)^{-1} \Delta^2 S_{l_k}.$$

Donc

$$x_{k+1} = x_k - (I - D_{l_k}) F'(s_*)^{-1} \Delta^2 S_{l_k} \Delta^2 S_{l_k}^{+} \Delta s_0. \tag{2.83}$$

Nous avons supposé que $\|\tilde{r}(t_{l_k}^{RRE})\| = 0$. *Cela implique que* $\|\Delta s_0 - \Delta^2 S_{l_k} \Delta^2 S_{l_k}^{+} \Delta s_0\| = 0$. *Il suit que*

$$\Delta s_0 = \Delta^2 S_{l_k} \Delta^2 S_{l_k}^{+} \Delta s_0. \tag{2.84}$$

En utilisant (2.84), (2.83) devient :

$$\begin{aligned} x_{k+1} &= x_k - (I - D_{l_k}) F'(s_*) \Delta s_0 \\ &= x_k - (I - D_{l_k}) F'(s_*) F(x_k) \end{aligned}$$

et

$$x_{k+1} - s_* = (D_{l_k} - I) F'(s_*)^{-1} \phi(x_k) + D_{l_k}(x_k - s_*).$$

D'où

$$\|x_{k+1} - s_*\| \leq \left\|(D_{l_k} - I)\right\| M \left\|\phi(x_k)\right\| + \left\|D_{l_k}\right\| \|(x_k - s_*)\|. \tag{2.85}$$

Mais

$$\left\|D_{l_k}\right\| \leq \frac{\left\|B_{l_k}\right\|}{1 - \left\|B_{l_k}\right\|} \text{ et } \left\|(D_{l_k} - I)\right\| \leq \frac{1}{1 - \left\|B_{l_k}\right\|}.$$

À partir de la définition de $\tau_{l_k}(x)$ *et de l'hypothèse, nous obtenons :*

$$\tau_{l_k}(x_k) \leq \frac{MLp^{p/2}M_2}{\alpha} \|x_k - s_*\|. \tag{2.86}$$

Alors $\lim_{k\to\infty} \tau_{l_k}(x_k) = 0$. *Cela implique qu'il existe un entier* M_3 *tel que*

$$\left|\frac{1}{1 - \tau_{l_k}(x_k)}\right| < M_3, \text{ pour } k > N_0. \tag{2.87}$$

En utilisant (2.77),(2.86) et (2.87), (2.85) devient :

$$\begin{array}{rcl} \|x_{k+1} - s_*\| & \leq & \frac{ML}{2(1-\tau_{l_k}(x_k))} \|x_k - s_*\|^2 + \frac{\tau_{l_k}(x_k)}{1-\tau_{l_k}(x_k)} \|x_k - s_*\| \\ & \leq & \frac{MLM_3}{2} \|x_k - s_*\|^2 + \frac{M_3 MLN^{N/2}M_2}{\alpha} \|x_k - s_*\|^2 \end{array}.$$

Finalement, on obtient :

$$\|x_{k+1} - s_*\| = O(\|x_k - s_*\|^2). \tag{2.88}$$

□

2.4.2 Résultats numériques

Dans cette section, nous allons comparer les méthodes d'extrapolation avec la méthode de Newton et les méthodes de Newton inexactes que l'on a rappelées dans le chapitre 1. On considère l'équation de Chandrasekhar définie dans l'exemple 1.

Convergence quadratique

On montre la convergence quadratique de la méthode RRE cyclique pour cet exemple. On pose $c = 0.99$ et $N = 500$. Le tableau 2.8 montre la norme de l'erreur et le valeur de l_k obtenus lorsque $\|\tilde{r}(t_{l_k})\| < 10^{-6}$. On observe bien la convergence quadratique de la méthode.

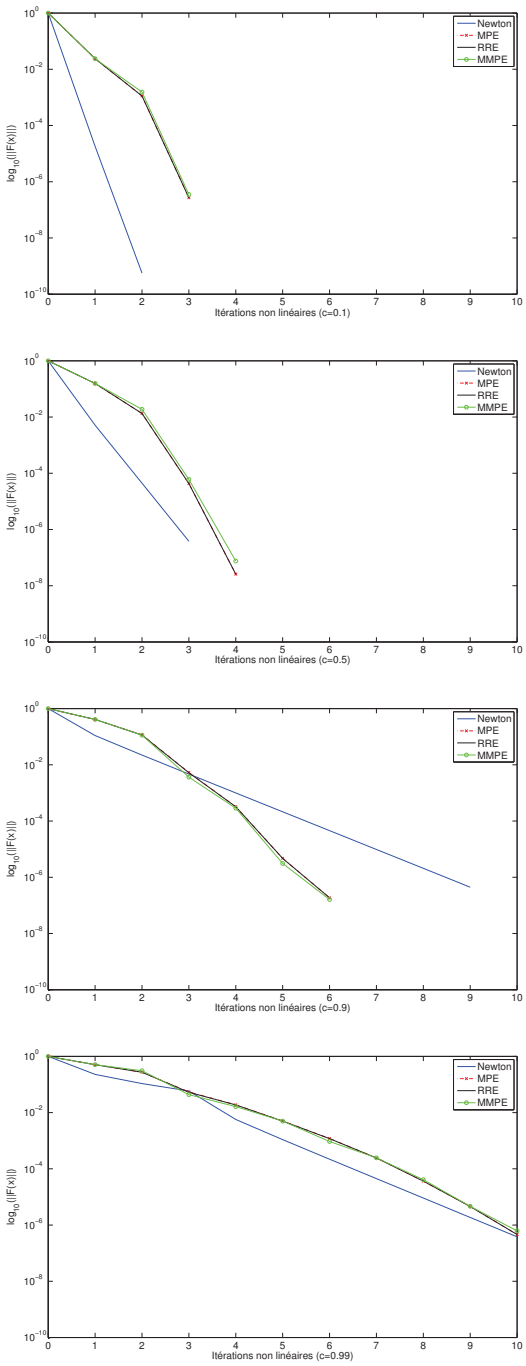


FIGURE 2.3 – Méthodes d'extrapolation et de Newton appliquées à l'équation de Chandrasekhar

k	$\|x_k - s_*\|$	l_k
0	20.46	
1	8.210^{-3}	6
2	1.210^{-5}	3

TABLE 2.8 – Convergence quadratique de la méthode RRE pour l'équation de Chandrasekhar

Comparaison méthode de Newton/méthodes d'extrapolation

La figure 2.3 montre la convergence des méthodes de Newton et d'extrapolation pour différentes valeurs de c. La figure en haut à gauche montre les résultats pour $c = 0.1$, en haut à droite pour $c = 0.5$, en bas à gauche pour $c = 0.9$ et en bas à droite pour $c = 0.99$. On choisit le nombre de points égal à 500.

Comparaison méthode Newton-GMRES /méthode RRE

Nous allons comparer la convergence des méthodes RRE et Newton-GMRES (cf. Chapitre 1) pour l'exemple de l'équation de Chandrasekhar.

Les courbes de la figure 2.4 montrent les résultats obtenus pour les mêmes valeurs de c que précédemment. Les courbes étant similaires pour les méthodes d'extrapolation MPE et MMPE, nous ne montrons ici que la courbe de la méthode RRE. Les courbes de gauche montrent la convergence des méthodes RRE et Newton-GMRES en fonction du nombre d'itérations tandis que les courbes de droite montrent la convergence en fonction du nombre d'évaluations de fonction. On voit très clairement que, à part pour $c = 0.5$, la méthode Newton-GMRES converge plus rapidement en nombre d'itérations mais moins rapidement en nombre d'évaluations de fonction.

2.5 Application à la résolution de systèmes linéaires

On s'intéresse, dans cette partie, à la résolution du système d'équations linéaires suivant :

$$Ax = b, \tag{2.89}$$

où $A \in \mathbb{R}^{N\times N}$ est une matrice inversible, x et $b \in \mathbb{R}^N$.

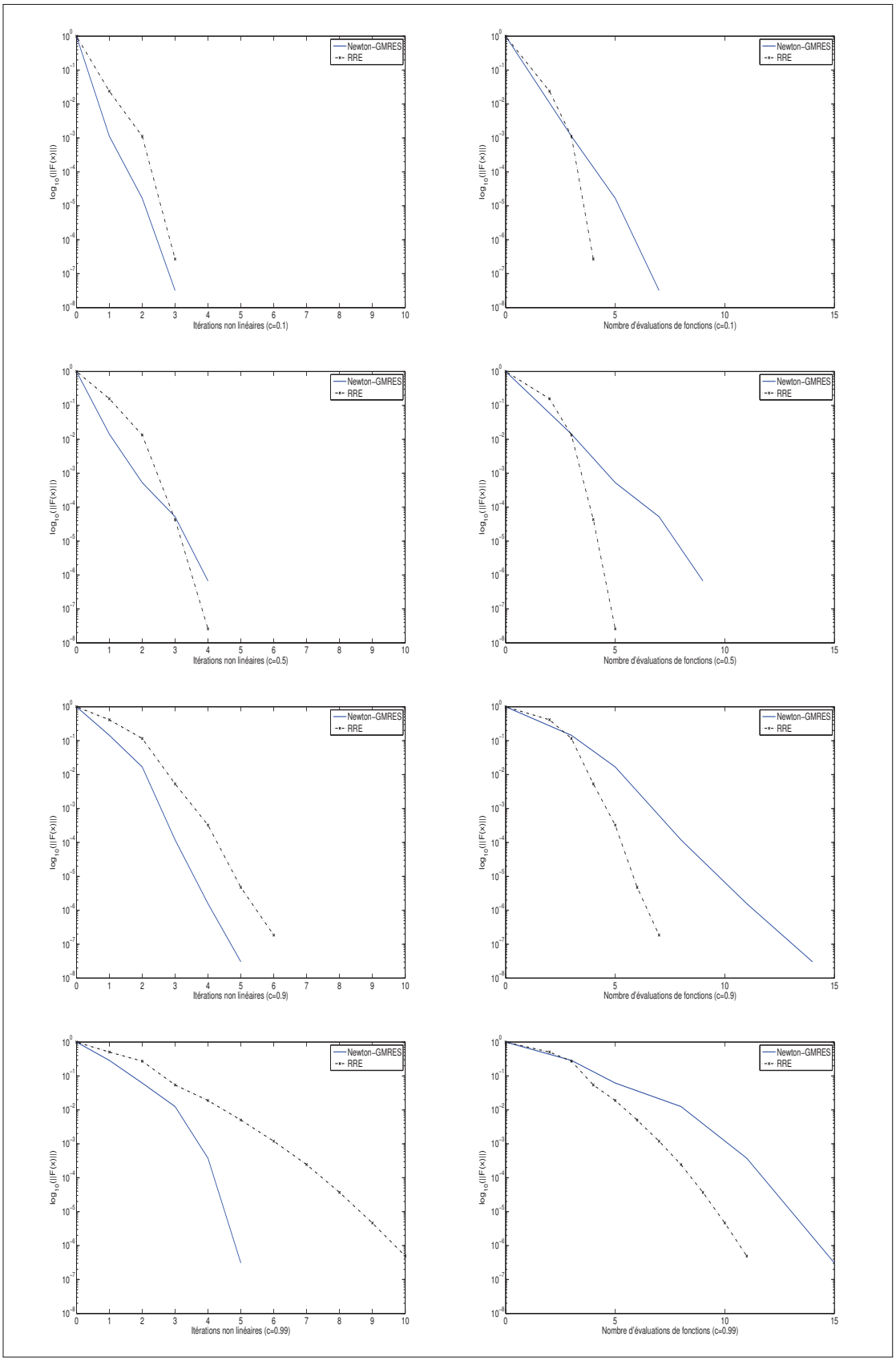


FIGURE 2.4 – Méthodes RRE et Newton-GMRES appliquées à l'équation de Chandrasekhar

2.5.1 Application directe de l'extrapolation vectorielle à la résolution de systèmes linéaires

Soit s_0, une approximation initiale de la solution (2.89). On définit la suite $\{s_k\}$ par la formule suivante :

$$s_{k+1} = Bs_k + b, \tag{2.90}$$

où $B = I_N - A$. Si la suite $\{s_k\}$ est convergente, alors sa limite s_* est la solution du système (2.89). Pour tout vecteur x, on définit le résidu $r(x) = b - Ax$. Alors, pour tout k, on a :

$$r(s_k) = b - As_k = b + Bs_k - s_k = \Delta s_k. \tag{2.91}$$

De plus, en utilisant (2.23) et (2.90), on obtient que le résidu généralisé de $t_{k,q}$ correspond au vrai résidu :

$$\tilde{r}(t_{k,q}) = r(t_{k,q}) = b - At_{k,q}. \tag{2.92}$$

On peut voir que $\Delta^2 s_k = -A\Delta s_k$, et donc $\Delta^2 S_{k,q} = -A\Delta S_{k,q}$.

Pour simplifier les notations, on utilise le même procédé que pour la définition des algorithmes. On pose $k = 0$. Soit d, le degré du polynôme minimal de B pour le vecteur $s_0 - s_*$. Comme B est inversible, le polynôme minimal de B pour le vecteur $s_0 - s_*$ est le même que le polynôme minimal de B pour $r_0 = \Delta s_0$. Les matrices ΔS_q et $\Delta^2 S_q$ sont donc de rang maximal lorsque $q \leq d$ et t_q existe et correspond à la solution de (2.89).

Lorsqu'elles sont appliquées à des suites générées par (2.90), nos méthodes d'extrapolation forment une approximation t_q de telle sorte que le résidu correspondant $r_q = b - At_q$ satisfasse les relations suivantes :

$$r_q \in W_q = AV_q \tag{2.93}$$

et

$$r_q \perp L_q, \tag{2.94}$$

où $V_q = span\{\Delta s_0, \cdots, \Delta s_{q-1}\}$ et L_q dépend de la méthode utilisée à savoir :

- $L_q \equiv W_q$ pour la méthode RRE,
- $L_q \equiv V_q$ pour la méthode MPE,
- $L_q \equiv Y_q$ pour la méthode MMPE.

On peut remarquer que $W_q \equiv K_q(A, Ar_0)$. Ce qui prouve que les méthodes d'extrapolation étudiées ici, sont comparables à des méthodes de sous espaces de Krylov. Il a été montré dans [64] que la méthode RRE est une méthode de projection orthogonale et est équivalente à la méthode GMRES tandis que les méthode MPE et MMPE sont des méthodes de projection oblique et sont équivalentes à la méthode d'Arnoldi et à la méthode de Hessenberg, respectivement. À partir de ce constat, on peut conclure que pour $q \leq d$, l'approximation t_q existe et est unique pour la méthode RRE, ce qui n'est pas le cas pour la méthode MPE et MMPE.

Nous continuons en donnant quelques résultats de comparaison des résidus pour ces méthodes. Soit P_q, le projeté orthogonal sur W_q. Alors, d'après (2.93) et (2.94), le résidu généré par la méthode RRE peut s'exprimer comme suit :

$$r_q^{RRE} = r_0 - P_q r_0. \tag{2.95}$$

De même, soient Q_q et R_q, les projecteurs obliques sur W_q orthogonaux à V_q et Y_q respectivement. On peut donc écrire les résidus obtenus par les méthodes MPE et MMPE comme suit :

$$r_q^{MPE} = r_0 - Q_q r_0 \tag{2.96}$$

et

$$r_q^{MMPE} = r_0 - R_q r_0. \tag{2.97}$$

On note θ_q, l'angle aigu entre r_0 et le sous espace W_q. Celui-ci est défini par

$$\cos\theta_q = \max_{z \in W_q - \{0\}} \left(\frac{|(r_0, z)|}{\|r_0\| \|z\|} \right). \tag{2.98}$$

Théorème 2.3

Soient Φ_q, l'angle aigu entre r_0 et $Q_q r_0$ et ψ_q, l'angle aigu entre r_0 et $R_q r_0$. De plus, pour la méthode MMPE, on pose $y_j = r_0$, pour $j = 1, \cdots, q$. On a les relations suivantes :

1. $\|r_q^{RRE}\|^2 = (\sin^2\theta_q)\|r_0\|^2$,
2. $\|r_q^{MPE}\|^2 = (\tan^2\Phi_q)\|r_0\|^2$,
3. $\|r_q^{RRE}\| \leq (\cos\Phi_q)\|r_q^{MPE}\|$,
4. $\|r_q^{MMPE}\|^2 = (\tan^2\psi_q)\|r_0\|^2$,
5. $\|r_q^{RRE}\| \leq (\cos\psi_q)\|r_q^{MMPE}\|$.

La preuve de ce théorème se trouve dans [40]. On peut remarquer que, grâce aux relations (1), (2) et (4) du théorème précédent, on prouve d'une autre façon que la méthode RRE existe toujours alors que les méthodes MPE et MMPE peuvent ne pas exister. On observe aussi que, si une stagnation se produit dans les itérations de la méthode RRE, c'est-à-dire que $\|r_q^{RRE}\| = \|r_0\|$ pour $q \leq d$, alors $\cos\theta_q = 0$ et grâce à (2.98), cela implique que $\cos\Phi_q = \cos\psi_q = 0$ et que les approximations produites par les méthodes MPE et MMPE ne sont pas définies.

2.5.2 Méthode CMRH

Nous supposons dans cette section que la matrice A est dense. On a vu dans la section précédente que la méthode MMPE était équivalente à la méthode de Hessenberg lorsque celle-ci est appliquée à la résolution d'un système linéaire. Dans [59], Sadok développe une méthode utilisant la méthode de Hessenberg pour résoudre des systèmes d'équations linéaires : la méthode CMRH (Changing Minimal Residual method based on the Hessenberg process). Cette méthode est une alternative à la méthode GMRES. Il a été montré que cette méthode requiert moins de stockage et moins d'opérations que la méthode GMRES, tout en ayant la même efficacité. Dans ce chapitre, nous donnons la définition de la méthode CMRH, son algorithme et quelques résultats de convergence. Dans le chapitre 6, nous poursuivrons l'étude de cette méthode en considérant le préconditionnement et l'implémentation sur des machines parallèles.

Définition

Nous donnons une description de la méthode CMRH analogue à [60]. Commençons par considérer la factorisation LU de la matrice de Krylov $K_k \in \mathbb{R}^{N\times k}$:

$$K_k = L_k U_k, \tag{2.99}$$

où $L_k \in \mathbb{R}^{N\times k}$ est une matrice trapézoïdale inférieure unitaire et $U_k \in \mathbb{R}^{k\times k}$ est une matrice triangulaire supérieure. On peut écrire :

$$K_{k+1}\begin{pmatrix} 0 \\ I_k \end{pmatrix} = L_{k+1}U_{k+1}\begin{pmatrix} 0 \\ I_k \end{pmatrix} = AK_k = AL_kU_k,$$

et comme U_k^{-1} est triangulaire supérieure, on peut définir la matrice de Hessenberg supérieure $H_{k+1,k} \in \mathbb{R}^{(k+1)\times k}$ suivante :

$$H_{k+1,k} = U_{k+1}\begin{pmatrix} 0_{1\times k} \\ I_k \end{pmatrix}U_k^{-1} = \begin{pmatrix} H_k \\ h_{k+1,k}(e_k)^T \end{pmatrix},$$

avec H_k, une matrice Hessenberg supérieure carrée d'ordre k. Nous obtenons ainsi la relation suivante :

$$\begin{aligned} AL_k &= L_{k+1}U_{k+1}\begin{pmatrix} 0 \\ I_k \end{pmatrix}U_k^{-1} \\ &= L_{k+1}H_{k+1,k} = L_kH_k + h_{k+1,k}l_{k+1}e_k^T. \end{aligned} \tag{2.100}$$

Les colonnes de L_k forment une base non orthogonale du sous espace $K_k(A, r_0)$. On utilise le processus de Hessenberg défini dans la section 1.4.3 pour construire cette base $\{l_1, \cdots, l_k\}$. Les coefficients de ces opérations sont précisément les éléments de $H_{k+1,k}$.

Résultats de convergence

Nous rappelons un résultat sur la convergence entre les méthodes CMRH et GMRES. Les détails de la preuve se trouvent dans [60].

Soit $\alpha = \|r_0\|_\infty$. À l'étape k, l'approximation de la méthode CMRH est donnée par :

$$x_k = x_0 + L_k y_k, \tag{2.101}$$

où y_k est solution du problème de moindres carrées :

$$\min_{y\in\mathbb{R}^k} \|\alpha e_1^{(k+1)} - H_{k+1,k}y\|. \tag{2.102}$$

Il a été prouvé que l'approximation de la méthode CMRH minimise le pseudo-résidu défini par :

$$\min_{z\in K_k} \|L_{k+1}^+(Az - r_0)\|. \tag{2.103}$$

Théorème 2.4

Soient r_k^C et r_k^G les résidus des méthodes CMRH et GMRES à l'étape k utilisant le même résidu initial r_0. Alors

$$\|r_k^G\| \le \|r_k^C\| \le \kappa(L_{k+1})\|r_k^G\|,$$

où $\kappa(L_{k+1}) = \|L_{k+1}\|\|L_{k+1}^{-1}\|$ représente le conditionnement de la matrice L_{k+1}.

Algorithme

Dans [36], les auteurs présentent un algorithme de la méthode CMRH qui réduit le coût de mémoire. En effet, ils ont montré que les matrices L_k et $H_{k+1,k}$ pouvaient être

stockées dans la matrice A. Ils ont aussi comparé le coût en nombre d'opérations de la méthode CMRH par rapport à la méthode GMRES. Pour k étapes de l'algorithme, la méthode CMRH requiert $N^2 - k(k+1)/2$ multiplications et additions, tandis que l'algorithme de la méthode GMRES requiert $N^2 + 2Nk$ multiplications et additions. Nous donnons ici l'algorithme de la méthode CMRH avec pivotage et stockage dans la matrice A, voir Algorithme 15. Nous rappelons que l'algorithme de la méthode GMRES a été donné dans le chapitre 1, Algorithme 6.

Algorithme 15: Méthode CMRH

Soient x_0 et tol ;
$p = [1, 2, \cdots, N]^T$; Calculer $b = b - Ax_0$;
On détermine i_0 tel que $\left|(b)_{i_0}\right| = \|b\|_\infty$;
$\beta = (b)_{i_0},\ b = b/\beta$;
$(p)_1 \leftrightarrow (p)_{i_0},\ (b)_1 \leftrightarrow (b)_{i_0}$;
$A_{1,:} \leftrightarrow A_{i_0,:},\ A_{:,1} \leftrightarrow A_{:,i_0}$;
pour $k = 1, \cdots,$ *jusqu'à convergence* **faire**
  $u = A_{:,k} + A_{:,k+1:N}(b)_{k+1:N}$;
  $A_{k+1:N,k} = (b)_{k+1:N}$;
  pour $j = 1, \cdots, k$ **faire**
    $A_{j,k} = (u)_j,\ u_j = 0$;
    $(u)_{j+1:N} = (u)_{j+1:N} - A_{j,k}A_{j+1:N,j}$;
  fin
  On détermine $i_0 \in \{k+1, \cdots, N\}$ tel que $\left|(u)_{(p)_{i_0}}\right| = \left\|(u)_{(p)_{k+1}:(p)_N}\right\|_\infty$;
  $h = (u)_{(p)_{i_0}}$;
  $v = u/h$;
  $(p)_{k+1} \leftrightarrow (p)_{i_0},\ (v)_{k+1} \leftrightarrow (v)_{i_0}$;
  $A_{k+1,:} \leftrightarrow A_{i_0,:},\ A_{:,k+1} \leftrightarrow A_{:,i_0}$;
  Factorisation QR de $H_{k+1,k}$;
  On applique les rotations de Givens à $H_{k+1,k}$ et βe_1 ;
fin
On résout $H_k d_k = \beta e_1,\ (H_k = triu(A_{1:k,1:k}))$;
On calcule $x_k = x_0 + L_k d_k,\ (L_k = diag(ones(k,1)) + tril(A_{:,1:k}, -1))$;
On réordonne les coordonnées de x_k ;
pour $i = 1, \cdots, N$ **faire**
  $b_{(p)_i} = (x_k)_i$;
fin

2.5.3 Résultats Numériques

On a vu dans la section 2.5.1 que les méthodes d'extrapolation que nous étudions sont comparables à certaines méthodes de Krylov. Dans cette section, nous allons montrer cette équivalence sur des exemples.

Exemple 8. *On souhaite résoudre le problème discrétisé suivant issu de [4] :*

$$-u_{xx}(x,y) - u_{yy}(x,y) + 2u_x(x,y) + 2u_y(x,y) - 10u(x,y) = \phi(x,y), \ sur\,\Omega$$
$$u(x,y) = 1 \ sur\,\partial\Omega,$$

où Ω *est le carré unité. On utilise une discrétisation par une méthode de différences finies centrées qui conduit à résoudre un système linéaire de la forme* $Ax = b$*. La fonction* ϕ *est choisie de telle sorte que la solution exacte soit* $u(x,y) = 1$ *sur* Ω*. Nous appliquons les méthodes d'extrapolation à la suite* $\{s_k\}$ *définie par*

$$s_{k+1} = B_\omega s_k + c_\omega, \tag{2.104}$$

où

$$c_\omega = \omega(2-\omega)(D-\omega U)^{-1} D (D-\omega L)^{-1} b, \tag{2.105}$$

$$B_\omega = (D-\omega U)^{-1}(\omega L + (1-\omega)D)(D-\omega L)^{-1}(\omega U + (1-\omega)D) \tag{2.106}$$

et $A = D - L - U$ *est une décomposition classique.*

Lorsque la suite $\{s_k\}$ converge, la solution du problème de point fixe (2.104) est la solution du système préconditionné par la méthode SSOR suivant :

$$(I - B_\omega)x = c_\omega. \tag{2.107}$$

Comparaison méthode RRE / méthode GMRES

Nous commençons par comparer les méthodes RRE et GMRES. La figure 2.5 montre le comportement de la norme du résidu en fonction du nombre d'itérations pour $N = 1600$. Cet exemple montre bien que la méthode RRE est moins stable numériquement que la méthode GMRES. En effet, on observe bien que la perte de précision dans la résolution par la méthode RRE entraîne une augmentation du nombre d'itérations.

Ce phénomène s'observe aussi lorsque l'on utilise les méthodes cycliques. La figure 2.6 montre le comportement de la norme du résidu pour $N = 1600$ et les méthodes redémarrées toutes les 20 itérations.

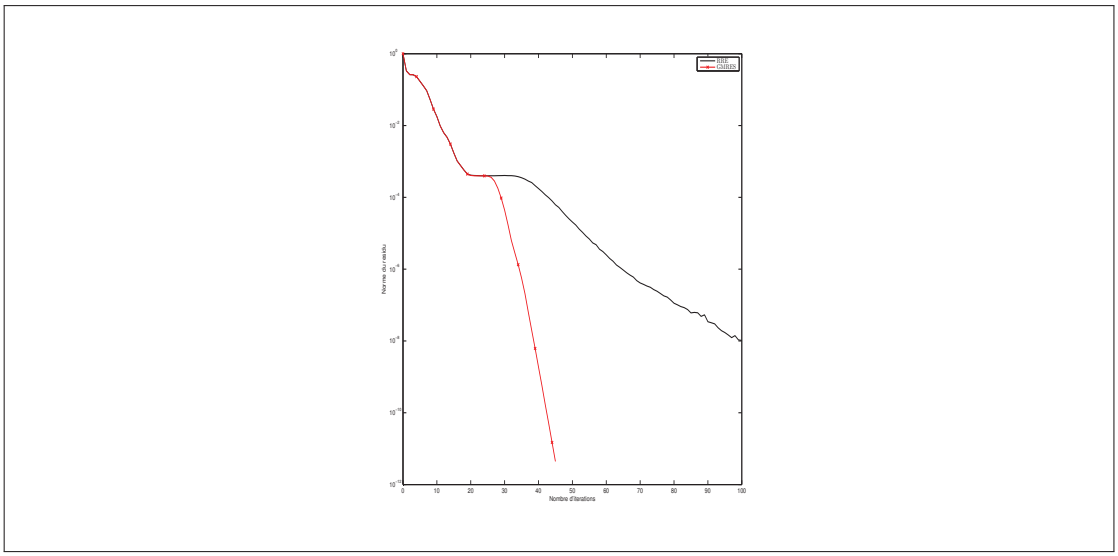


FIGURE 2.5 – Méthodes RRE et GMRES appliquées au problème discrétisé de l'exemple 8

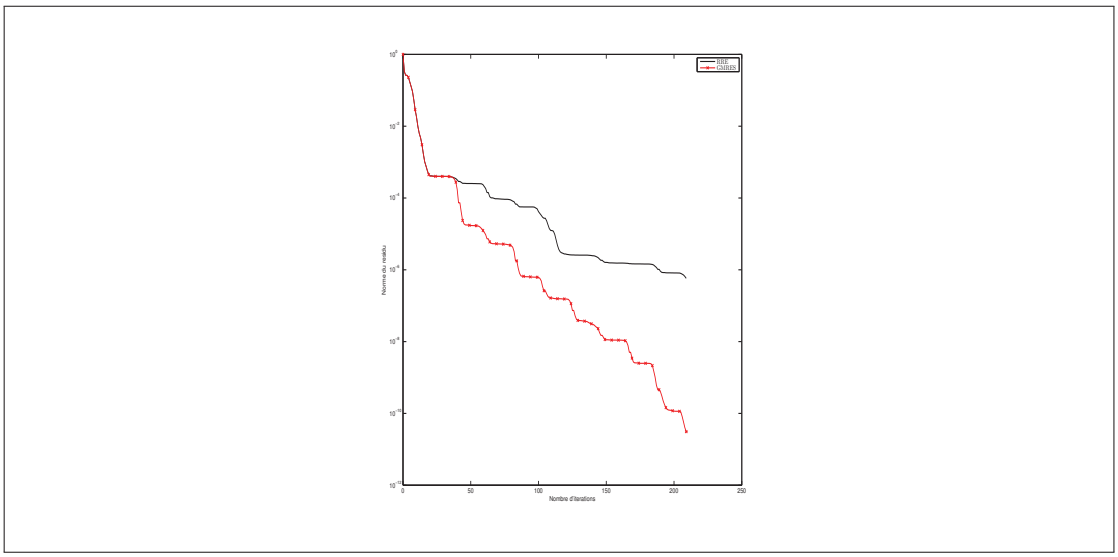


FIGURE 2.6 – Méthodes RRE et GMRES cycliques appliquées au problème discrétisé de l'exemple 8

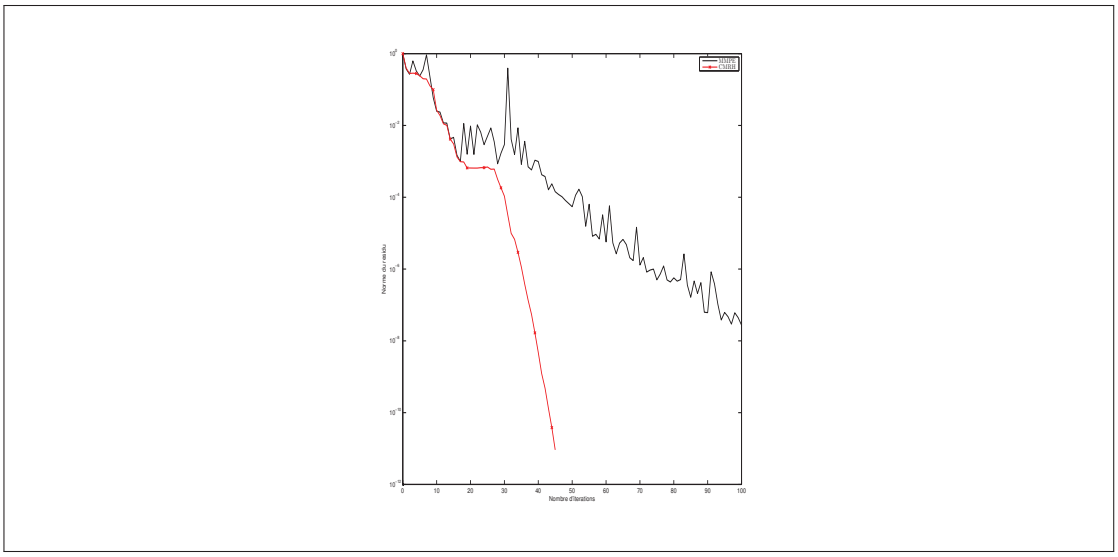


FIGURE 2.7 – Méthodes MMPE et CMRH appliquées au problème discrétisé de l'exemple 8

Comparaison méthode MMPE / méthode CMRH

Comparons maintenant les méthodes MMPE et CMRH pour le même exemple. La figure 2.7 montre le comportement de la norme du résidu pour $N = 1600$. Là encore, le même phénomène est observé. La méthode MMPE est encore moins stable que la méthode RRE. En effet, on observe une augmentation du nombre d'itérations et une grande instabilité de la norme du résidu.

Les résultats numériques montrent que les méthodes d'extrapolation, lorsqu'elles sont appliquées à la résolution d'un système linéaire, ne sont pas aussi stables que les méthodes de sous espaces de Krylov. Cela est dû à l'utilisation des matrices ΔS_q et $\Delta^2 S_q$ qui sont très mal conditionnées en général. De plus, on ne prend pas en compte les propriétés de la suite.

2.6 Conclusion

Dans ce chapitre, nous nous sommes intéressés à l'étude des méthodes d'extrapolation polynomiales scalaires et vectorielles. On a vu que la méthode de Newton entre dans la catégorie des méthodes d'extrapolation. Nous avons aussi défini les méthodes RRE, MPE et MMPE puis nous avons donné leur algorithme. Nous avons montré que la méthode RRE cyclique admettait une convergence quadratique avec des hypothèses plus réalisables dans la pratique pour des problèmes de petites tailles. Nous avons ensuite montré l'efficacité des méthodes d'extrapolation en les comparant aux méthodes de Newton et Newton inexactes. L'étude théorique sur la résolution

de systèmes d'équations linéaires a montré que ces méthodes étaient comparables à des méthodes de sous espaces de Krylov. Nous avons aussi défini la méthode CMRH qui est équivalente à la méthode MMPE. Nous avons remarqué que, sur les exemples numériques, leurs utilisations étaient moins intéressantes du fait du mauvais conditionnement des matrices ΔS_q et $\Delta^2 S_q$.

Le dernier point reste encore à être éclairci. En effet, on pourrait s'intéresser à trouver un algorithme plus stable numériquement dans le cas de systèmes linéaires. Des résultats de convergence similaires pour les méthodes MPE et MMPE restent encore à être développés.

Chapitre 3

Application 1 : Les équations de Navier-Stokes

3.1 Introduction

En mécanique des fluides, les équations de Navier-Stokes sont des équations aux dérivées partielles non linéaires qui décrivent le mouvement des fluides. La résolution de ces équations constitue l'un des enjeux majeurs en ingénierie. Elles jouent un rôle dans la conception des avions, dans les prédictions météorologiques. Elles modélisent les courants océaniques, les mouvements de l'air dans l'atmosphère, etc. La résolution ou l'analyse mathématiques des équations de Navier-Stokes constitue l'un des problèmes du prix du millénaire.

Il existe de nombreuses formes des équations de Navier-Stokes. En effet, les équations seront différentes dans le cas où le fluide est visqueux ou ne l'est pas, dans le cas où le fluide est compressible ou incompressible, dans le cas où les équations sont stationnaires ou pas. Ici, on s'intéressera uniquement aux équations stationnaires modélisant des fluides incompressibles.

Dans ce chapitre, on commencera par rappeler les équations de Navier-Stokes avec les conditions aux bords. Puis, nous en déduirons une formulation faible, ce qui nous permettra d'aboutir à la résolution d'un système non linéaire. Ensuite, nous rappellerons les deux méthodes de linéarisation utilisées actuellement. Il s'agit de la méthode de Picard et de la méthode de Newton. La méthode de Picard correspond à une simple méthode de point fixe, tandis que la méthode de Newton est une méthode à convergence quadratique locale. Chacune de ces méthodes possède ses avantages et ses inconvénients dans la résolution de notre problème. On se servira de la suite formée par la méthode de Picard pour appliquer les méthodes d'extrapolation étudiées dans le chapitre 2. Enfin, on donnera quelques exemples numériques pour des cas

concrets qui montrent l'efficacité de nos méthodes lorsque celles-ci sont intégrées dans IFISS (Incompressible Flow Iterative Solution Software) [23].

3.2 Les équations de Navier-Stokes

Les notations utilisées dans cette partie sont identiques à [24, ch. 7]. On s'intéresse à la résolution du système d'équations suivant :

$$\begin{cases} -\nu\nabla^2 u + u\cdot\nabla u + \nabla p &= f \\ \nabla\cdot u &= 0, \end{cases} \tag{3.1}$$

où $u \in \mathbb{R}^3$ et $p \in \mathbb{R}^3$ représentent respectivement la vitesse et la pression du fluide et où le vecteur f et la constante ν sont connus. ν désigne la viscosité cinématique du fluide. Cette valeur correspond au quotient de la viscosité dynamique par la masse volumique du fluide. Par exemple, la viscosité dynamique de l'eau vaut environ $10^{-3} kg/ms$. La première équation modélise la conservation du moment du fluide. On ne s'intéresse qu'aux équations stationnaires, c'est-à-dire qu'on supposera que le vecteur vitesse ne dépend pas du temps. La deuxième équation modélise la conservation de la masse. On peut aussi dire qu'elle modélise la contrainte d'incompressibilité du fluide. On dira qu'un fluide est incompressible lorsque sa masse volumique ne dépend pas (ou pratiquement pas) de la pression ou de la température.

On considère le problème posé sur un domaine Ω de dimension 2 ou 3 avec des conditions aux bords définies par :

$$\begin{aligned} u &= w \quad \text{sur } \partial\Omega_D \\ \nu\tfrac{\partial u}{\partial n} - \vec{n}p &= 0 \quad \text{sur } \partial\Omega_N, \end{aligned} \tag{3.2}$$

où $\vec{n}$ dénote le vecteur normal sortant et $\partial\Omega = \partial\Omega_D \cup \partial\Omega_N$. Si la vitesse est donnée sur tout le bord, c'est-à-dire, si $\partial\Omega_D \equiv \partial\Omega$, alors la solution pour le vecteur pression du problème de Navier-Stokes (3.1) et (3.2) est unique à une constante hydrostatique près.

3.2.1 Formulation faible

Notations et Définitions

Commençons par définir quelques notations qui seront utilisées dans ce chapitre. L'espace des fonctions carrées intégrables au sens de Lebesgue sera défini par :

$$L_2(\Omega) := \left\{ u : \Omega \to \mathbb{R} \,\middle|\, \int_\Omega u^2 < \infty \right\},$$

et l'espace de Sobolev par :

$$\mathscr{H}^1(\Omega) := \left\{ u : \Omega \to \mathbb{R} \,\middle|\, u, \frac{\partial u}{\partial x}, \frac{\partial u}{\partial y} \in L_2(\Omega) \right\}.$$

Nous définissons aussi des espaces de solutions et des espaces tests :

$$H_E^1 := \left\{ u \in \mathscr{H}^1(\Omega)^2 \,|\, u = w \text{ sur } \partial\Omega_D \right\},$$
$$H_{E_0}^1 := \left\{ v \in \mathscr{H}^1(\Omega)^2 \,|\, v = 0 \text{ sur } \partial\Omega_D \right\}.$$

Formulation faible

Avec ces notations, nous pouvons maintenant définir la formulation faible de (3.1) et (3.2). Trouver $u \in H_E^1$ et $p \in L_2(\Omega)$ tels que :

$$\nu \int_\Omega \nabla u : \nabla v + \int_\Omega (u \cdot \nabla u) \cdot v - \int_\Omega p \cdot \nabla v = \int_\Omega f \cdot v \quad \forall v \in H_{E_0}^1$$
$$\int_\Omega q \cdot \nabla u = 0 \quad \forall q \in L_2(\Omega). \tag{3.3}$$

On note par $\nabla u : \nabla v$ le produit terme à terme entre ∇u et ∇v, par exemple, en deux dimensions $\nabla u : \nabla v = \nabla u_x \cdot \nabla v_x + \nabla u_y \cdot \nabla v_y$.

Preuve 3.1. *Soient $v \in H_{E_0}^1$ et $q \in L_2(\Omega)$, alors d'après (3.1) :*

$$\int_\Omega v.\left(-\nu\nabla^2 u + u.\nabla u + \nabla p\right) = \int_\Omega f.v \tag{3.4}$$
$$\int_\Omega q \cdot \nabla u = 0 \tag{3.5}$$

En utilisant la formule de Green, on a :

$$\int_\Omega v \cdot \nabla p = -\int_\Omega p \cdot \nabla v + \int_\Omega \nabla(pv) \tag{3.6}$$
$$= -\int_\Omega p \cdot \nabla v + \int_{\partial\Omega} pv \cdot \vec{n} \tag{3.7}$$
$$= -\int_\Omega p \cdot \nabla v \; car \; v \in H_{E_0}^1 \tag{3.8}$$

et

$$-\int_\Omega v \cdot \nabla^2 u = \int_\Omega \nabla u : \nabla v - \int_\Omega \nabla(\nabla u \cdot v)$$
$$= \int_\Omega \nabla u : \nabla v - \int_{\partial\Omega} (\vec{n}.\nabla u).v$$
$$= \int_\Omega \nabla u : \nabla v \; car \; v \in H_{E_0}^1$$

d'où (3.3).

La deuxième équation est immédiate.

□

Linéarisation

Comme le problème est non linéaire, on doit passer par une phase de linéarisation. Ainsi étant donné une solution initiale $(u_0, p_0) \in H^1_E \times L_2(\Omega)$, on construit une suite (u_0, p_0), (u_1, p_1), (u_2, p_2),... $\in H^1_E \times L_2(\Omega)$ qui converge vers la solution de la formulation faible (3.3). On rappellera les méthodes de linéarisation dans la prochaine section de ce chapitre.

Comme pour toutes les méthodes itératives, on doit définir un critère d'arrêt. Pour cela, on introduit le résidu non linéaire associé à la formulation faible (3.3). Étant donné une itération (u_k, p_k), le résidu à l'étape k est défini par un couple de vecteurs $(R_k(v), r_k(q))$ satisfaisant :

$$\begin{aligned} R_k(v) &= \int_\Omega f \cdot v - \int_\Omega (u_k \cdot \nabla u_k) \cdot v - \nu \int_\Omega \nabla u_k : \nabla v + \int_\Omega p_k \cdot \nabla v, \\ r_k(q) &= -\int_\Omega q \cdot \nabla u_k \end{aligned} \tag{3.9}$$

pour tout $v \in H^1_{E_0}$ et $q \in L_2(\Omega)$. En posant dans (3.3), $u = u_k + \delta u_k$ et $p = p_k + \delta p_k$, on montre que :

Proposition 8. *$\delta u_k \in H^1_{E_0}$ et $\delta p_k \in L_2(\Omega)$ satisfont :*

$$\begin{aligned} D(u_k, \delta u_k, v) + \nu \int_\Omega \nabla \delta u_k : \nabla v - \int_\Omega \delta p_k \cdot \nabla v &= R_k(v) \\ \int_\Omega q \cdot \nabla \delta u_k &= r_k(q) \end{aligned} \tag{3.10}$$

pour tout $v \in H^1_{E_0}$, $q \in L_2(\Omega)$ où $D(u_k, \delta u_k, v)$ est la différence des termes non linéaires définie par

$$D(u_k, \delta u_k, v) = \int_\Omega (\delta u_k \cdot \nabla \delta u_k) \cdot v + \int_\Omega (\delta u_k \cdot \nabla u_k) \cdot v + \int_\Omega (u_k \cdot \nabla \delta u_k) \cdot v. \tag{3.11}$$

Preuve 3.2. *Commençons par montrer la première équation. Soit $v \in H^1_{E_0}$,*

$$\nu \int_\Omega \nabla u : \nabla v + \int_\Omega (u \cdot \nabla u) \cdot v - \int_\Omega p \cdot \nabla v = \int_\Omega f \cdot v \Leftrightarrow$$

$$\nu \int_\Omega \nabla (u_k + \delta u_k) : \nabla v + \int_\Omega ((u_k + \delta u_k) \cdot \nabla (u_k + \delta u_k)) \cdot v - \int_\Omega (p_k + \delta p_k) \cdot \nabla v = \int_\Omega f \cdot v \Leftrightarrow$$

$$\nu \int_\Omega \nabla \delta u_k : \nabla v + D(u_k, \delta u_k, v) - \int_\Omega \delta p_k \cdot \nabla v = R_k(v),$$

d'où le résultat. La deuxième équation se démontre facilement.

□

Le nouvel itéré est obtenu en posant $u_{k+1} = u_k + \delta u_k$ et $p_{k+1} = p_k + \delta p_k$.

3.2.2 Formulation faible discrète

Pour pouvoir résoudre numériquement ces équations, nous avons besoin de les définir sur des espaces de dimension fini. Soient $X_0^h \subset H_{E_0}^1$ et $M^h \subset L_2(\Omega)$. Étant donné un espace de solution vitesse X_E^h, la version discrète de (3.3) devient : Trouver $u_h \in X_E^h$ et $p \in M^h$ tels que

$$\begin{aligned} \nu \int_\Omega \nabla u_h : \nabla v_h + \int_\Omega (u_h \cdot \nabla u_h) \cdot v_h - \int_\Omega p_h \cdot \nabla v_h = \int_\Omega f \cdot v_h \quad \forall v_h \in X_0^h \\ \int_\Omega q_h \cdot \nabla u_h = 0 \quad \forall q_h \in M^h. \end{aligned} \tag{3.12}$$

En précédant de la même façon que dans la section précédente, on obtient un système d'équations analogue pour (3.10) : trouver $\delta u_{h,k} \in X_0^h$ et $\delta p_{h,k} \in M^h$ satisfaisant

$$\begin{aligned} D_k(u_{h,k}, \delta u_{h,k}, v_h) + \nu \int_\Omega \nabla \delta u_{h,k} : \nabla v_h - \int_\Omega \delta p_{h,k} \cdot \nabla v_h = R_k(v_h) \\ \int_\Omega q_h \cdot \nabla \delta u_{h,k} = r_k(q_h), \end{aligned} \tag{3.13}$$

pour tout $v_h \in X_0^h$ et $q_h \in M^h$.

On définit maintenant des fonctions de base pour les espaces d'éléments finis. Cela nous permettra de nous ramener à la résolution d'un système d'équations linéaires algébrique. Introduisons donc un ensemble de fonctions de base $\{\phi_j\}$ vérifiant :

$$\begin{aligned} u_{h,k} &= \sum_{j=1}^{n_u} u_{j,k}\phi_j + \sum_{j=n_u+1}^{n_u+n_\delta} u_{j,k}\phi_j, \\ \delta u_{h,k} &= \sum_{j=1}^{n_u} \delta u_{j,k}\phi_j. \end{aligned} \tag{3.14}$$

On fixe aussi les coefficients $u_{j,k}, j = n_u + 1, \ldots, n_u + n_\delta$ pour que le second terme interpole les données aux bords $\delta\Omega_D$. On fait la même chose pour le vecteur pression.

On introduit des fonctions de base $\{\psi_l\}$ vérifiant

$$p_{h,k} = \sum_{l=1}^{n_p} p_{l,k}\psi_l,$$
$$\delta p_{h,k} = \sum_{l=1}^{n_p} \Delta p_{l,k}\psi_l. \tag{3.15}$$

En remplaçant les expressions (3.14) et (3.15) dans (3.13), on obtient le système d'équations suivant :

$$\begin{bmatrix} \nu A + D_k & B^T \\ B & O \end{bmatrix} \begin{bmatrix} \delta u_k \\ \delta p_k \end{bmatrix} = \begin{bmatrix} R_k \\ r_k \end{bmatrix}, \tag{3.16}$$

où A est la matrice représentant le Laplacien

$$A = \left[a_{i,j}\right], \quad a_{i,j} = \int_\Omega \nabla\phi_i : \nabla\phi_j,\ i,j = 1,\cdots,n_u,$$

B la matrice de divergence

$$B = \left[b_{l,j}\right], \quad b_{l,j} = -\int_\Omega \psi_l \cdot \nabla\phi_j,\ j = 1,\cdots,n_u,\ l = 1,\cdots,n_p,$$

D_k la matrice des termes non linéaires

$$D_k = \left[d_{i,j}\right], \quad d_{i,j} = \int_\Omega (\phi_j \cdot \nabla\delta u_{h,k})\cdot\phi_i + \int_\Omega (\phi_j \cdot \nabla u_{h,k})\cdot\phi_i + \int_\Omega (u_{h,k}\cdot\nabla\phi_j)\cdot\phi_i$$

où $i = 1,\ldots,n_u$ et $j = 1,\ldots,n_u$. Le membre de droite dans (3.16) représente le résidu non linéaire associé défini par :

$$R_k = [R_i], \quad R_i = \int_\Omega f\cdot\phi_i - \int_\Omega (u_{h,k}\cdot\nabla u_{h,k})\cdot\phi_i - \nu\int_\Omega \nabla u_{h,k} : \nabla\phi_i + \int_\Omega p_{h,k}\cdot\nabla\phi_i,$$

$$r_k = [r_l], \quad r_l = \int_\Omega \psi_l \cdot \nabla u_{h,k}.$$

On définira les espaces d'approximation dans la section des résultats numériques. Pour certaines approximations, la matrice nulle dans (3.16) est remplacée par une matrice que l'on nommera $-C$ qui est semi-définie négative.

Maintenant, nous allons définir les techniques de linéarisation : la méthode de Newton et la méthode de Picard. Ces techniques permettront de définir des algorithmes pour la résolution de ce système et seront intéressantes dans la suite de ce chapitre pour l'utilisation des méthodes d'extrapolation.

3.3 Méthodes de linéarisation

Dans cette section, on rappelle deux techniques classiques de linéarisation utilisées dans la résolution numérique des équations de Navier-Stokes. La première technique est la méthode de Newton. L'avantage de cette méthode est qu'elle possède une convergence quadratique, ce qui est très intéressant dans notre cas. Le gros problème est que cette convergence ne soit que locale. Son rayon de convergence est proportionnel au paramètre de viscosité ν. Ainsi, plus le paramètre de viscosité diminue, meilleure devra être la donnée initiale pour que la méthode converge. Pour pallier à ce problème, on utilise une autre technique de linéarisation, qui elle, a un très grand rayon de convergence. Il s'agit de la méthode de Picard, qui est une méthode de point fixe. Le problème de cette méthode et de toutes les méthodes de point fixe est que sa convergence soit très lente. La problématique est donc la suivante : comment obtenir une bonne convergence ? La réponse à cette question, donnée par Silvester et al. dans [24], est la suivante. On commence par faire quelques itérations avec la méthode de Picard puis, lorsque l'on est suffisamment proche de la solution, on utilise la méthode de Newton. Cette technique fonctionne bien mais le problème du nombre d'itérations de la méthode de Picard se pose.

3.3.1 Méthode de Newton

Pour mettre en place la méthode de Newton, il suffit de considérer la forme développée de $D(u_k, \delta u_k, v)$ et de retirer le terme du second terme $\int_\Omega (\delta u_k \cdot \nabla \delta u_k) \cdot v$. Dans ce cas, la formulation faible (3.10) devient un problème linéarisé : pour tout $v \in H^1_{E_0}$ et $q \in L_2(\Omega)$, trouver $\delta u_k \in H^1_{E_0}$ et $\delta p_k \in L_2(\Omega)$ satisfaisant

$$\begin{aligned} \int_\Omega (u_k \cdot \nabla \delta u_k) \cdot v + \int_\Omega (\delta u_k \cdot \nabla u_k) \cdot v + \nu \int_\Omega \nabla \delta u_k : \nabla v - \int_\Omega \delta p_k \cdot \nabla v &= R_k(v) \\ \int_\Omega q \cdot \nabla \delta u_k &= r_k(q). \end{aligned} \tag{3.17}$$

Comme dans la section précédente, on doit maintenant se ramener à un problème en dimension finie. En utilisant les mêmes notations, on obtient : trouver $\delta u_{h,k} \in X^h_0$ et $\delta p_{h,k} \in M^h$ satisfaisant

$$\begin{aligned} \int_\Omega (\delta u_{h,k} \cdot \nabla u_{h,k}) \cdot v_h + \int_\Omega (u_{h,k} \cdot \nabla \delta u_{h,k}) \cdot v_h + \nu \int_\Omega \nabla \delta u_{h,k} : \nabla v_h - \int_\Omega \delta p_{h,k} \cdot \nabla v_h &= R_k(v_h) \\ \int_\Omega q_h \cdot \nabla \delta u_{h,k} &= r_k(q_h), \end{aligned} \tag{3.18}$$

pour tout $v_h \in X_0^h$ et $q_h \in M^h$. Cela permet de définir le problème d'algèbre linéaire correspondant :

$$\begin{bmatrix} \nu A + N_k & B^T \\ B & O \end{bmatrix} \begin{bmatrix} \delta u_k \\ \delta p_k \end{bmatrix} = \begin{bmatrix} R_k \\ r_k \end{bmatrix}, \tag{3.19}$$

où N_k désigne la matrice correspond à la somme des termes de convection et des dérivées. Ces coefficients vérifient :

$$N_k = \left[n_{i,j}\right], \quad n_{i,j} = \int_\Omega (u_{h,k} \cdot \nabla \phi_j) \cdot \phi_i + \int_\Omega (\phi_j \cdot \nabla u_{h,k}) \cdot \phi_i, \tag{3.20}$$

où $i = 1, \ldots, n_u$, $j = 1, \ldots, n_u$.

On peut maintenant définir un processus qui permet de résoudre numériquement notre problème (voir Algorithme 16).

Algorithme 16: Algorithme de la méthode de Newton

Étape 1. Entrée : le vecteur $x_0 = \left(u_0, p_0\right)^T$;
Calculer le vecteur $(R_0, r_0)^T$ et poser $k = 0$;
Étape 2. Utiliser A, B, et C (si besoin). Calculer N_k et f_k ;
Résoudre le système (3.19) pour trouver le vecteur $\delta x_k = \left(\delta u_k, \delta p_k\right)^T$;
Poser $x_{k+1} = x_k + \delta x_k$;
Calculer le vecteur $(R_{k+1}, r_{k+1})^T$;
Étape 3. Si $\left\|(R_{k+1}, r_{k+1})^T\right\| < tol$, stop ;
Sinon poser $k = k + 1$ et retour à l'étape 2;

3.3.2 Méthode de Picard

On s'intéresse maintenant à la méthode de Picard ou Oseen [6, 50]. La mise en place est similaire à celle de la méthode de Newton. Cette fois-ci, en plus du terme $\int_\Omega (\delta u_k \cdot \nabla \delta u_k) \cdot v$, on enlève le terme $\int_\Omega (\delta u_k \cdot \nabla u_k) \cdot v$. La formulation faible (3.10) devient : pour tout $v \in H^1_{E_0}$ et $q \in L_2(\Omega)$, trouver $\delta u_k \in H^1_{E_0}$ et $\delta p_k \in L_2(\Omega)$ satisfaisant

$$\begin{aligned} \int_\Omega (u_k \cdot \nabla \delta u_k) \cdot v + \nu \int_\Omega \nabla \delta u_k : \nabla v - \int_\Omega \delta p_k \cdot \nabla v &= R_k(v) \\ \int_\Omega q \cdot \nabla \delta u_k &= r_k(q). \end{aligned} \tag{3.21}$$

En comparant (3.21) avec la formulation faible, nous voyons que l'itération de Picard correspond à une simple itération de la méthode de point fixe pour résoudre (3.3). Comme précédemment, on se ramène à un problème en dimension finie. On

définit la formulation faible discrète de (3.21) : trouver $\delta u_{h,k} \in X_0^h$ et $\delta p_{h,k} \in M^h$ satisfaisant

$$\int_\Omega (u_{h,k} \cdot \nabla \delta u_{h,k}) \cdot v_h + \nu \int_\Omega \nabla \delta u_{h,k} : \nabla v_h - \int_\Omega \delta p_{h,k} \cdot \nabla v_h = R_k(v_h)$$
$$\int_\Omega q_h \cdot \nabla \delta u_{h,k} = r_k(q_h), \tag{3.22}$$

pour tout $v_h \in X_0^h$ et $q_h \in M^h$. On en déduit le système d'équations linéaires en utilisant les mêmes notations que dans la section précédente :

$$\begin{bmatrix} \nu A + W_k & B^T \\ B & O \end{bmatrix} \begin{bmatrix} \delta u_k \\ \delta p_k \end{bmatrix} = \begin{bmatrix} R_k \\ r_k \end{bmatrix}, \tag{3.23}$$

où W_k représente la matrice de convection :

$$W_k = \left[w_{i,j} \right], \quad w_{i,j} = \int_\Omega (\phi_j \cdot \nabla u_{h,k}) \cdot \phi_i, \tag{3.24}$$

où $i = 1, \ldots, n_u$, $j = 1, \ldots, n_u$. Dans l'algorithme 17, on donne la méthode pour résoudre numériquement les équations de Navier-Stokes par la méthode de Picard.

Algorithme 17: Algorithme de la méthode de Picard

Étape 1. Entrée : le vecteur $x_0 = \left(u_0, p_0\right)^T$;
Calculer W_0 et le vecteur $(R_0, r_0)^T$ et poser $k = 0$;
Étape 2. Utiliser A, B, et C (si besoin) pour
résoudre le système (3.23) et obtenir le vecteur $\delta x_k = \left(\delta u_k, \delta p_k\right)^T$;
Poser $x_{k+1} = x_k + \delta x_k$;
Calculer W_{k+1} et le vecteur $(R_{k+1}, r_{k+1})^T$;
Étape 3. Si $\left\| (R_{k+1}, r_{k+1})^T \right\| < tol$, stop ;
Sinon poser $k = k + 1$ et retour à l'étape 2 ;

3.3.3 Relation entre la méthode de Picard et la méthode de Newton

Posons M_P et M_N, les matrices des systèmes linéaires formant, respectivement, les suites de la méthode de Picard et de la méthode de Newton. On peut remarquer que :

$$M_N = M_P + \begin{pmatrix} E & 0 \\ 0 & 0 \end{pmatrix} \tag{3.25}$$

où E est la matrice définie par

$$E = [e_{i,j}],\ e_{i,j} = \int_\Omega (u_h \cdot \nabla \Phi_j) \cdot \Phi_i,\ i,j = 1, \cdots, n_u. \tag{3.26}$$

Nous donnons ici une nouvelle relation de l'inverse de la matrice M_N qui pourra être utile en théorie pour la suite de l'étude.

Lemme 2. *Supposons que M_P et E soient inversibles alors*

$$M_N^{-1} = M_P^{-1} - M_P^{-1} \begin{pmatrix} I \\ 0 \end{pmatrix} \left[E^{-1} + \begin{pmatrix} I \\ 0 \end{pmatrix} M_P^{-1} \begin{pmatrix} I & 0 \end{pmatrix} \right]^{-1} \begin{pmatrix} I & 0 \end{pmatrix} M_P^{-1}. \tag{3.27}$$

Preuve 3.3.

$$\begin{aligned} M_N &= M_P + \begin{pmatrix} E & 0 \\ 0 & 0 \end{pmatrix} \\ &= M_P + \begin{pmatrix} I \\ 0 \end{pmatrix} E \begin{pmatrix} I & 0 \end{pmatrix}. \end{aligned} \tag{3.28}$$

Et on applique la formule de Sherman-Morrison-Woodbury définie dans la proposition 2.

□

3.4 Application de la méthode RRE

Dans ce paragraphe, on s'intéresse à l'application de la méthode RRE à la suite obtenue par la méthode de Picard. On donne dans cette section, l'algorithme que l'on va considérer dans les exemples numériques. On ne donne ici que l'algorithme correspond à la méthode RRE. Les algorithmes correspondant aux autres méthodes se déduisent facilement. Comme il a été dit dans la section précédente, la méthode de Picard a une convergence lente. On va donc appliquer la méthode d'extrapolation à la suite formée par les itérations de Picard pour améliorer la vitesse de convergence. Commençons par définir explicitement la suite considérée.

Posons G_P, la fonction définissant la suite de la méthode de Picard. Alors G_P est définie par :

$$G_P(x) = \begin{pmatrix} \nu A + W(x) & B^T \\ B & 0 \end{pmatrix}^{-1} \begin{pmatrix} f(x) \\ g \end{pmatrix}. \tag{3.29}$$

L'algorithme 18 montre l'application de la méthode d'extrapolation à la suite formée par la méthode de Picard. Nous utiliserons cet algorithme dans la section suivante.

Algorithme 18: Algorithme de la méthode RRE appliquée à la méthode de Picard

Étape 1. Entrée : le vecteur s_0 et $k = 0$;

Étape 2. Calculer $s_{k+1} = G_P(s_k)$;

Poser $y = \delta s_k = s_{k+1} - s_k$;

pour $i = 1 : k-1$ **faire**

$R_{i,k} = Q_{:,i}^T y$;

$y = y - R_{i,k} Q_{:,i}$;

fin

$R_{k,k} = \|y\|$;

$Q_{:,k} = (1/R_{k,k})y$;

Poser $e_k = (1, \cdots, 1)^T$;

Résoudre $RR^T d_k = e_k$;

$\gamma_k = (1/\sum d_k) d_k$;

$(\epsilon_k)_1 = 1 - (\gamma_k)_1$;

pour $i = 1 : k-1$ **faire**

$(\epsilon_k)_i = (\gamma_k)_{i-1} - (\gamma_k)_i$;

fin

$t_k = s_0 + QR\epsilon_k$;

Calculer le vecteur $\left(R'_{k+1}, r'_{k+1}\right)^T$ en utilisant t_k;

Étape 3. Si $\left\| \left(R'_{k+1}, r'_{k+1}\right)^T \right\| < tol$, stop;

Sinon poser $k = k+1$ et retour à l'étape 2;

3.5 Résultats Numériques

Tous les programmes de ce chapitre ont été réalisés en utilisant MATLAB 7.1 avec la version la plus récente du package IFISS (Incompressible Flow Iterative Solution Software). IFISS utilise des méthodes d'éléments finis utilisant des rectangles pour discrétiser le système de Navier-Stokes (3.1). On teste notre méthode sur deux exemples concrets : un fluide s'écoulant dans un tunnel contenant une expansion soudaine et un fluide s'écoulant dans un tunnel contenant un obstacle. Pour tous les tests, le programme s'arrête lorsque la norme du résidu non linéaire défini dans l'algorithme 17 est inférieure à 10^{-8}. Le vecteur initial x_0 est donné par la solution du problème de Stokes associé définie par :

$$\begin{aligned} -\nu\nabla^2 u + \nabla p &= 0 \\ \nabla \cdot u &= 0. \end{aligned} \tag{3.30}$$

Le but de cette partie est de montrer qu'en utilisant les méthodes d'extrapolation, on observe une accélération de la convergence en se servant de la suite formée par les itérations de Picard.

3.5.1 Fluide s'écoulant dans un tunnel contenant une expansion soudaine

Dans cet exemple, on considère un fluide s'écoulant dans un conduit rectangulaire contenant une soudaine expansion. On suppose un écoulement de Poiseuille à l'entrée du conduit, c'est-à-dire que l'écoulement du fluide est parallèle aux parois, la vitesse est nulle sur le bord et la pression ne varie pas dans l'épaisseur de l'écoulement. On impose de plus, des conditions de Neumann à la sortie du conduit, ce qui entraîne une pression nulle. La figure 3.1 montre le comportement de la solution et le rendu en trois dimensions de la pression lorsque le terme $\nu = 1/600$. Pour cet exemple, on va tester l'efficacité des méthodes d'extrapolation sur la convergence lorsque l'on fait varier le terme de viscosité, le mode de discrétisation et la longueur du conduit.

Variation du terme de viscosité

Commençons par faire varier le terme de viscosité. On choisit la longueur du conduit égale à 30 et $Q_1 - P_0$ comme mode de discrétisation. La taille du système est $N = 12242$. Le mode de discrétisation sera défini après. Les figures 3.2 et 3.3 montrent les courbes représentant la convergence de la méthode de Picard et des méthodes RRE, MPE et MMPE pour des termes de viscosité différents ($\nu = 1/50$, $\nu = 1/100$, $\nu = 1/300$ et $\nu = 1/600$). Les courbes montrent que plus le terme de viscosité est petit, plus la convergence de la méthode de Picard est lente. Pour les méthodes d'extrapolation,

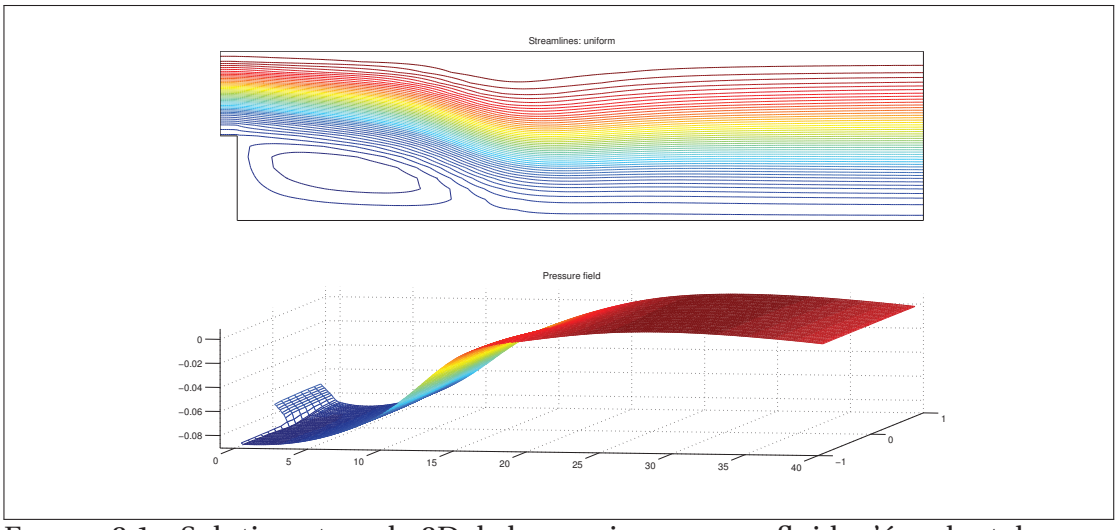


FIGURE 3.1 – Solution et rendu 3D de la pression pour un fluide s'écoulant dans un tunnel contenant une expansion soudaine

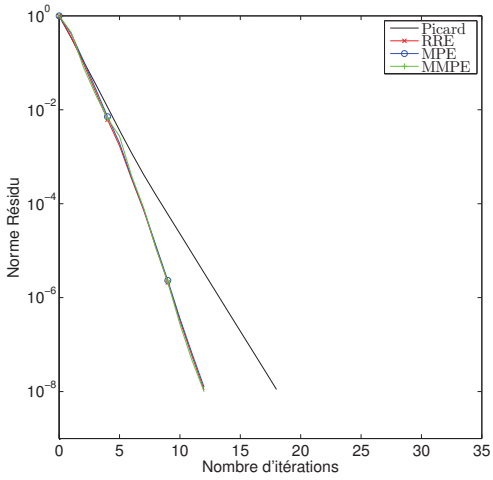
on observe le même phénomène sur chaque courbe. On a une convergence similaire à celle de Picard pendant la première partie, puis une accélération soudaine de la convergence. Ce processus s'accentue au fur et à mesure que le terme de viscosité diminue. On remarque aussi que les courbes des méthodes d'extrapolation sont similaires, même si la méthode MMPE est parfois moins régulière que les autres.

Variation de la longueur du conduit

Maintenant, on fait varier la longueur du conduit toujours avec $Q_1 - P_0$. On pose le terme de viscosité égal à $1/300$ et on regarde le comportement de la convergence pour différentes longueurs de conduit ($L = 5$, $L = 10$, $L = 30$, $L = 40$) donc pour différentes tailles de systèmes ($N = 2242$, $N = 4242$, $N = 12242$, $N = 16242$). Les figures 3.4 et 3.5 montrent les résultats obtenus. La longueur du conduit n'a pas d'incidence sur la convergence de la méthode de Picard pour $L \geq 10$. Cela en est de même pour les méthodes d'extrapolation. Pour le cas où $L = 5$, on observe une différence dans la convergence de la méthode de Picard qui est moins lisse que pour les autres courbes. Ces différences s'observent aussi pour les méthodes d'extrapolation.

Variation de la discrétisation

Regardons ce qui se produit lorsque l'on modifie le mode de discrétisation. Posons la longueur du conduit égale à $L = 30$ et le terme de viscosité égale à $\nu = 1/300$. Nous allons tester la convergence de nos méthodes pour quatre discrétisations différentes. Nous considérons deux discrétisations dites stables $Q_2 - Q_1$ et $Q_2 - P_{-1}$ et deux discrétisations dites non stables $Q_1 - Q_1$ et $Q_1 - P_0$. Pour ces deux dernières la matrice



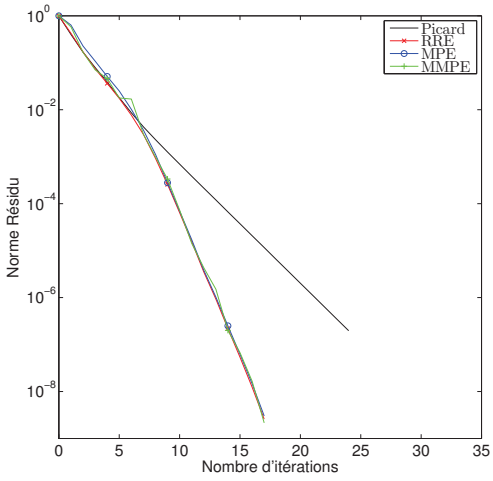
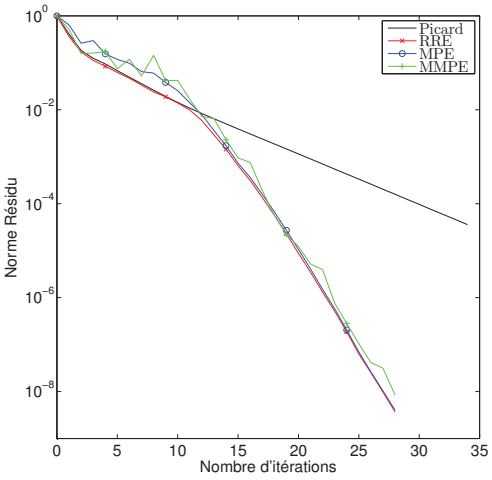


FIGURE 3.2 – Variation du terme de viscosité ($\nu = 1/50, 1/100$)



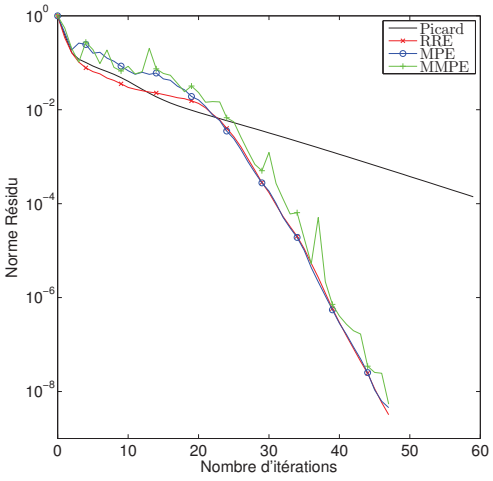
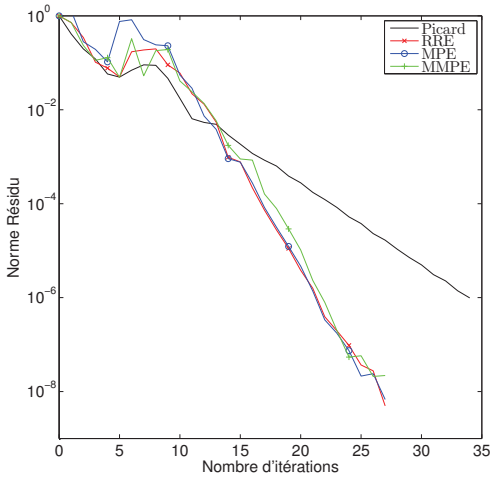


FIGURE 3.3 – Variation du terme de viscosité ($\nu = 1/300, 1/600$)



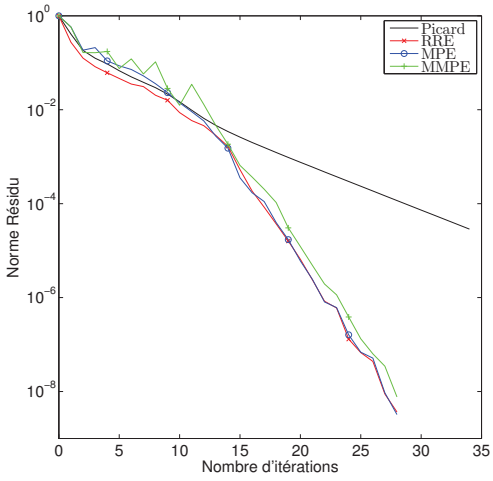
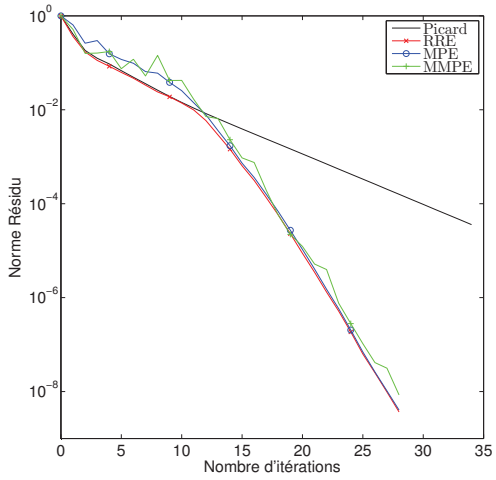


FIGURE 3.4 – Variation de la longueur du conduit ($L = 5,10$)



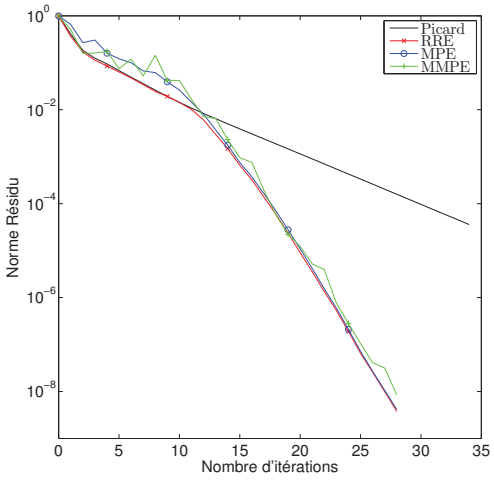


FIGURE 3.5 – Variation de la longueur du conduit ($L = 30, 40$)

nulle utilisée dans (3.23) est remplacée par une matrice $-C$ semi-définie négative dépendante de l'inverse du terme de viscosité.

La figure 3.6 montre les résultats obtenus et la position des nœuds utilisés pour la discrétisation $Q_1 - Q_1$ et une taille de système valant $N = 12507$. Les points noirs correspondent aux nœuds du maillage utilisés pour discrétiser le vecteur vitesse et les points cerclés correspondent aux nœuds utilisés pour discrétiser le vecteur pression. Pour ce maillage, on considère quatre nœuds par élément pour chaque vecteur. La figure 3.7 montre les résultats obtenus et la position des nœuds pour la discrétisation $Q_1 - P_0$ ($N = 12242$). Cette fois-ci, on considère uniquement le point au centre du rectangle pour le vecteur pression. Par comparaison avec la figure précédente, on observe une convergence similaire pour toutes les courbes. La figure 3.8 montre les résultats obtenus et la position des nœuds pour la discrétisation $Q_2 - Q_1$ ($N = 9447$). Neuf points par élément sont utilisés pour la discrétisation du vecteur vitesse. On observe qu'avec cette discrétisation, la convergence est plus lente (environ 50 itérations au lieu de 28 pour les autres courbes). Mais le comportement est toujours le même (convergence similaire puis accélération). La figure 3.9 montre les résultats obtenus et la position des nœuds pour la discrétisation $Q_2 - P_{-1}$ ($N = 11266$). La discrétisation P_{-1} utilise la valeur du point au centre du rectangle et les dérivées dans les deux directions. Les courbes obtenues sont comparables à la discrétisation $Q_2 - Q_1$.

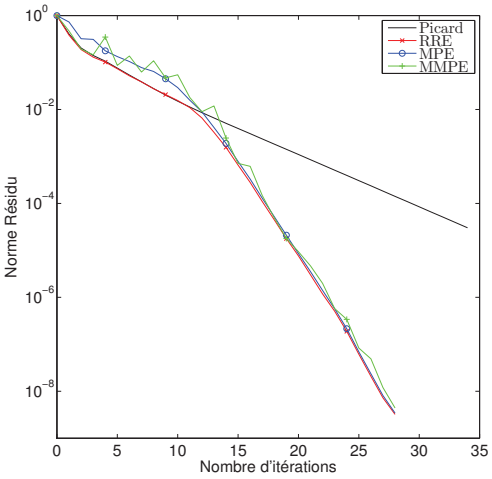
On observe que la stabilité des discrétisations entraîne une augmentation du nombre d'itérations de chaque méthodes. Le comportement de la convergence est la même pour toutes les discrétisations utilisées

3.5.2 Fluide s'écoulant dans un tunnel contenant un obstacle

On s'intéresse, dans cette exemple, à un fluide s'écoulant dans un tuyau contenant un obstacle. Comme dans l'exemple précédent, on suppose un écoulement de Poiseuille à l'entrée du conduit et des conditions de Neumann à la sortie. On fixe la longueur à 8. La taille du système est fixée à $N = 3200$. La figure 3.10 montre la solution du système et le rendu 3D de la pression pour le terme de viscosité valant $1/600$.

Pour cet exemple, on décide de comparer les méthodes de Newton et la méthode RRE cyclique pour différentes valeurs de ν ($\nu = 1/100$, $\nu = 1/200$, $\nu = 1/400$, $\nu = 1/600$ et $\nu = 1/800$). Les figures 3.11, 3.12 et 3.13 montrent les résultats. On note par $RRE - k$, la méthode RRE cyclique redémarrée toutes les k itérations, $Picard - Newton$ pour la méthode de Newton précédée par quelques itérations de la méthode de Picard. Pour la méthode RRE, on décide de représenter uniquement les deux meilleures convergences pour chaque courbe.

Les courbes montrent que, pour des valeurs de ν jusque $1/400$, la méthode de Newton converge immédiatement, c'est-à-dire que l'on n'utilise pas d'itérations de la méthode de Picard. La méthode RRE accélère la convergence de la méthode de



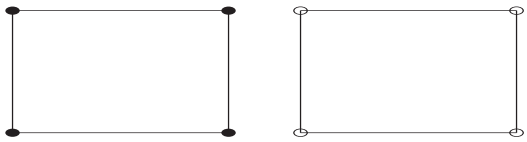
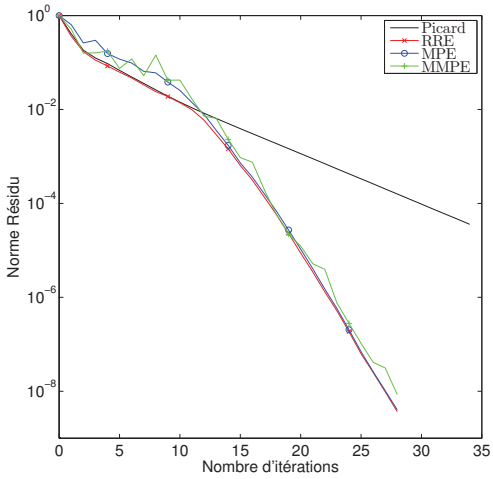


FIGURE 3.6 – Variation de la discrétisation $(Q_1 - Q_1)$



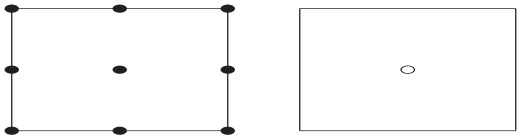
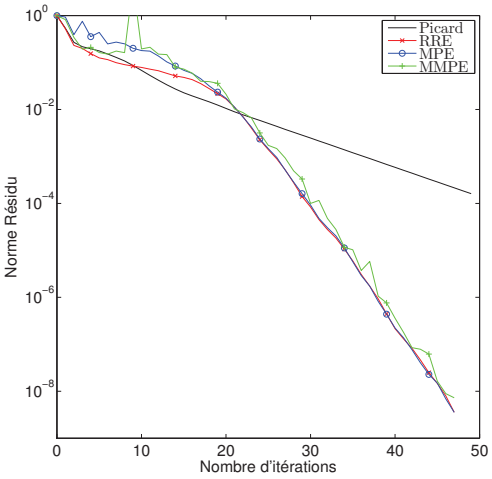


FIGURE 3.7 – Variation de la discrétisation ($Q_1 - P_0$)



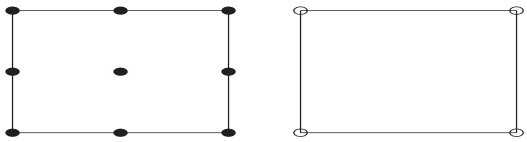
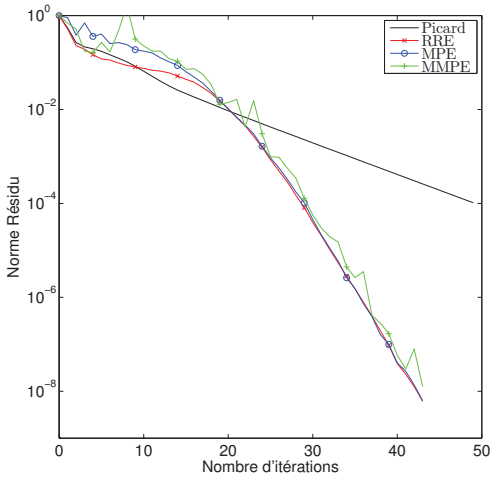


FIGURE 3.8 – Variation de la discrétisation $(Q_2 - Q_1)$



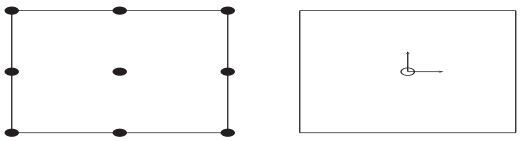


FIGURE 3.9 – Variation de la discrétisation $(Q_2 - P_{-1})$

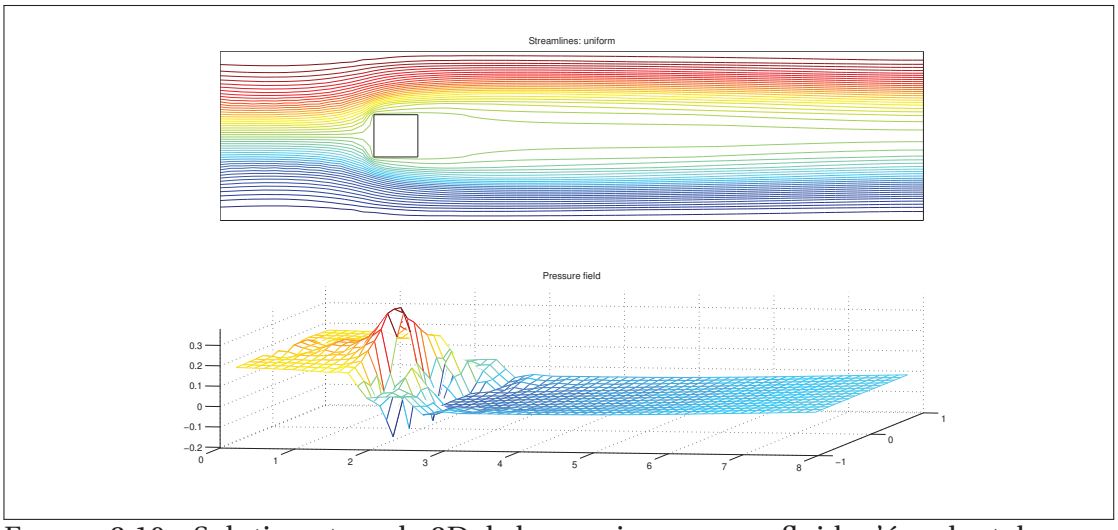
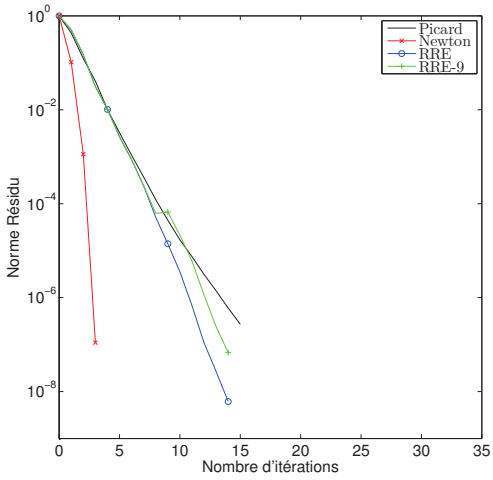


FIGURE 3.10 – Solution et rendu 3D de la pression pour un fluide s'écoulant dans un tunnel contenant un obstacle

Picard à partir de $\nu = 1/200$. À partir de $\nu = 1/600$, les itérations de la méthode de Newton ne convergent plus sans l'utilisation de quelques itérations de la méthode de Picard. La méthode de Picard, quant à elle, stagne. Pour $\nu = 1/800$, les itérations de la méthode de Picard-Newton ne convergent plus. Cela est dû à la non convergence de la méthode de Picard. Pour cette valeur de ν, seule la méthode redémarrée converge. On remarque que les meilleures convergences sont obtenues pour des valeurs de k petites.

3.6 Conclusion

Dans ce chapitre, on a présenté les équations de Navier-Stokes stationnaires dans le cas de fluides incompressibles. Nous en avons rappelé une formulation faible qui nous a permis d'obtenir une formulation matricielle et ainsi en déduire une méthode de résolution. Ces équations étant non linéaires, nous avons présenté deux méthodes de linéarisation : l'une correspondant à la méthode de Newton et l'autre à la méthode de Picard. L'inconvénient de la méthode de Picard est que sa convergence soit très lente. Nous avons donc décidé d'appliquer les méthodes d'extrapolation à la suite générée par la méthode de Picard. Les résultats montrent une accélération de la convergence dans tous les cas considérés. Ensuite, nous avons voulu comparer les méthodes cycliques et la méthode de Newton. En effet, il a été énoncé, par Silvester et al., que la méthode de Newton avait une convergence très rapide mais dans certains cas, l'utilisation de quelques itérations de la méthode de Picard était indispensable à cette convergence. On a remarqué que plus le terme de viscosité diminuait, plus la



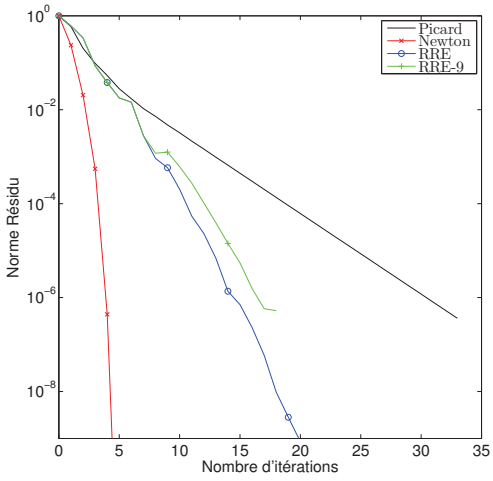
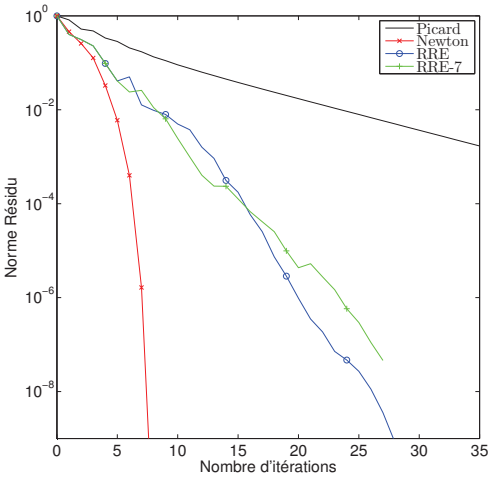


FIGURE 3.11 – Variation du terme de viscosité dans le cas d'un obstacle ($\nu = 1/100, 1/200$)



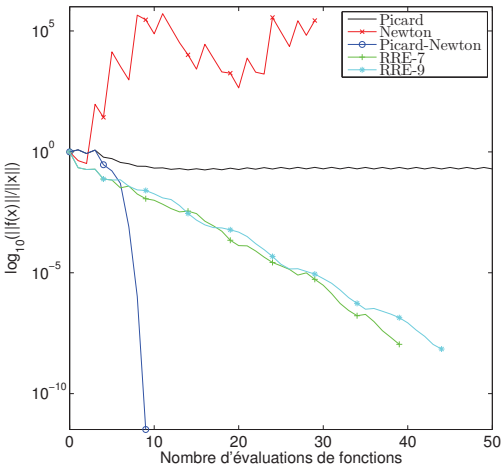


FIGURE 3.12 – Variation du terme de viscosité dans le cas d'un obstacle ($\nu = 1/400, 1/600$)

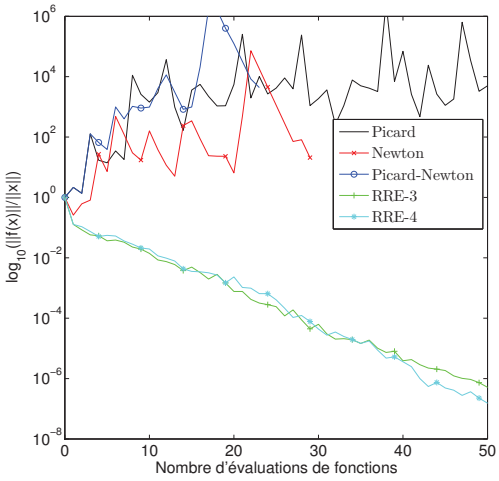


FIGURE 3.13 – Variation du terme de viscosité dans le cas d'un obstacle ($\nu = 1/800$)

convergence de la méthode de Picard-Newton s'allongeait, jusqu'à ne plus converger. Par contre, ce phénomène de non convergence n'a pas été observé pour les méthodes d'extrapolation cycliques.

Les résultats de ce chapitre ont été soumis à publication dans [21].

Chapitre 4

Application 2 : L'équation de Schrödinger

4.1 Introduction

Un des problèmes fondamentaux rencontrés aujourd'hui en chimie et en physique est de comprendre la dynamique des particules microscopiques. Il est possible d'expliquer ou de prédire certaines propriétés matérielles à l'échelle microscopique à partir des connaissances de leurs états initiaux et perturbations extérieurs connues. Par exemple, il peut être intéressant dans certains cas, de suivre la dynamique des atomes et des électrons jusqu'à un état stationnaire correspondant à une énergie totale minimum atteinte. L'équation de Schrödinger est une équation fondamentale en physique quantique non relativiste. Elle décrit l'évolution dans le temps d'une particule massive non relativiste et remplit ainsi le même rôle que la relation fondamentale de la dynamique en mécanique classique.

Dans ce chapitre, on s'intéressera au calcul de structures électroniques en résolvant la formulation dite de Kohn-Sham de l'équation de Schrödinger. La solution de cette équation peut être déterminée en résolvant un problème de valeurs propres non linéaire. Actuellement, le principe de résolution de ce problème est d'utiliser la méthode SCF (Self-Consistent Field) accélérée par la méthode d'Anderson et par la méthode de Broyden. Les détails de ces méthodes peuvent être trouvés dans [26, 48, 68, 69]. Notre but est d'utiliser les méthodes d'extrapolation à la place de ces méthodes d'accélération pour améliorer la vitesse de convergence.

Ce chapitre se présentera de cette façon. Dans un premier temps, on donnera la définition de l'équation de Schrödinger et les démarches pour aboutir à la formulation Kohn-Sham. Ensuite, on définira la méthode SCF et on donnera les définitions des méthodes d'accélération utilisées actuellement. Puis, on montrera les effets des méthodes

d'extrapolation sur la méthode SCF et on terminera par des résultats numériques.

4.2 L'équation de Schrödinger

L'ensemble des résultats et des notations de cette section se trouve dans [57]. La structure électronique d'une matière condensée, comme par exemple un liquide ou un solide, est décrite par une fonction d'onde Ψ qui peut être obtenue en résolvant l'équation de Schrödinger suivante :

$$H\Psi = E\Psi, \tag{4.1}$$

où H est un opérateur hamiltonien et E représente l'énergie totale du système. On suppose ici que l'équation est indépendante du temps. Dans sa forme originale, l'opérateur H est très complexe, comportant des sommes sur tous les électrons et les noyaux. Pour se donner une idée, considérons N noyaux de charge Z_n en position $\{R_n\}$ pour $n = 1, \cdots, N$ et M électrons en position $\{r_i\}$ pour $i = 1, \cdots, M$. L'opérateur H dans sa forme la plus simple peut être défini par :

$$\begin{aligned} H(R_1, R_2, \cdots; r_1, r_2, \cdots) = &- \sum_{n=1}^{N} \frac{\hbar^2 \nabla_n^2}{2M_n} + \frac{1}{2} \sum_{n,n'=1, n\neq n'}^{N} \frac{Z_n Z_{n'} e^2}{|R_n - R_{n'}|} \\ &- \sum_{i=1}^{M} \frac{\hbar^2 \nabla_i^2}{2m} - \sum_{n=1}^{N} \sum_{i=1}^{M} \frac{Z_n e^2}{|R_n - r_i|} + \frac{1}{2} \sum_{i,j=1, i\neq j}^{M} \frac{e^2}{|r_i - r_j|}, \end{aligned} \tag{4.2}$$

où M_n est la masse du noyau, $\hbar$ est la constante de Planck, m est la masse de l'électron et e est la charge de l'électron. Pour simplifier le problème, on utilise deux approximations fondamentales : l'approximation de Born-Oppenheimer et l'approximation qui suppose que les électrons n'interagissent pas entre eux. Pour avoir des informations sur ces simplifications voir [44, 55, 62]. Avec ces simplifications, on obtient une forme simplifiée de l'équation de Schrödinger connue sous le nom de formulation Kohn-Sham :

$$H(\rho(r))\Psi(r) = \left[\frac{-\hbar\nabla^2}{2m} + V_{tot}\left[\rho(r), r\right] \right] \Psi(r) = E\Psi(r), \tag{4.3}$$

où le Laplacien ∇^2 représente l'opérateur cinétique et V_{tot}, le potentiel total en tout point r de l'espace. Le potentiel dépend de la densité de charge ρ qui sera définie dans la suite. Le potentiel V_{tot} est la somme de trois composantes : le potentiel ionique V_{ion}, qui reflète l'énergie provenant du cœur des électrons, le potentiel de Hartree V_H, qui représente l'énergie de Coulomb entre les électrons et enfin le potentiel V_{XC}, qui naît de l'approximation faite sur chaque électron.

$$V_{tot} = V_{ion} + V_H + V_{XC}. \tag{4.4}$$

Les termes V_{XC} et V_H dépendent de la densité de charge $\rho(r)$ qui dépend des fonctions d'ondes comme le montre la formule suivante :

$$\rho(r) = \sum_{i=1}^{M} |\Psi_i(r)|^2. \tag{4.5}$$

Le potentiel V_{XC} est facilement approché par un potentiel venant de l'expression de la densité locale. Lorsque la densité de charge $\rho(r)$ est connue, le potentiel Hartree s'obtient en résolvant l'équation de Poisson :

$$\nabla^2 V_H = -4\pi\rho(r).$$

Les potentiels V_H et V_{XC} sont tous les deux représentés par des matrices diagonales pour la forme discrète de notre problème. Le potentiel ionique est plus complexe. Pour obtenir des détails sur son expression, voir [58]. D'après (4.5), ρ peut être calculée à partir des fonctions Ψ. Les fonctions d'ondes Ψ sont solutions du problème aux valeurs propres (4.3), dont les coefficients dépendent du potentiel. Le calcul principal de notre problème est de résoudre de façon répétée un problème aux valeurs propres de grande taille et symétrique. On peut aussi voir ça comme un problème aux valeurs propres non linéaire, où la non linéarité est traitée par une itération de SCF. Dans l'algorithme 19, nous donnons la méthode la plus simple pour résoudre notre problème. Nous appellerons cette méthode "simplemix". La quantité ω utilisée dans l'algorithme est une constante donnée pour notre méthode simplemix. Notre but pour ce problème est d'explorer la possibilité d'accélérer la méthode en utilisant les méthodes d'extrapolation définies dans le chapitre 2.

Sous forme matricielle, le Hamiltonien H, défini dans l'équation (4.3), est la somme de cinq matrices : la matrice du Laplacien, trois matrices diagonales provenant de la discrétisation de V_H, V_{XC} et de la partie locale du potentiel ionique et une matrice correspondant à la partie non locale et qui est la somme de simples mises à jour de rang un sur tous les atomes et les nombres quantiques.

Nous pouvons aussi exprimer la méthode SCF comme une méthode de point fixe pour l'équation suivante :

$$s_{k+1} = G(s_k), \tag{4.6}$$

où $G(s)$ représente le nouveau potentiel obtenu en résolvant (4.3), en utilisant (4.4) et en posant $s = V_{tot}$. Comme toutes les méthodes de points fixes, la méthode SCF est souvent moins efficace que nous le souhaitons : la convergence peut être lente. Pour améliorer cette convergence, nous allons maintenant nous intéresser aux méthodes qui sont utilisées actuellement. Pour cela, nous remplaçons l'équation $V_{tot} = V_{tot} + \omega(\tilde{V}_{tot} - V_{tot})$ dans l'algorithme 19 par $V_{tot} = \Phi(\tilde{V}_{tot}, V_{tot})$ correspondant à des méthodes quasi-Newton. Les méthodes de type Broyden et la méthode d'Anderson sont utilisées pour améliorer la convergence. Commençons par rappeler les définitions et les algorithmes de celles-ci.

Algorithme 19: Algorithme de la méthode SCF

Entrée : $\rho(r)$;
Évaluer $V_{tot}(\rho(r), r)$;
pour *iter*=1,2,...,*Maxiter* **faire**
  Calculer $H(\rho(r)) = \left[\frac{-\bar{h}\nabla^2}{2m} + V_{tot}\left(\rho(r), r\right)\right]$;
  Résoudre $H(\rho(r))\psi_i(r) = E_i\psi_(r)$ pour $i = 1,2,\cdots$;
  Calculer $\rho(r)_{new}$ avec (4.5);
  Calculer $\rho(r) = \sum_{occupied\ states}\left|\psi_i(r)\right|^2$;
  Résoudre $\nabla^2 V_H(r) = -4\pi\rho(r)$;
  Mettre à jour V_{XC} et V_{ion};
  Calculer $\tilde{V}_{tot}(\rho(r), r) = V_{ion}(r) + V_H(\rho(r), r) + V_{XC}(\rho(r), r)$;
  Si $\frac{\|\tilde{V}_{tot} - V_{tot}\|}{\|\tilde{V}_{tot}\|} < tol$ stop ;
  Sinon, calculer $V_{tot} = V_{tot} + \omega(\tilde{V}_{tot} - V_{tot})$;
fin

4.3 Méthodes de Résolution connues

Dans [26], Fang et Saad montrent que la convergence est meilleure pour l'une ou l'autre des méthodes selon l'exemple considéré. Nous utiliserons donc ces deux méthodes dans les expériences numériques que nous réaliserons plus tard dans ce chapitre. Commençons donc par les définir et par donner leur algorithme respectif.

4.3.1 Méthode de Broyden

Introduction

Introduisons l'approximation linéaire de F en s,

$$F(s + \Delta s) \approx F(s) + J(s)\Delta s, \tag{4.7}$$

où $J(s)$ représente la matrice Jacobienne de F en s. Nous rappelons que la fonction F vérifie $F(s_*) = 0$. La méthode de Newton utilise cette approximation pour déterminer une correction Δs_k de la solution approximative s_k. Δs_k est obtenu en résolvant

$$J(s_k)\Delta s_k = -F(s_k), \tag{4.8}$$

$$s_{k+1} = s_k + \Delta s_k, \tag{4.9}$$

pour $k = 1,2,3,\ldots$. Les itérations sont réalisées jusqu'à ce que l'on obtienne une approximation précise de la solution, en supposant que la méthode de Newton converge.

La méthode de Newton, comme nous l'avons dit dans le chapitre 1, requiert le calcul de la matrice Jacobienne à chaque itération. Ce calcul est très coûteux. Pour pallier à ce problème, les méthodes quasi-Newton utilisent une approximation de $J(s_k)$, notée J_k qui est plus intéressante à calculer. Broyden [13, 41] propose de poser J_{k+1}, une modification de rang un de J_k. En fait, Broyden suggère que l'équation (4.8) soit remplacée par :

$$J_{k+1}\Delta s_k = \Delta f_k, \tag{4.10}$$

où $\Delta f_k = F(s_{k+1}) - F(s_k)$ et

$$J_{k+1} = J_k + \left(\Delta f_k - J_k \Delta s_k\right) \frac{\Delta s_k^T}{\Delta s_k^T \Delta s_k}. \tag{4.11}$$

Dans ce cas, J_{k+1} satisfait

$$J_{k+1} q = J_k q \quad \forall q \text{ vérifiant } q^t \Delta s_k = 0. \tag{4.12}$$

Résultats de convergence

Dans cette section, nous donnons le théorème présentant la convergence superlinéaire de la méthode de Broyden.

Théorème 4.1

Supposons que F soit continûment différentiable dans un domaine ouvert convexe $\mathscr{D} \subset \mathbb{R}^n$ contenant la solution s_*, que $F(s_*) = 0$ et que la matrice jacobienne $J(s_*)$ au point s_* soit non singulière. Supposons de plus, que la matrice jacobienne J soit continue K-lipschitzienne dans $\mathscr{D}$. Alors la fonction de mise à jour $\mathscr{B}(t, B) = \{\bar{B}\}$, où

$$\bar{B} = B + \frac{(y - Bt)\, t^T}{t^T t} \tag{4.13}$$

est bien définie pour tout $t \neq 0$, dans un voisinage V de $(s_*, J(s_*))$, et l'itération correspondante

$$s_{k+1} = s_k - B_k^{-1} F(s_k) \tag{4.14}$$

avec $B_{k+1} \in \mathscr{B}(s_k, B_k)$, $k \geq 0$, a une convergence super-linéaire locale vers s_*.

Implémentation de la méthode de Broyden

La matrice de Broyden peut être stockée en utilisant uniquement k vecteurs et la matrice initiale B_0, où k est le nombre d'itérations effectuées [41]. La base de cette

idée est d'utiliser la formule de Shermann-Morrison, définie dans la proposition 1, pour inverser une mise à jour de rang un.

L'algorithme de Broyden, défini par Kelley, est donné dans [41, 43]. Nous allons rappeler la démarche utilisée. Il est tout d'abord facile de voir que la mise à jour de Broyden (4.11) peut être exprimée sous la forme suivante :

$$B_{k+1} = B_k + u_k v_k^T, \tag{4.15}$$

avec

$$u_k = \frac{F(s_{k+1})}{\|\Delta s_k\|} \text{ et } v_k = \frac{\Delta s_k}{\|\Delta s_k\|}. \tag{4.16}$$

À l'aide de la formule de Shermann-Morrison (1.1), nous avons :

$$B_{k+1}^{-1} = \left(I - \frac{(B_k^{-1} u_k) v_k^T}{1 + v_k^T B_k^{-1} u_k} \right) B_k^{-1}. \tag{4.17}$$

Définissons

$$w_k = \frac{B_k^{-1} u_k}{1 + v_k^T B_k^{-1} u_k} = \frac{B_k^{-1} F(s_{k+1})}{\|\Delta s_k\| + v_k^T B_k^{-1} F(s_{k+1})}. \tag{4.18}$$

Alors si on pose $B_0 = I$, on obtient :

$$\begin{array}{rcl} B_k^{-1} & = & (I - w_{k-1} v_{k-1}^T)(I - w_{k-2} v_{k-2}^T) \cdots (I - w_0 v_0^T) \\ & = & \prod_{j=0}^{k-1} (I - w_j v_j^T), \end{array} \tag{4.19}$$

et le pas Δs_k peut être calculé, en stockant $2k$ vecteurs w_j, v_j, $j = 0, \cdots, k-1$, par

$$\Delta s_k = -B_k^{-1} F(s_k) = -\prod_{j=0}^{k-1} (I - w_j v_j^T) F(s_k). \tag{4.20}$$

On peut montrer qu'il n'est pas nécessaire de stocker la suite w_k. En effet, les calculs de w_{k-1} et Δs_k peuvent être combinés :

$$\Delta s_k = -B_k^{-1} F(s_k) = -(I - w_{k-1} v_{k-1}^T) B_{k-1}^{-1} F(s_k) = -(I - w_{k-1} v_{k-1}^T) z, \tag{4.21}$$

où

$$z = B_{k-1}^{-1} F(s_k) = \prod_{j=0}^{k-2} (I - w_j v_j^T) F(s_k). \tag{4.22}$$

L'équation (4.18) donne

$$w_{k-1} = \frac{B_{k-1}^{-1} u_{k-1}}{1 + v_{k-1}^T B_{k-1}^{-1} u_{k-1}} = \frac{z}{\|\Delta s_{k-1}\| (1 + v_{k-1}^T z / \|\Delta s_{k-1}\|)} = \alpha^{-1} z, \tag{4.23}$$

où $\alpha = \|\Delta s_{k-1}\| + v_{k-1}^T z$. Par conséquent,

$$\begin{aligned} \Delta s_k &= -(I - w_{k-1} v_{k-1}^T) z = -z(1 - \alpha^{-1} v_{k-1}^T z) \\ &= -z(1 - \alpha^{-1}(\alpha - \|\Delta s_{k-1}\|)) = -\alpha^{-1}\|\Delta s_{k-1}\| z = -\|\Delta s_{k-1}\| w_{k-1}. \end{aligned} \tag{4.24}$$

Le pas de Broyden peut donc s'écrire :

$$\Delta s_k = -\prod_{j=0}^{k-1} \left(I + \frac{\Delta s_{j+1} \Delta s_j^T}{\|\Delta s_j\|^2} \right) F(s_k). \tag{4.25}$$

Par conséquent, nous avons besoin de stocker uniquement les pas $\{\Delta s_k\}$ et leurs normes pour construire Δs_k. Comme Δs_k apparaît dans les deux membres de l'équation (4.25), on ne peut pas appliquer directement cette formule. Par contre si on modifie (4.25) on obtient :

$$\begin{aligned} \Delta s_k &= -\left(I + \frac{\Delta s_k \Delta s_{k-1}^T}{\|\Delta s_{k-1}\|^2} \right) \prod_{j=0}^{k-2} \left(I + \frac{\Delta s_{j+1} \Delta s_j^T}{\|\Delta s_j\|^2} \right) F(s_k) \\ &= -\left(I + \frac{\Delta s_k \Delta s_{k-1}^T}{\|\Delta s_{k-1}\|^2} \right) B_{k-1}^{-1} F(s_k). \end{aligned} \tag{4.26}$$

En résolvant l'équation (4.26), on aboutit à :

$$\Delta s_k = -\frac{B_{k-1}^{-1} F(s_k)}{1 + \frac{\Delta s_{k-1}^T B_{k-1}^{-1} F(s_k)}{\|\Delta s_{k-1}\|^2}}. \tag{4.27}$$

On a supposé, dans cette méthode, que $B_0 = I$. Nous allons montrer que ce choix n'est pas restrictif. En effet, nous avons la proposition suivante :

Proposition 9. *Soit $\{s_k, B_k\}$ la suite de Broyden générée par (F, s_0, B_0) et soit $\{z_k, C_k\}$ la suite de Broyden générée par $(B_0^{-1}F, s_0, I)$. Alors*

$$s_k = z_k \text{ et } B_k = B_0 C_k. \tag{4.28}$$

La preuve de cette proposition se trouve dans [41]. L'algorithme 20 reprend la méthode de Broyden que nous venons d'exposer. Dans cet algorithme, l'indice k désigne le nombre d'itérations effectuées et l'indice i désigne le nombre de mises à jour de la matrice.

4.3.2 Méthode de Broyden généralisée

La méthode de Broyden généralisée a été développée par Fang et Saad dans [26]. Dans cette section, nous donnons la définition de cette méthode comme la donnent

Algorithme 20: Algorithme de la méthode de Broyden

Entrée : F, s_0, $kmax$, $imax$ et tol ;
Calculer $r_0 = \|F(s_0)\|$. Poser $i = -1$ et $\Delta s_0 = -F(s_0)$;
pour $k = 1 : kmax$ **faire**
  Poser $i = i + 1$;
  Calculer $s_k = s_{k-1} + \Delta s_i$;
  Calculer $F(s_k)$;
  Si $\|F(s_k)\| < tol$ exit;
  Sinon, poser $z = -F(s_k)$;
  si $i < imax$ **alors**
    Poser $z = -F(s_k)$;
    pour $j = 0 : i - 1$ **faire**
      Calculer $z = z + \frac{\Delta s_j \Delta s_j^T z}{\|\Delta s_{j-1}\|^2}$;
    fin
    Calculer $\Delta s_k = \frac{z}{1 + \frac{\Delta s_{i-1}^T z}{\|\Delta s_{i-1}\|^2}}$;
    Si $i = imax$, poser $i = -1$ et $\Delta s_0 = -F(s_k)$;
  fin
fin

Fang et Saad. Supposons que les $m+1$ plus récentes itérations $\{s_{k-m}, s_{k-m+1}, \cdots, s_k\}$ soient disponibles. Posons

$$\Delta s_i = s_{i+1} - s_i, \qquad \Delta f_i = F(s_{i+1}) - F(s_i), \qquad i = k-m, k-m+1, \cdots, k-1.$$

Nous allons décrire la méthode de Broyden généralisée avec un rang flexible pour calculer l'approximation de la Jacobienne J_{k+1}. Cette approximation satisfait les m équations suivantes :

$$J_k \Delta s_i = \Delta f_i, \qquad i = k-m, k-m+1, \cdots, k-1, \tag{4.29}$$

où les vecteurs $\Delta f_{k-m}, \Delta f_{k-m+1}, \cdots, \Delta f_{k-1}$ sont supposés être linéairement indépendant et $m \leq N$. On peut réécrire ces équations sous forme matricielle

$$J_k \Delta S_k = \Delta F_k, \tag{4.30}$$

où

$$\Delta S_k = [\Delta s_{k-m}, \Delta s_{k-m+1}, \cdots, \Delta s_{k-1}], \qquad \Delta F_k = \left[\Delta f_{k-m}, \Delta f_{k=m+1}, \cdots, \Delta f_{k-1}\right].$$

De la même façon, on définit une condition analogue à (4.12)

$$(J_k - J_{k-m})q = 0$$

pour tout q orthogonal à l'image de ΔS_k. Comme on a la relation

$$Ker(X)^{\perp} = Im(X^T),$$

cela est équivalent à :

$$(J_k - J_{k-m})^T = \Delta S_k Z^T$$

pour une certaine matrice Z. Cette matrice est obtenue grâce à l'équation (4.30). En effet, si on multiplie $(J_k - J_{k-m}) = Z\Delta S_k$ par ΔS_k à droite, on obtient :

$$(J_k - J_{k-m})\Delta S_k = Z\Delta S_k \Delta S_k. \tag{4.31}$$

Si on suppose que ΔS_k est de rang maximum, on a :

$$Z = (J_k - J_{k-m})\Delta S_k (\Delta S_k \Delta S_k)^{-1}. \tag{4.32}$$

Cela mène à une formule de mise à jour de rang m donnée par :

$$J_k = J_{k-m} + (\Delta F_k - J_{k-m}\Delta S_k)(\Delta S_k^T \Delta S_k)^{-1} \Delta S_k^T.$$

La première approximation est obtenue en posant

$$J_1 = \omega I,$$

où ω est le paramètre de mixage. Le nouvel itéré est calculé en posant

$$s_{k+1} = s_k - \Delta s_k.$$

4.3.3 Méthode d'Anderson

Soit $F : \mathbb{R}^N \to \mathbb{R}^N$ une fonction non linéaire. La méthode d'Anderson est une procédure itérative pour déterminer la solution d'un système d'équations non linéaires de la forme $F(s) = 0$. Notons $s_{k-m}, \cdots, s_k \in \mathbb{R}^N$, les itérées connues et $f_{k-m}, \cdots, f_k \in \mathbb{R}^N$, les images correspondantes par la fonction F. Supposons que l'évaluation de $F(s)$ soit très coûteuse ou que l'on ne connaisse pas la fonction F explicitement. La méthode d'Anderson [2, 26, 48, 69, 68] permet de déterminer s_{k+1} en utilisant le principe suivant :

$$\bar{s}_k = s_k - \sum_{i=k-m}^{k-1} \gamma_i^{(k)} \Delta s_i = s_k - \Delta S_k \gamma^{(k)}, \tag{4.33}$$

$$\bar{f}_k = f_k - \sum_{i=k-m}^{k-1} \gamma_i^{(k)} \Delta f_i = f_k - \Delta F_k \gamma^{(k)}, \tag{4.34}$$

où

$$\Delta s_i = s_{i+1} - s_i,$$
$$\Delta f_i = f_{i+1} - f_i,$$
$$\gamma^{(k)} = \left[\gamma_{k-m}^{(k)}, \cdots, \gamma_{k-1}^{(k)}\right]$$

et

$$\Delta S_k = [\Delta s_{k-m}, \cdots, \Delta s_{k-1}],$$
$$\Delta F_k = \left[\Delta f_{k-m}, \cdots, \Delta f_{k-1}\right].$$

En réarrangeant, nous obtenons :

$$\bar{s}_k = \sum_{i=k-m}^{k} \omega_j s_i, \quad \bar{f}_k = \sum_{i=k-m}^{k} \omega_j f_i,$$

avec $\sum_{i=k-m}^{k} \omega_j = 1$. Les quantités $\bar{s}_k$ et $\bar{f}_k$ peuvent être considérées comme des moyennes pondérées de $s_{k-m}, \cdots, s_k$ et $f_{k-m}, \cdots, f_k$, respectivement. $\gamma^{(k)}$ est déterminé en minimisant

$$E(\gamma^{(k)}) = \left\langle \bar{f}_k, \bar{f}_k \right\rangle = \left\| f_k - \Delta F_k \gamma^{(k)} \right\|_2^2.$$

La solution de cette dernière équation satisfait l'équation normale suivante :

$$(\Delta F_k^T \Delta F_k) \gamma^{(k)} = \Delta F_k^T f_k.$$

Le nouvel itéré s_{k+1} est calculé en posant

$$\begin{aligned} s_{k+1} &= \bar{s}_k + \beta \bar{f}_k \\ &= s_k + \beta f_k - (\Delta S_k + \beta \Delta F_k) \gamma^{(k)} \\ &= s_k + \beta f_k - (\Delta S_k + \beta \Delta F_k)(\Delta F_k^T \Delta F_k)^{-1} \Delta F_k^T f_k. \end{aligned}$$

Méthodes	Broyden généralisée	Anderson
Multiplications et Additions	$2nm^2$	$2nm^2$
Stockage	$2nm$	$2nm$

TABLE 4.1 – Coût des méthodes de Broyden généralisée et d'Anderson

4.3.4 Coût des méthodes

Pour les deux méthodes que l'on vient de considérer (Broyden généralisée et Anderson), nous donnons maintenant le détail des coûts. Le tableau 4.1 résume le nombre d'opérations et le stockage mémoire de celles-ci.

Si l'on compare ce coût avec les méthodes d'extrapolation, on remarque que le coût en nombre d'opérations est sensiblement le même que pour les méthodes MPE et RRE. On remarque aussi que les méthodes de Broyden généralisée et d'Anderson ont besoin de stocker plus de données.

4.4 Résultats Numériques

Nous allons maintenant donner quelques résultats numériques montrant l'accélération de la convergence de méthodes d'extrapolation par rapport à la méthode simplemix et comparer ensuite les méthodes cycliques avec les méthodes de Broyden généralisées et la méthode de Anderson. Tous les résultats présentés maintenant ont été réalisés en utilisant MATLAB 7.1 et le programme RSDFT (real density functional theory). RSDFT calcule les propriétés de la structure électronique des molécules. Il se base sur les travaux de Chelikowsky et al. de l'université du Texas (Austin) en collaboration avec Y. Saad et ses collègues de l'université de Minnesota. Ils ont mis au point un programme nommé PARSEC (Pseudopotential Algorithms for Real Space Eigenvalue Calculations) sur lequel RSDFT se base. Les algorithmes de RSDFT sont plus simples que ceux utilisés dans PARSEC pour pouvoir utiliser de nouveaux algorithmes.

Pour tous les tests pratiqués ici, les itérations se terminent lorsque

$$\frac{\|F(s)\|_2}{\|s\|_2} \leq tol$$

ou lorsque le nombre d'itérations devient trop important.

4.4.1 Accélération de la convergence

Nous allons explorer le potentiel d'accélération des itérations de simplemix en utilisant les méthodes RRE, MPE et MMPE. Les itérations de la méthode simplemix sont construites à partir de l'algorithme SCF 19 pour différentes valeurs de ω. Les

figures 4.1 à 4.9 montrent le comportement de la norme résiduelle, en utilisant l'échelle logarithmique, pour trois problèmes de molécules différentes.

Le premier problème traite le cas de l'atome de Carbone (C). La taille du système à résoudre est $n = 54872$. Les courbes des figures 4.1, 4.2 et 4.3 montrent les résultats obtenus pour des valeurs de ω allant de 0.1 à 0.5. Pour tous ces tests, on obtient une accélération de la convergence pour les méthodes d'extrapolation. Celle-ci est plus ou moins importante selon la valeur de ω. On remarque que les meilleures convergences sont obtenues lorsque $\omega = 0.3$ et 0.4. Les courbes des méthodes d'extrapolation sont comparables. Seule la méthode MMPE présente quelques irrégularités qui sont assez rapidement rattrapées.

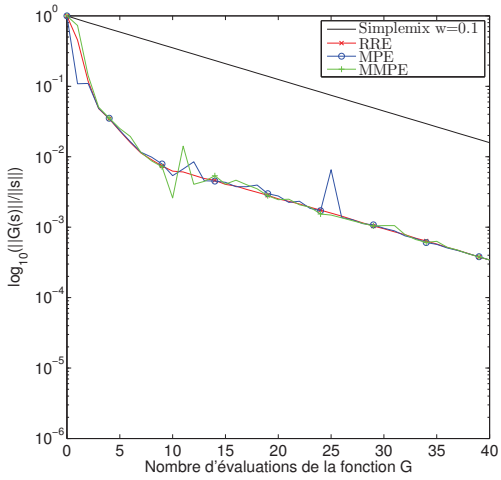
De la même façon, appliquons maintenant la méthode simplemix et les méthodes d'extrapolation à la molécule Silice Sodium (SiNa). Cette fois, la taille du système est $n = 74088$. Les courbes des figures 4.4, 4.5 et 4.6 montrent les résultats obtenus. Comme pour l'exemple précédent, on observe bien l'accélération de la convergence pour les méthodes d'extrapolation. Cette fois-ci, les méthodes d'extrapolation ont des comportements quelque peu différents lorsque $\omega \leq 0.3$. En effet, on peut remarquer que la méthode RRE converge plus rapidement, tandis que les méthodes MPE et MMPE sont semblables. Ce phénomène s'atténue pour $\omega > 0.3$ avec un comportement similaire de toutes les méthodes d'extrapolation.

Pour ce dernier exemple, on s'intéresse à la molécule de monoxyde de Carbone (CO). La taille du système est $n = 157464$. Les courbes des figures 4.7, 4.8 et 4.9 montrent le comportement du résidu non linéaire pour des valeurs de ω allant de 0.1 à 0.5. Le comportement est différent pour cet exemple. En effet, on observe une accélération de la convergence des méthodes d'extrapolation pour $\omega \leq 0.3$. Par contre, pour $\omega = 0.4$, la convergence de toutes les méthodes, y compris la méthode simplemix, est similaire. Pour $\omega = 0.5$, on obtient la non convergence des méthodes d'extrapolation. Ce phénomène s'observe pour toutes les méthodes.

4.4.2 Méthode cyclique

Nous allons maintenant comparer les méthodes d'extrapolation redémarrée avec la méthode de Broyden généralisée et la méthode de Anderson définies précédemment. Pour tous les tests, on notera RRE-k, MPE-k et MMPE-k, les méthodes RRE, MPE et MMPE redémarrées après k itérations, Broyden, la méthode de Broyden généralisée et Anderson, la méthode de Anderson. Nous utiliserons les mêmes atomes que dans le paragraphe précédent. Nous allons comparer toutes ces méthodes pour différents choix de ω.

Les courbes des figures 4.10 à 4.12 montrent les résultats obtenus dans le cas de l'atome de Carbone pour ω variant de 0.1 à 0.6. Sur toutes les courbes, on observe que la méthode de Anderson possède la convergence la moins rapide. Ce phénomène s'observe aussi pour la méthode de Broyden généralisée sauf dans le cas $\omega = 0.6$. Pour



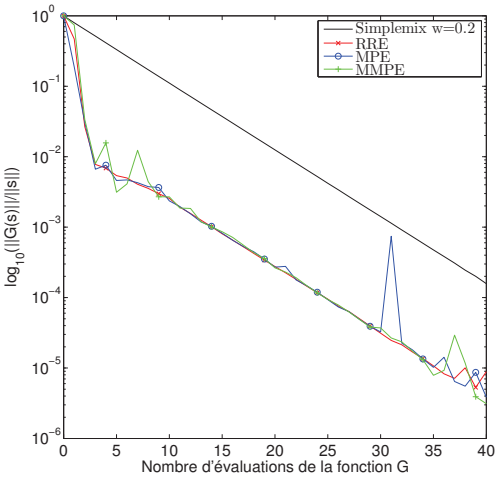
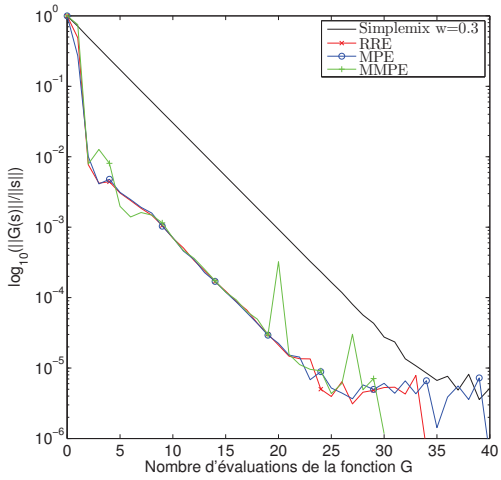


FIGURE 4.1 – Accélération de la convergence pour l'atome C, $\omega = 0.1, 0.2$



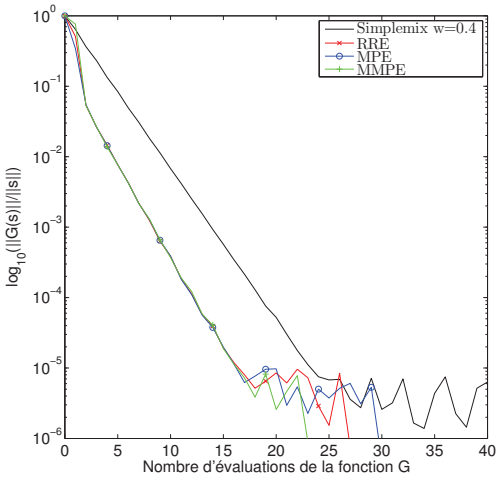


FIGURE 4.2 – Accélération de la convergence pour l'atome C, $\omega = 0.3, 0.4$

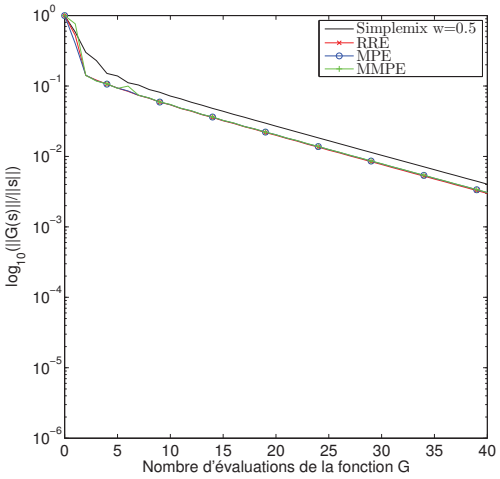
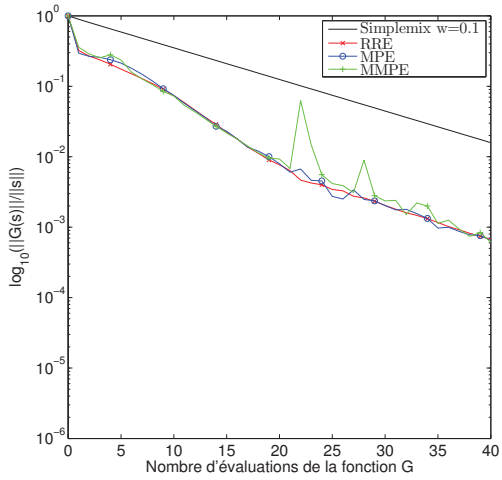


FIGURE 4.3 – Accélération de la convergence pour l'atome C, $\omega = 0.5$



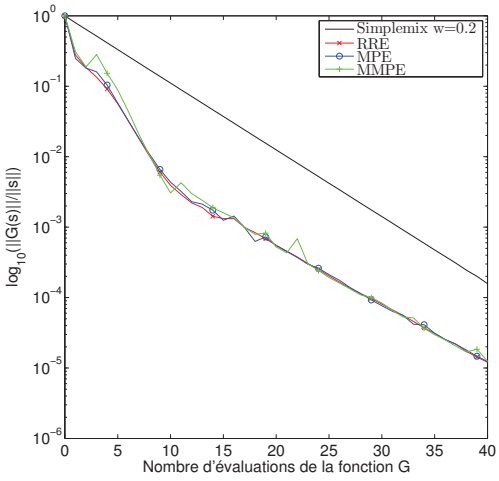
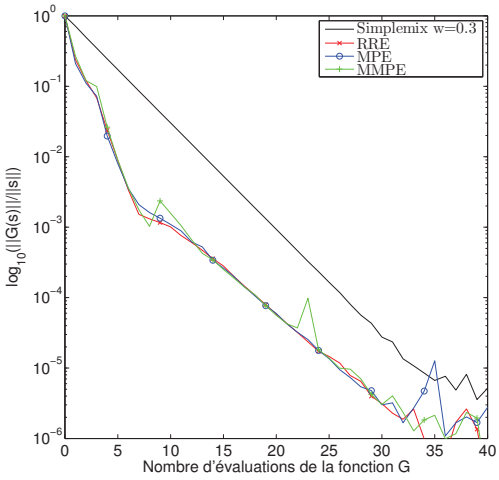


FIGURE 4.4 – Accélération de la convergence pour la molécule $SiNa$, $\omega = 0.1, 0.2$



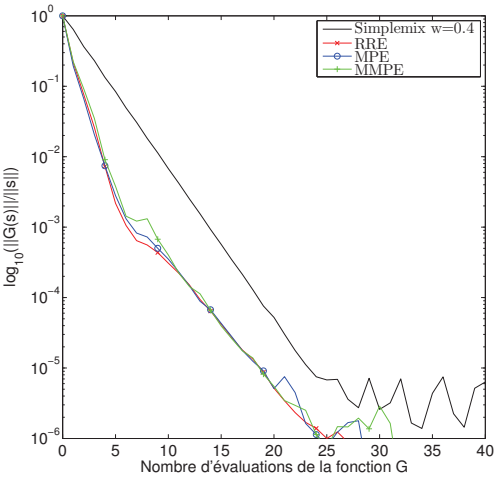


FIGURE 4.5 – Accélération de la convergence pour la molécule $SiNa$, $\omega = 0.3, 0.4$

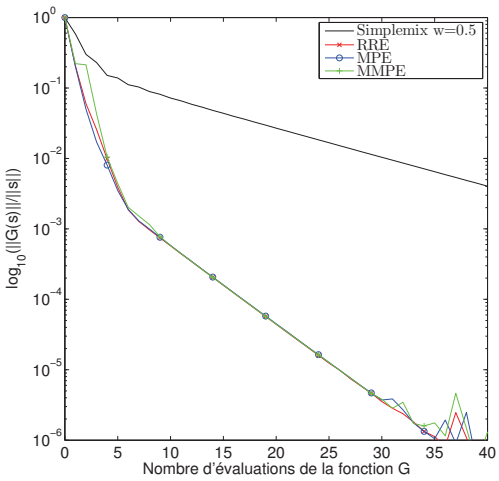
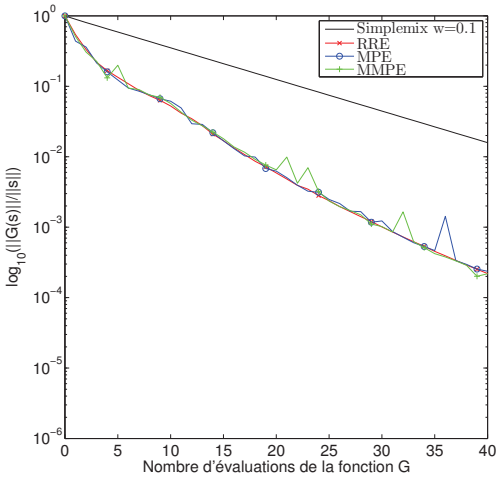


FIGURE 4.6 – Accélération de la convergence pour la molécule $SiNa$, $\omega = 0.5$



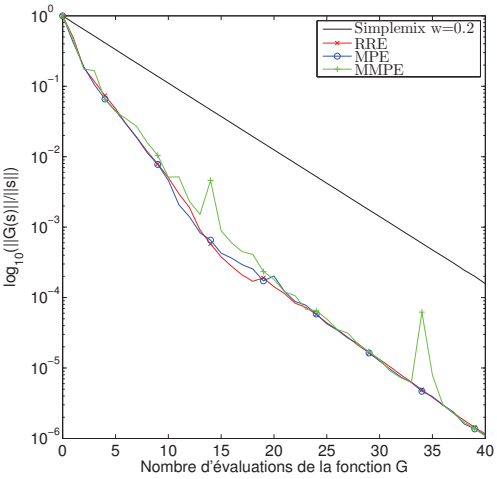
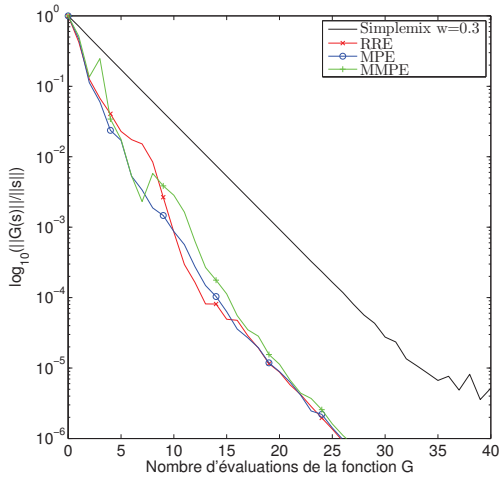


FIGURE 4.7 – Accélération de la convergence pour la molécule CO, $\omega = 0.1, 0.2$



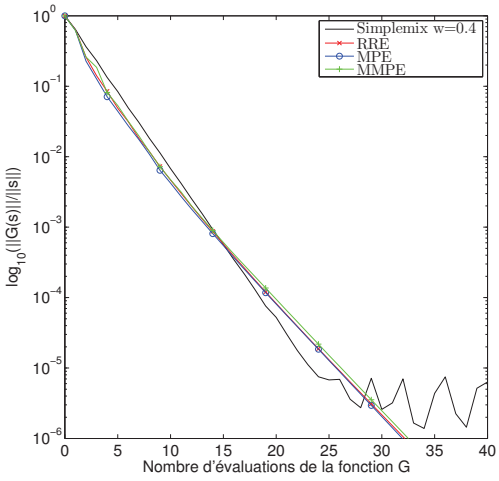


FIGURE 4.8 – Accélération de la convergence pour la molécule CO, $\omega = 0.3, 0.4$

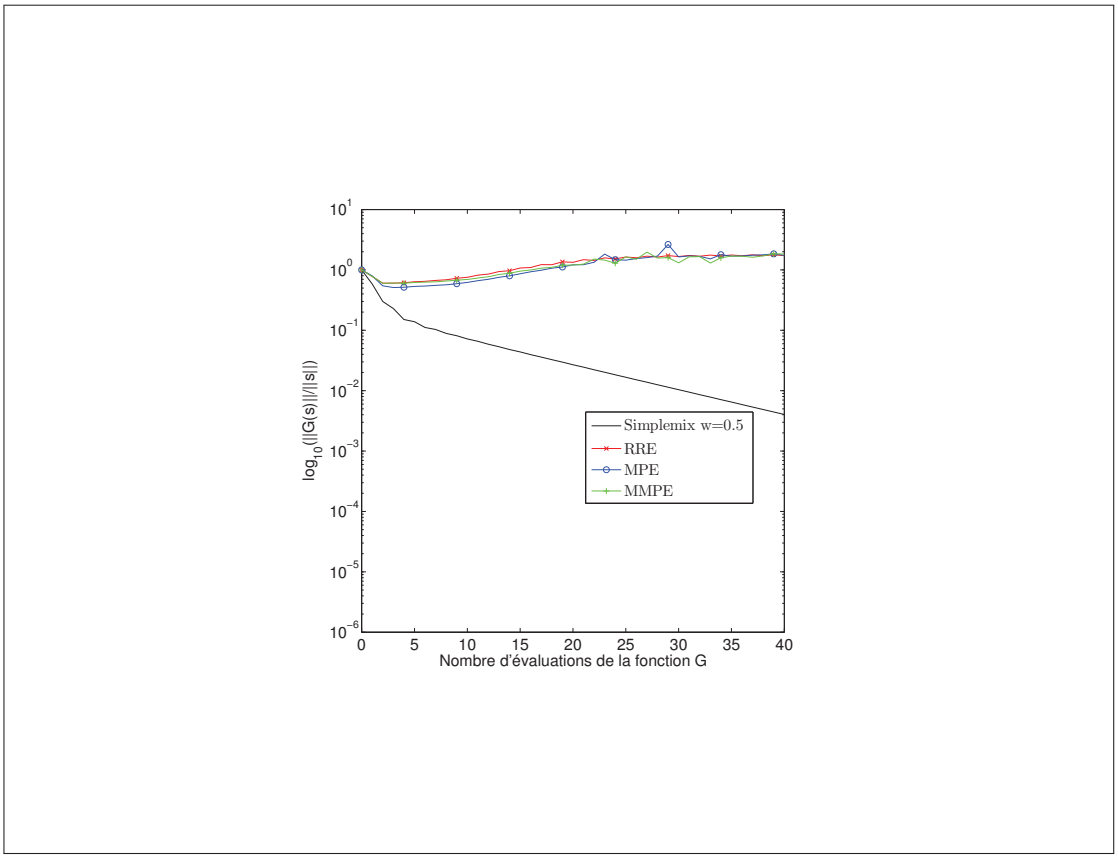


FIGURE 4.9 – Accélération de la convergence pour la molécule CO, $\omega = 0.5$

les autres méthodes, le comportement est différent en fonction de la valeur de ω. Dans pratiquement tous les cas, les méthodes d'extrapolation redémarrée convergent plus rapidement en nombre d'itérations.

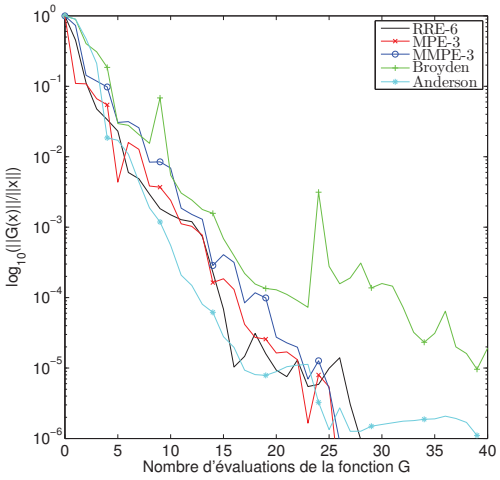
Les courbes des figures 4.13, 4.14 et 4.15 montrent les résultats obtenus pour le cas de la molécule $SiNa$. Pour cet exemple, on peut remarquer que la méthode de Broyden généralisée converge mieux pour $\omega = 0.3$, $\omega = 0.5$ et $\omega = 0.6$. Sinon la convergence est moins bonne que les autres méthodes. La méthode de Anderson converge aussi bien que les méthodes d'extrapolation pour $\omega = 0.3$ et $\omega = 0.4$. Les méthodes d'extrapolation ont toujours une bonne convergence, presque comparable aux deux autres méthodes dans le cas où celles-ci convergent bien, meilleures dans les autres cas. La convergence des méthodes de Broyden généralisée et de Anderson finissent toujours par stagner après une étape d'accélération qui est comparable aux méthodes d'extrapolation.

Pour le dernier test, on reprend l'exemple de la molécule CO. Les figures 4.16 à 4.18 montrent les résultats obtenus pour ω variant de 0.1 à 0.6. Contrairement aux résultats obtenus pour le processus non redémarré, les méthodes d'extrapolation convergent toutes cette fois-ci. Le comportement des méthodes de Broyden généralisée et d'Anderson est le même que pour les autres exemples.

4.5 Conclusion

Dans ce chapitre, nous avons testé l'efficacité des méthodes d'extrapolation pour la résolution d'un problème de valeurs propres non linéaire. Nous avons considéré la formulation Khon-Sham de l'équation de Schrodinger. Les études précédentes utilisaient les méthodes de Broyden et de Anderson pour accélérer la convergence. Sur plusieurs tests, nous avons observé le potentiel d'accélération des méthodes d'extrapolation classiques en considérant une suite de points fixes, notée ici simplemix. Puis nous avons comparé les méthodes d'accélération aux méthodes d'extrapolation cycliques. Les résultats ont montré que les méthodes d'extrapolation se comportaient très bien, voire mieux sur la plupart des tests. Une utilisation de ces méthodes dans le programme PARSEC permettrait de pouvoir tester l'efficacité sur des problèmes plus grands ou sur des problèmes où les méthodes de Broyden et d'Anderson ne fonctionnent pas.

Les résultats de ce chapitre ont été publié dans [19].



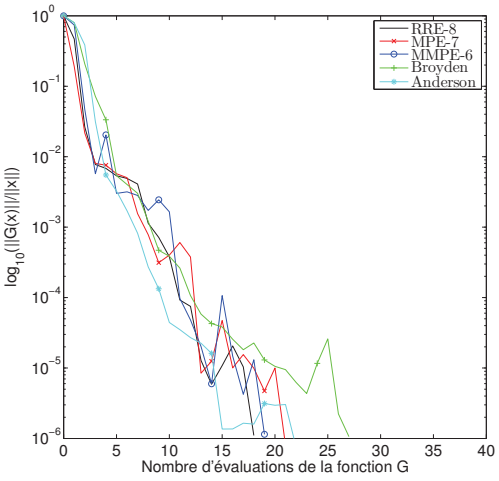
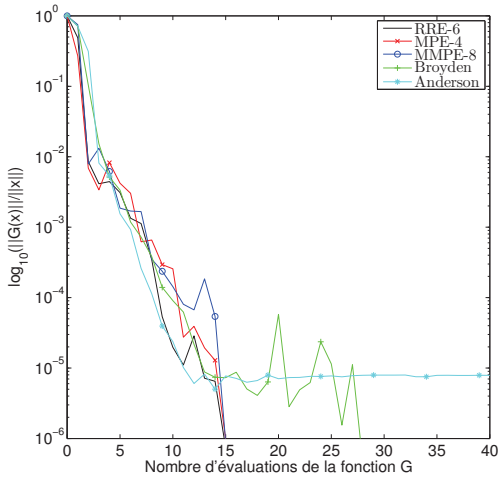


FIGURE 4.10 – Atome C, méthodes cycliques, $\omega = 0.1, 0.2$



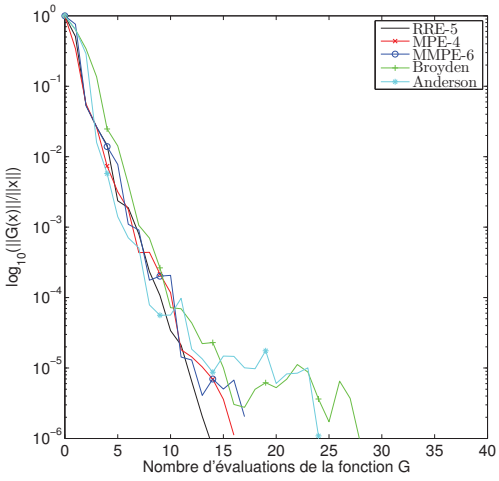
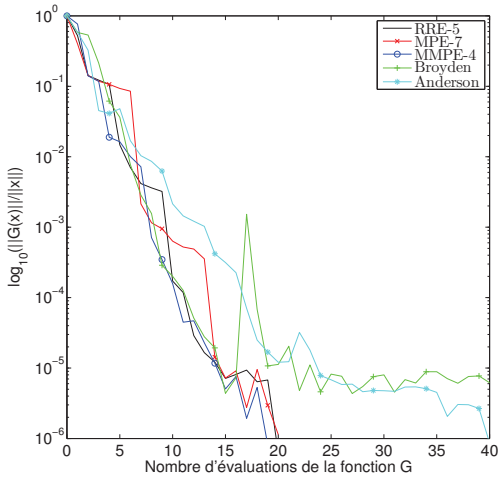


FIGURE 4.11 – Atome C, méthodes cycliques, $\omega = 0.3, 0.4$



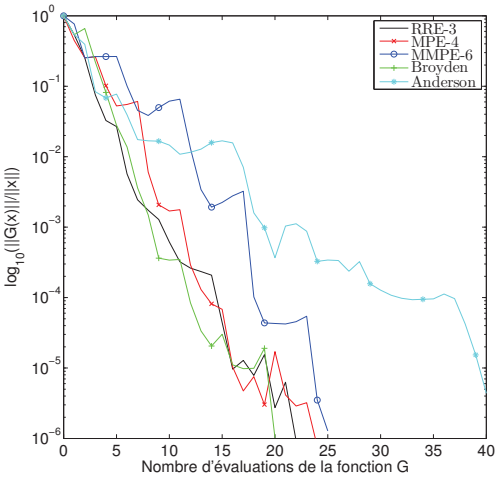
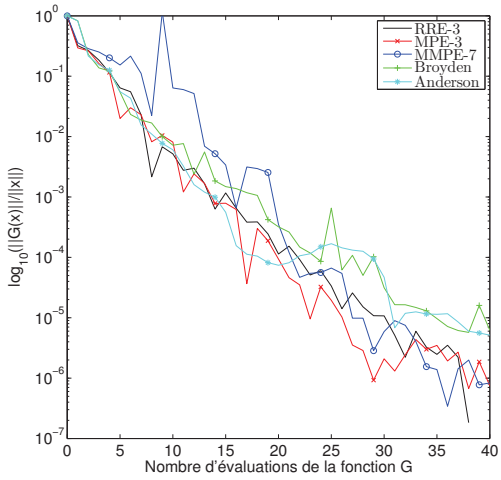


FIGURE 4.12 – Atome C, méthodes cycliques, $\omega = 0.5, 0.6$



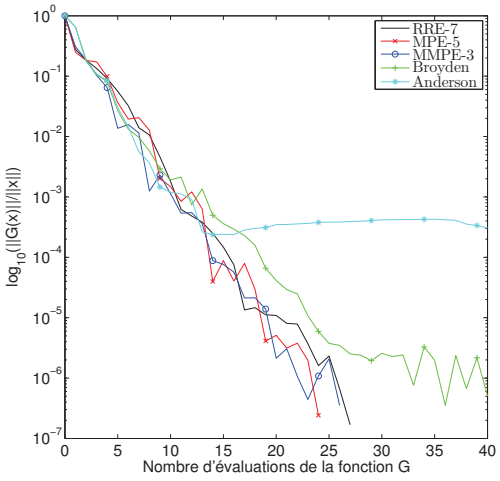
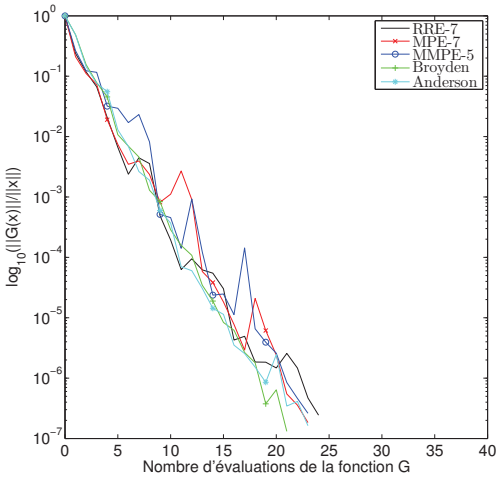


FIGURE 4.13 – Molécule $SiNa$, méthodes cycliques, $\omega = 0.1, 0.2$



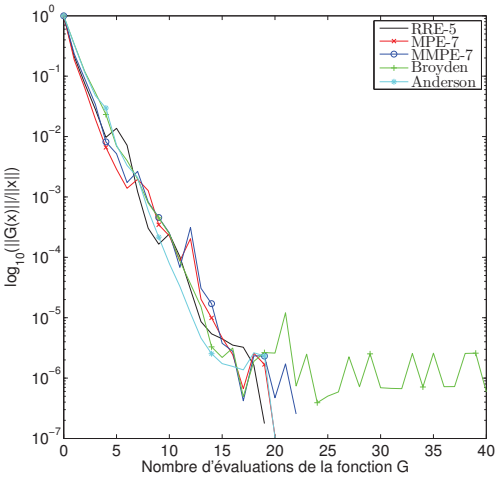
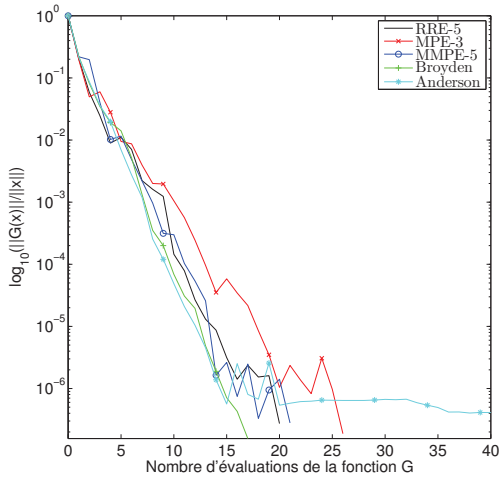


FIGURE 4.14 – Molécule $SiNa$, méthodes cycliques, $\omega = 0.3, 0.4$



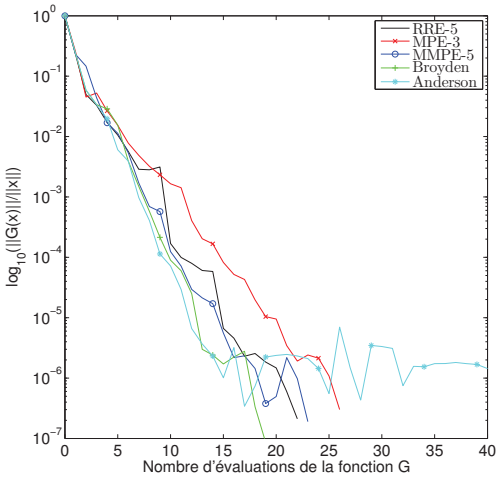
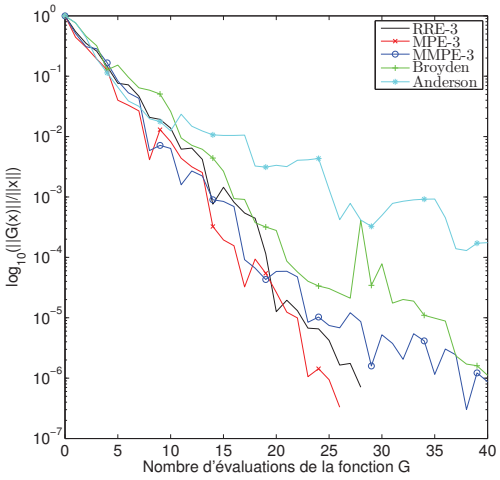


FIGURE 4.15 – Molécule $SiNa$, méthodes cycliques, $\omega = 0.5, 0.6$



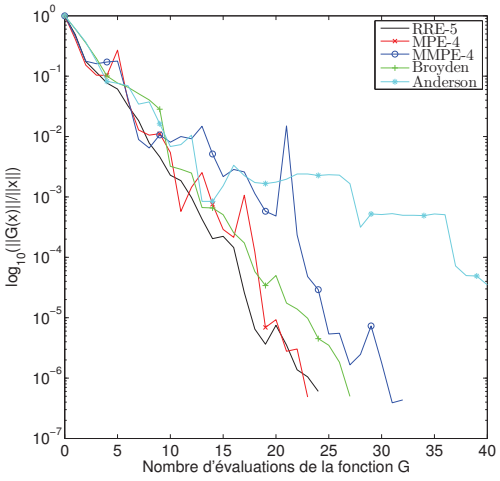
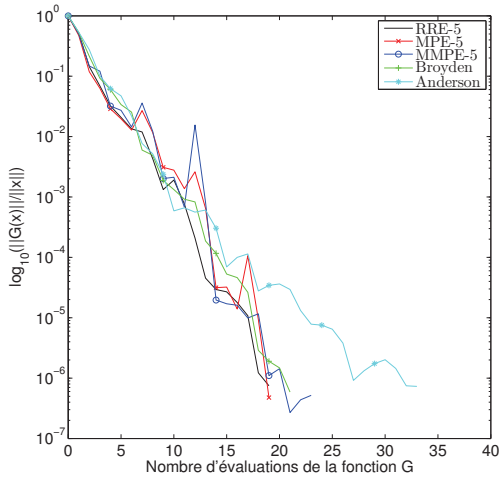


FIGURE 4.16 – Molécule CO, méthodes cycliques, $\omega = 0.1, 0.2$



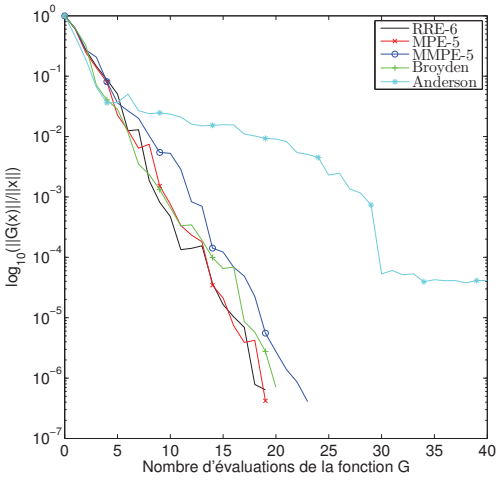
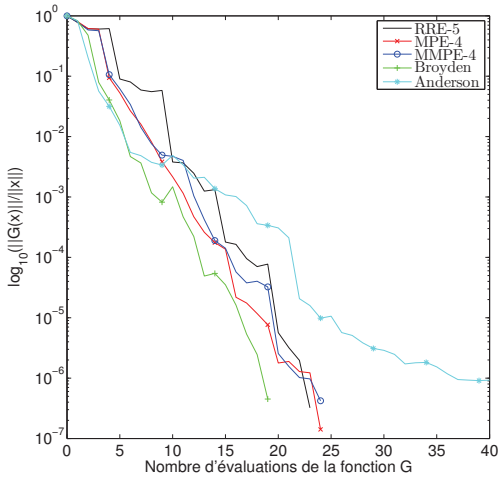


FIGURE 4.17 – Molécule CO, méthodes cycliques, $\omega = 0.3, 0.4$



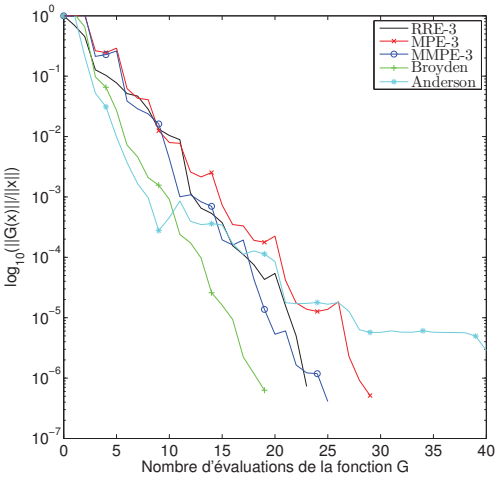


FIGURE 4.18 – Molécule CO, méthodes cycliques, $\omega = 0.5, 0.6$

Chapitre 5

Application 3 : Méthodes Multigrilles

5.1 Introduction

Les méthodes multigrilles (MG) sont au cœur des solveurs d'algèbre linéaire les plus performants. Ce sont des méthodes performantes pour résoudre de grands problèmes provenant de la discrétisation des équations aux dérivées partielles linéaires et non linéaires [9, 11, 47, 67]. Comme leur nom l'indique, la particularité de ces méthodes est de considérer un problème non pas sur un seul maillage, mais sur différents maillages successivement raffinés dans le but de fournir une méthode performante pour approcher le résultat à l'échelle la plus fine. Cette théorie et quelques applications ont été développées dans [9, 31, 32].

Ces méthodes dites géométriques ont cependant quelques inconvénients. Elles ne sont, par exemple, adaptées qu'aux maillages structurés ou uniformes. Pour généraliser aux matrices quelconques, les méthodes multigrilles algébriques (AMG) ont été développées [52, 30].

Ce chapitre se présente comme suit. Nous commençons par rappeler la méthode à 2 grilles (TG) puis la méthode MG. Ensuite, nous donnons quelques détails sur le choix des paramètres utilisés. Enfin, nous définissons la méthode MG algébrique (AMG) avant de donner quelques résultats numériques montrant l'efficacité de nos méthodes d'extrapolation.

5.2 Méthode Multigrille

5.2.1 Principe

Considérons le système d'équations aux dérivées partielles linéaires de la forme suivante :

$$L(u) = f, \tag{5.1}$$

dans un domaine Ω avec des conditions aux bords de Dirichlet. Nous discrétisons Ω en utilisant un maillage régulier de pas h. Supposons que l'équation (5.1) est discrétisée en utilisant une méthode de différence finie. On obtient une équation de la forme suivante :

$$A_h u_h = b_h, \tag{5.2}$$

où A_h est l'opérateur matriciel discret correspondant, b_h est un vecteur donné et u_h est un vecteur inconnu. Soit u_h^k, une approximation de u_h et $e_h^k = u_h - u_h^k$, l'erreur correspondante. On définit le résidu par :

$$r_h^k = b_h - A_h u_h^k. \tag{5.3}$$

On obtient

$$A_h e_h^k = A_h u_h - A_h u_h^k = b_h - A u_h^k = r_h^k. \tag{5.4}$$

Grâce à cette dernière équation, si l'on connait u_h^k, on est capable de calculer r_h^k. Le problème est que la résolution de $A_h e_h^k = r_h^k$ est aussi difficile que le problème original (5.2). Par contre, si l'on connait une approximation v_h^k de e_h^k, alors $v_h^k + u_h^k$ sera probablement une meilleure approximation de u_h. Le principe des méthodes MG est de calculer l'approximation v_h^k sur une grille plus grossière. Nous allons commencer par décrire l'algorithme MG sous la forme la plus simple en ne considérant que 2 grilles.

5.2.2 Méthode à 2 grilles

Algorithme

Dans cette section, nous notons par Ω_h, la grille fine de pas h. De même, Ω_H représentera le maillage grossier de pas H, tel que $H = 2h$. Pour pouvoir résoudre le problème sur Ω_H, nous devons définir un opérateur de restriction pour passer de Ω_h à Ω_H. On note R, cet opérateur :

$$R : \Omega_h \rightarrow \Omega_H. \tag{5.5}$$

Après avoir résolu le problème sur Ω_H, nous devons revenir sur Ω_h. Pour cela, on définit un opérateur de prolongation (ou d'interpolation) qui sera noté par P :

$$P : \Omega_H \to \Omega_h. \tag{5.6}$$

Nous devons maintenant nous intéresser à la résolution du problème sur Ω_H. Nous définissons la matrice A_H par :

$$A_H = RA_hP. \tag{5.7}$$

Nous pouvons maintenant définir un algorithme de la méthode TG (voir Algorithme 21).

Algorithme 21: Algorithme de la méthode TG

Étape 1. $r_h^k = b_h - A_h u_h^k$
Étape 2. $r_H^k = Rr_h^k$
Étape 3. Résoudre exactement $A_H e_H^k = r_H^k$
Étape 4. $v_h^k = Pe_H^k$
Étape 5. $u_h^{k+1} = v_h^k + u_h^k$. Si pas de convergence, retour à l'étape 1.

Lemme 3. *La méthode TG est définie par l'itération suivante :*

$$u_h^{k+1} = (I - PA_H^{-1}RA)u_h^k + PA_H^{-1}Rb_h. \tag{5.8}$$

Preuve 5.1. *À l'aide de cet algorithme, on peut définir une itération de la méthode comme suit :*

$$\begin{aligned} u_h^{k+1} &= Pe_H^k + u_h^k \\ &= PA_H^{-1}r_H^k + u_h^k \\ &= PA_H^{-1}Rr_h^k + u_h^k \\ &= PA_H^{-1}R(b_h - A_h u_h^k) + u_h^k \\ &= (I - PA_H^{-1}RA)u_h^k + PA_H^{-1}Rb_h. \end{aligned}$$

□

Convergence

Une condition nécessaire et suffisante pour avoir convergence de la méthode TG est que $\rho(I - PA_H^{-1}RA) < 1$, ce qui n'est pas souvent le cas comme le montre le lemme suivant donné dans [47].

Lemme 4. *Soit A_h, une matrice symétrique définie positive. Supposons que $A_H = RA_hP$ et $P = R^T$, alors les valeurs propres de $I - PA_H^{-1}RA$ sont* 0 *et* 1.

Preuve 5.2. *$I - PA_H^{-1}RA$ est similaire à $I - A_h^{1/2}PA_H^{-1}RA_h^{1/2}$ qui est symétrique. Posons $z \neq 0$ et λ, un vecteur et une valeur propre de cette matrice :*

$$(I - A_h^{1/2}PA_H^{-1}RA_h^{1/2})z = \lambda z. \tag{5.9}$$

Multiplions par $RA_h^{1/2}$ de chaque côté. On obtient :

$$(RA_h^{1/2} - RA_hPA_H^{-1}RA_h^{1/2})z = \lambda RA_h^{1/2}z, \tag{5.10}$$

or la matrice du membre de gauche vaut 0 *donc $\lambda RA_h^{1/2}z = 0$. On obtient donc $\lambda = 0$ ou $RA_h^{1/2}z = 0$. Nous allons montrer maintenant qu'il existe $z \neq 0$ tel que $RA_h^{1/2}z = 0$. Supposons que Ω_h possède m^2 points avec $m > 1$ et Ω_H possède p^2 points tel que $p = \frac{m-1}{2}$. Alors*

$$dim\ Ker(RA_h^{1/2}) + dim\ Im(RA_h^{1/2}) = m^2, \tag{5.11}$$

donc

$$dim\ Ker(RA_h^{1/2}) \geq m^2 - p^2 > 1. \tag{5.12}$$

Cela implique qu'il existe $z \neq 0$ dans $Ker(RA_h^{1/2})$. Mais, si on suppose $z \neq 0$ et $RA_h^{1/2}z = 0$, nous obtenons $z = \lambda z$ d'où $\lambda = 1$.

□

Ce lemme nous montre que $\rho(I - PA_H^{-1}RA) = 1$. Pour obtenir une convergence, on utilise des méthodes itératives classiques comme les méthodes de Jacobi ou Gauss-Seidel. Nous noterons par S, la matrice représentant la méthode de lissage choisie. L'algorithme 22 donne le processus de résolution de la méthode TG en considérant les étapes de lissage.

Grâce à cet algorithme, on peut donner la formule pour le calcul d'une itération de la méthode TG.

Lemme 5. *La méthode TG utilisant les méthodes de lissage est définie par l'itération suivante :*

$$u_h^{k+1} = S^{\nu_2}(I - PA_H^{-1}RA)S^{\nu_1}u_h^k + S^{\nu_2}PA_H^{-1}Rb_h. \tag{5.13}$$

On peut voir que différentes méthodes peuvent être obtenues. En effet, on peut faire varier tous les paramètres : S, R, P, ν_1, ν_2. Nous allons maintenant définir les paramètres de restriction, de prolongation et de lissage que nous utiliserons dans les exemples numériques.

Algorithme 22: Algorithme de la méthode TG avec lissage

Étape 1. $\bar{u}_h^k = S^{\nu_1} u_h^k$
Étape 2. $\bar{r}_h^k = b_h - A_h \bar{u}_h^k$
Étape 3. $\bar{r}_H^k = R\bar{r}_h^k$
Étape 4. Résoudre exactement $A_H e_H^k = \bar{r}_H^k$
Étape 5. $v_h^k = P\epsilon_H^k$
Étape 6. $\bar{u}_h^{k+1} = v_h^k + \bar{u}_h^k$.
Étape 7. $u_h^{k+1} = S^{\nu_2}\bar{u}_h^{k+1}$. Si pas de convergence, retour à l'étape 1.

Opérateur de restriction

Nous devons choisir un opérateur de restriction pour aller de Ω_h dans Ω_H. Plusieurs choix sont possibles. Nous allons donner ici deux méthodes : la première (la plus simple) consiste à choisir uniquement les valeurs des points correspondant sur le maillage Ω_h. Posons $r_H^k = R_I r_h^k$ défini par :

$$(r_H^k)_{i,j} = (r_h^k)_{2i,2j} \text{ pour } 1 \le i \le p, 1 \le j \le p. \tag{5.14}$$

Le deuxième choix que nous considérons est celui utilisant une combinaison des points voisins de la grille Ω_h. Cette méthode est définie comme suit : Posons $r_H^k = R_W r_h^k$ où

$$\begin{array}{rcll} (r_H^k)_{i,j} & = & \frac{1}{4}(r_h^k)_{2i,2j} + & \\ & & \frac{1}{8}((r_h^k)_{2i-1,2j} + (r_h^k)_{2i+1,2j} + (r_h^k)_{2i,2j-1} + (r_h^k)_{2i,2j-1}) + & \forall 1 \le i \le p, 1 \le j \le p. \\ & & \frac{1}{16}((r_h^k)_{2i-1,2j-1} + (r_h^k)_{2i-1,2j+1} + (r_h^k)_{2i+1,2j-1} + (r_h^k)_{2i+1,2j+1}) & \end{array} \tag{5.15}$$

Nous utiliserons ce deuxième choix pour les résultats numériques.

Opérateur de prolongation

Comme dans la section précédente, nous définissons maintenant un opérateur de prolongation pour aller de Ω_H dans Ω_h. Nous choisissons un opérateur linéaire P_L défini par :

$$\begin{array}{rcll} (r_h^k)_{2i,2j} & = & (r_H^k)_{2i,2j}, & \\ (r_h^k)_{2i+1,2j} & = & \frac{1}{2}\left((r_H^k)_{2i,2j} + (r_H^k)_{2i+2,2j}\right), & \\ (r_h^k)_{2i,2j+1} & = & \frac{1}{2}\left((r_H^k)_{2i,2j} + (r_H^k)_{2i,2j+2}\right), & \forall 1 \le i \le p, 1 \le j \le p. \\ (r_h^k)_{2i+1,2j+1} & = & \frac{1}{4}\left((r_H^k)_{2i,2j} + (r_H^k)_{2i+2,2j} + (r_H^k)_{2i,2j+2} + (r_H^k)_{2i+2,2j+2}\right), & \end{array} \tag{5.16}$$

Méthode de Jacobi

Décomposons la matrice A de la manière suivante :

$$A = D - L - U, \tag{5.17}$$

où D est la matrice contenant la diagonale de A ; L et U les parties strictement inférieure et supérieure de $-A$. Alors l'équation $Au = f$ devient :

$$\begin{aligned} (D-L-U)u &= f \\ Du &= (L+U)u + f \\ u &= D^{-1}(L+U)u + D^{-1}f. \end{aligned}$$

Posons $S_J = D^{-1}(L+U)$. Alors l'itération de la méthode de Jacobi se calcule comme suit :

$$v^{new} = S_J v^{old} + D^{-1}f. \tag{5.18}$$

Méthode de Gauss-Seidel

Reprenons la décomposition de la matrice A donnée dans la méthode de Jacobi.

$$\begin{aligned} (D-L-U)u &= f \\ (D-L)u &= Uu + f \\ u &= (D-L)^{-1}Uu + (D-L)^{-1}f. \end{aligned}$$

Posons $S_{GS} = (D-L)^{-1}U$. Alors l'itération de la méthode de Gauss-Seidel est définie par :

$$v^{new} = S_{GS} v^{old} + (D-L)^{-1}f. \tag{5.19}$$

La méthode de Gauss-Seidel converge plus rapidement que la méthode de Jacobi, mais demande plus de calculs par itération en général. De plus, la méthode de Jacobi est très bien parallélisable.

Intéressons nous maintenant à la généralisation de la méthode TG : la méthode MG.

5.2.3 Méthode Multigrille

Introduction

Dans la section précédente, nous avons résolu le système (5.2) en utilisant uniquement 2 grilles. Dans l'étape 4 de l'algorithme 22, on a fait l'hypothèse que l'on peut résoudre exactement $A_H e_H^k = \bar{r}_H^k$, ce qui n'est pas réaliste dans la pratique. En effet,

si on veut résoudre, par exemple, un système 2D avec un maillage standard, Ω_H ne possède qu'un quart du nombre de points de Ω_h. Du coup, le problème grossier reste encore très large.

Une idée naturelle, dans ce cas, est de résoudre approximativement $A_H e_H^k = \bar{r}_H^k$ en utilisant le même algorithme TG défini sur une grille plus grossière de pas $4h$; et ceci jusqu'à une grille contenant 1 ou 4 points seulement. C'est le principe de la méthode MG. La dernière remarque est très importante. Par exemple, pour un problème 2D, nous avons besoin de choisir $m = 2^p - 1$ points dans chaque direction.

Algorithme

Pour définir un algorithme de la méthode MG, introduisons un indice l qui correspond à la grille utilisée. On définit ainsi une suite Ω_l de grilles de pas h_l, $l = 0$ correspond à la grille la plus grossière et $l = L$ à la grille la plus fine. Définissons maintenant quelques notations :

- A_l l'approximation de A sur la grille Ω_l,
- R_l l'opérateur de restriction : $\Omega_l \to \Omega_{l-1}$,
- P_l l'opérateur d'interpolation : $\Omega_{l-1} \to \Omega_l$,
- S_l la matrice de la méthode de lissage sur Ω_l.

Le processus de la méthode MG pour $l+1$ grilles est donné dans l'algorithme 23.

Algorithme 23: Algorithme de la méthode $l+1$-grilles

Si $l = 1$, appliquer l'algorithme 22.
si $l > 1$ **alors**
  Étape 1. $\bar{u}_l^k = S^{\nu_1} u_l$,
  Étape 2. $\bar{r}_l^k = b_l - A_l \bar{u}_l^k$,
  Étape 3. $\bar{r}_{l-1}^k = R_l \bar{r}_l^k$,
  Étape 4. Calculer $\bar{v}_{l-1}^k$, une approximation de la solution de $A_{l-1} v_{l-1}^k = \bar{r}_{l-1}^k$ on Ω_{l-1} en faisant γ itérations de l'algorithme l-grilles $(\Omega_{l-1}, \cdots, \Omega_0)$ en partant de 0.
  Étape 5. $\bar{v}_l^k = P_{l-1} \bar{v}_{l-1}^k$
  Étape 6. $\bar{u_h}^{k+1} = v_h^k + \bar{u_h}^k$.
  Étape 7. $u_l^{k+1} = S^{\nu_2} \bar{u}_l^k$. Si pas de convergence, aller à l'étape 1
fin

On remarque que cet algorithme contient un nouveau paramètre γ qui représente le nombre de fois que la méthode MG est appliquée à la grille grossière. Si on choisit $\gamma = 1$, on obtient la méthode V-cycle. Si on choisit $\gamma = 2$, on obtient la méthode W-cycle.

L'expression de la méthode MG est donnée par le lemme suivant.

Lemme 6. *La matrice itérative du processus MG est donnée sous une forme récursive de la manière suivante : pour $l = 1$, poser $M_1 = 0$, pour $l = 2, \cdots, L$:*

$$M_l = S_l^{\nu_2} \left(I - P(I - M_{l-1}^{\gamma}) A_{l-1}^{-1} R_l A_l \right) S_l^{\nu_1} \tag{5.20}$$

où M_l est la matrice de la méthode MG au niveau l.

5.3 Méthode Multigrille algébrique

5.3.1 Définition

La méthode MG géométrique, que nous venons d'expliquer dans la section précédente, ne peut pas être utilisée pour tous les problèmes aux dérivées partielles. En effet, lorsque le problème est anisotrope ou lorsque le problème possède de fortes variations des coefficients, la méthode MG standard n'est pas assez robuste. Dans ce cas, on peut utiliser une méthode multigrilles algébrique (AMG). Le principe de cette méthode est que l'on utilise uniquement la matrice A sans considérer le problème d'équations aux dérivées partielles correspondant. Ce procédé a été introduit par Ruge et Stuben dans [52].

Considérons le système d'équations discrétisé de la forme

$$Au = f. \tag{5.21}$$

Une méthode AMG , comme les méthodes MG standards, utilise deux étapes : une étape de relaxation et une étape raffinement celle-ci comprenant l'étape de restriction et l'étape de prolongation. Généralement, la méthode de lissage utilisée pour la méthode AMG est la méthode de Gauss-Seidel définie dans la section 5.2.2. Nous allons appliquer l'algorithme de prolongation défini dans [30].

5.3.2 Méthode de raffinement

À chaque niveau, nous avons besoin de partitionner l'ensemble des inconnus dans une grille de points grossière et dans une grille de points fine. Posons N, le nombre d'inconnus. L'algorithme que nous allons définir sera utilisé pour des M-matrices symétriques. L'idée de cet algorithme est d'identifier les points fortement associés dans le sens suivant. Soit I, un sous ensemble de N et

$$d(i, I) = \frac{\sum_{j \in I} -a_{i,j}}{\max_{j \neq i}(-a_{i,j})}. \tag{5.22}$$

Soit α, un paramètre donné et

$$S_i = \{j \in N | d(i, \{j\}) \geq \alpha\}, \;\; U_i = \{j \in N | i \in S_j\}. \tag{5.23}$$

On notera C, l'ensemble des points de la grille grossière et F, l'ensemble des points contenus dans la grille fine. La sélection des points de C et F se fait en deux étapes grâce aux algorithmes 24 et 25.

Algorithme 24: Étape I

Poser $C = \emptyset$, $F = \emptyset$;
tant que $C \cup F \neq N$ **faire**
  Choisir $i \in N \setminus (C \cup F)$ avec un nombre maximal d'éléments : $|U_i| + |U_i \cap F|$;
  si $|U_i| + |U_i \cap F|$=*0* **alors**
    Poser $F = N \setminus C$;
  sinon
    Poser $C = C \cup \{i\}$ et $F = F \cup (U_i \setminus C)$;
  fin
fin

Algorithme 25: Étape II

Poser $T = \emptyset$;
tant que $T \subset F$ **faire**
  Choisir $i \in F \setminus T$ et poser $T = T \cup \{i\}$;
  Poser $\tilde{C} = \emptyset$, $C_i = S_i \cap C$ et $F_i = S_i \cap F$;
  tant que $F_i \neq \emptyset$ **faire**
    Choisir $j \in F_i$ et poser $F_i = F_i \setminus \{j\}$;
    si $d(j, C_i)/d(i, \{j\}) \leq \beta$ **alors**
      si $|\tilde{C}| = 0$ **alors**
        poser $\tilde{C} = \{j\}$ et $C_i = C_i \cup \{j\}$;
      sinon
        poser $C = C \cup \{i\}$, $F = F \setminus \{i\}$ et retour à la ligne 2.
      fin
    fin
  fin
  $C = C \cup \tilde{C}$, $F = F \setminus \tilde{C}$.
fin

Au niveau l, après avoir divisé les points de N_l dans C_l et F_l, nous définissons l'opérateur d'interpolation P_l comme suit : pour un vecteur x donné

$$\begin{cases} (P_l x)_i & = x_i \text{ si } i \in C_l \\ (P_l x)_i & = -\dfrac{\sum_{j \in C_l} (a^+_{i,j} c_{i,j}) x_j}{a^l_{i,i} + c_{i,i}}, \text{ si } i \in F_l, \end{cases} \tag{5.24}$$

où

$$c_{i,j} = \sum_{k \notin C_l, k \neq i} \frac{a^l_{i,k} a^l_{k,j}}{a^l_{k,i} + \sum_{p \in C_l} a^l_{k,p}}. \tag{5.25}$$

La matrice à l'étape $l-1$ est donnée par l'identité de Galerkin

$$A_{l-1} = (P_l)^T A_l P_l. \tag{5.26}$$

Ensuite, on utilise les algorithmes 22 et 23 avec ce choix d'opérateur. Il reste à choisir les paramètres α et β. Dans [30], les auteurs recommandent de poser $\alpha = 0.25$ et $\beta = 0.35$.

5.4 Résultats numériques

Tous les tests de cette section ont été réalisés en utilisant Matlab 7.1. Le critère d'arrêt utilisé est la norme du résidu. On arrêtera les itérations lorsque que cette norme sera inférieure à 10^{-7}. Nous nous intéressons à la résolution de l'équation suivante :

$$\begin{cases} -\nabla^2 u = f, \text{ sur } \Omega \\ \quad u = 0, \text{ sur } \partial\Omega, \end{cases} \tag{5.27}$$

où Ω désigne le carré unité de $\mathbb{R}^2$. On suppose des conditions aux bords de Dirichlet. Nous testons le pouvoir d'accélération de la convergence grâce aux méthodes d'extrapolation.

5.4.1 Méthode Multigrille géométrique

Nous commençons par tester les méthodes d'extrapolation sur la méthode MG standard. Nous utilisons le programme MGLab1.0 [8]. MGLab est un ensemble de fonctions qui définit un environnement interactif afin d'expérimenter de nouveaux procédés avec les algorithmes MG. Le programme résout des équations aux dérivées partielles elliptiques en deux dimensions avec une discrétisation par différences finies. Nous utilisons l'opérateur de restriction R_W défini dans l'équation (5.15) et l'opérateur de prolongation P_L défini dans l'équation 5.16. Tous les tests seront réalisés en posant $N = 3969$. Nous présentons les résultats en utilisant la suite générée par la méthode MG avec la méthode de Jacobi. Nous regardons l'efficacité de la convergence sur la méthode de Jacobi car celle-ci est plus parallélisable que la méthode Gauss-Seidel.

Les figures 5.1 à 5.4 montrent le comportement de la norme du résidu et la norme de l'erreur en utilisant l'échelle logarithmique, pour différentes valeurs de l et différents choix pour ν_1 et ν_2. Pour chaque figure, les deux premières courbes représentent

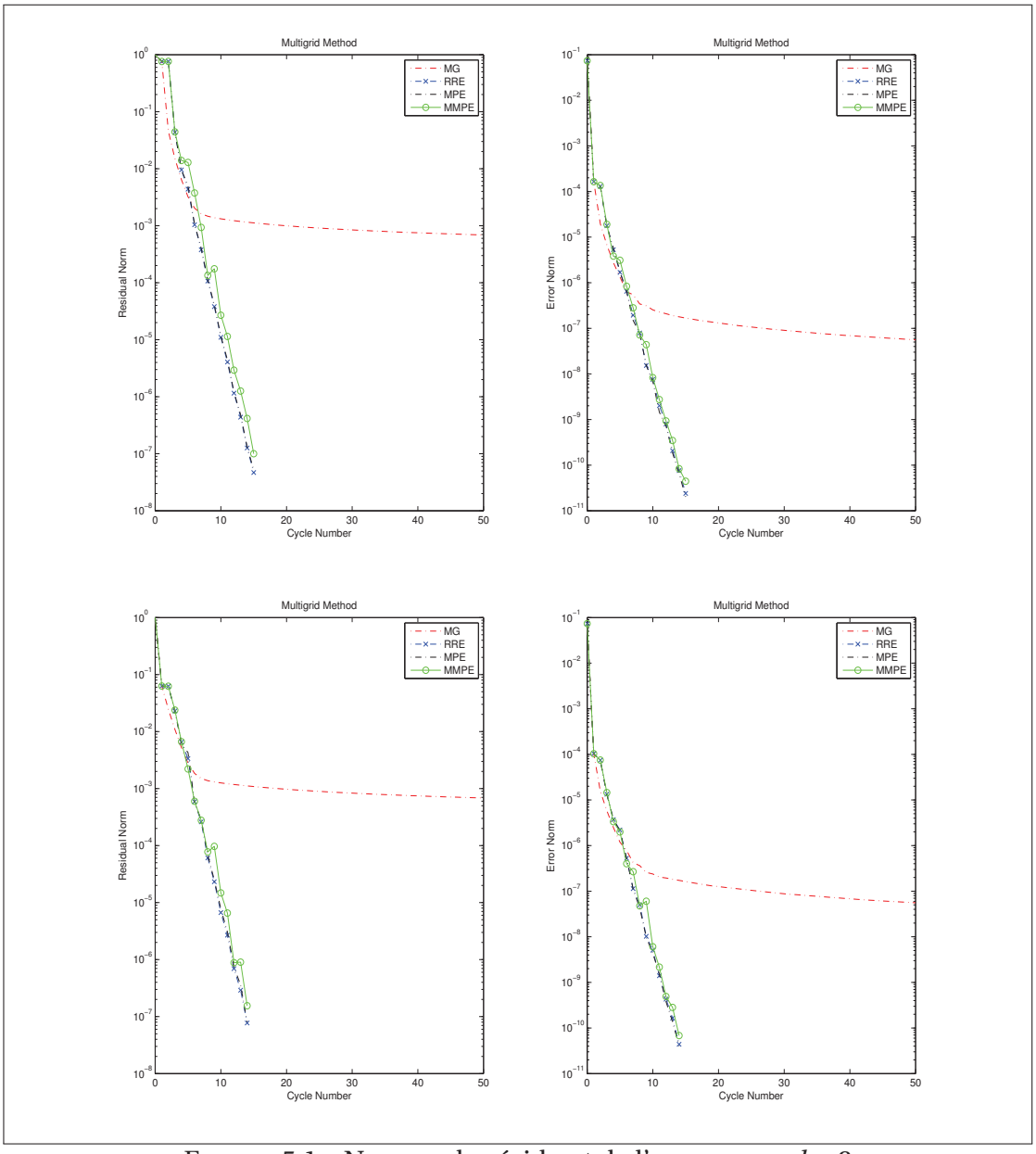


FIGURE 5.1 – Normes du résidu et de l'erreur pour $l = 2$

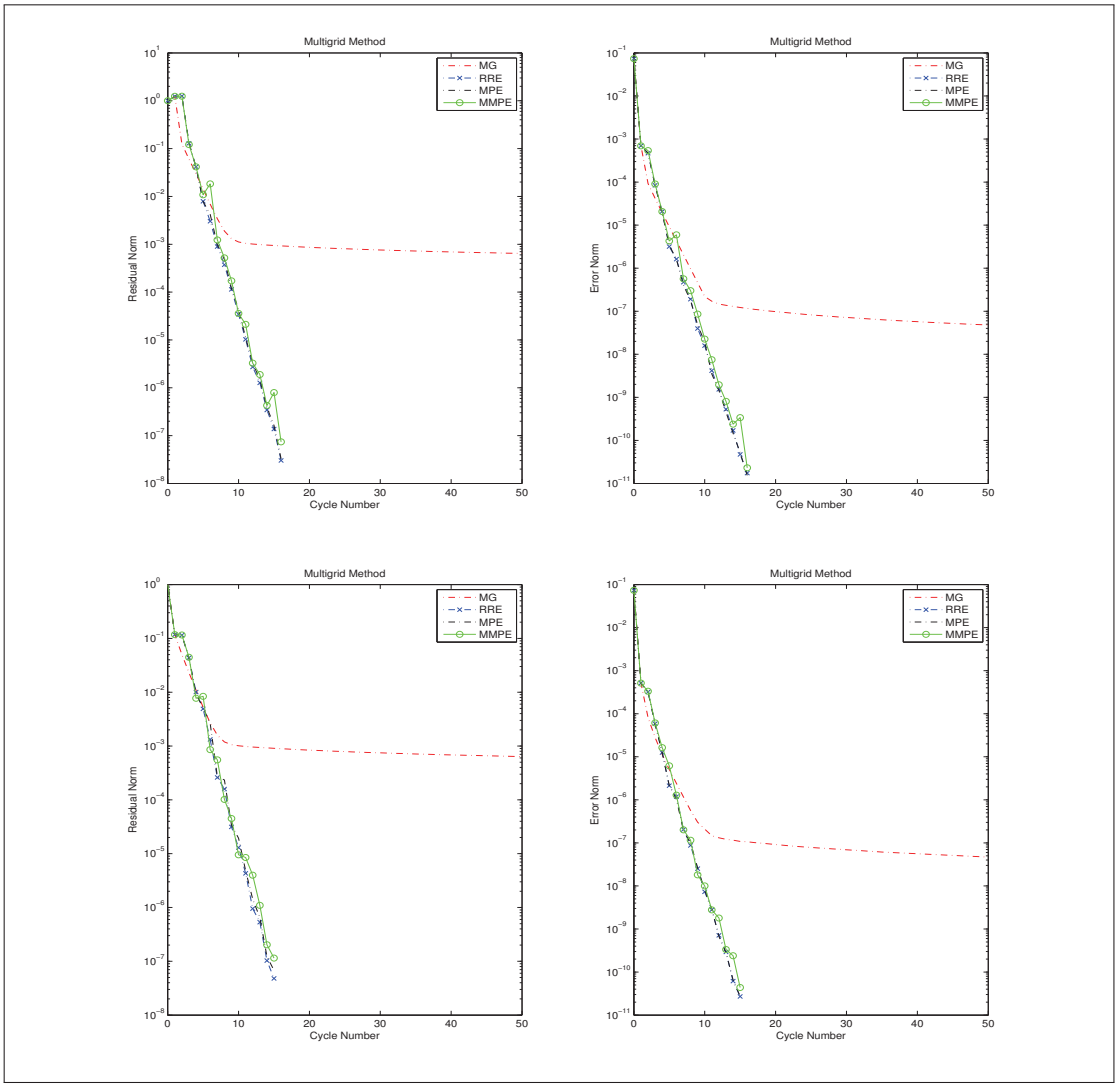


FIGURE 5.2 – Normes du résidu et de l'erreur pour $l = 3$

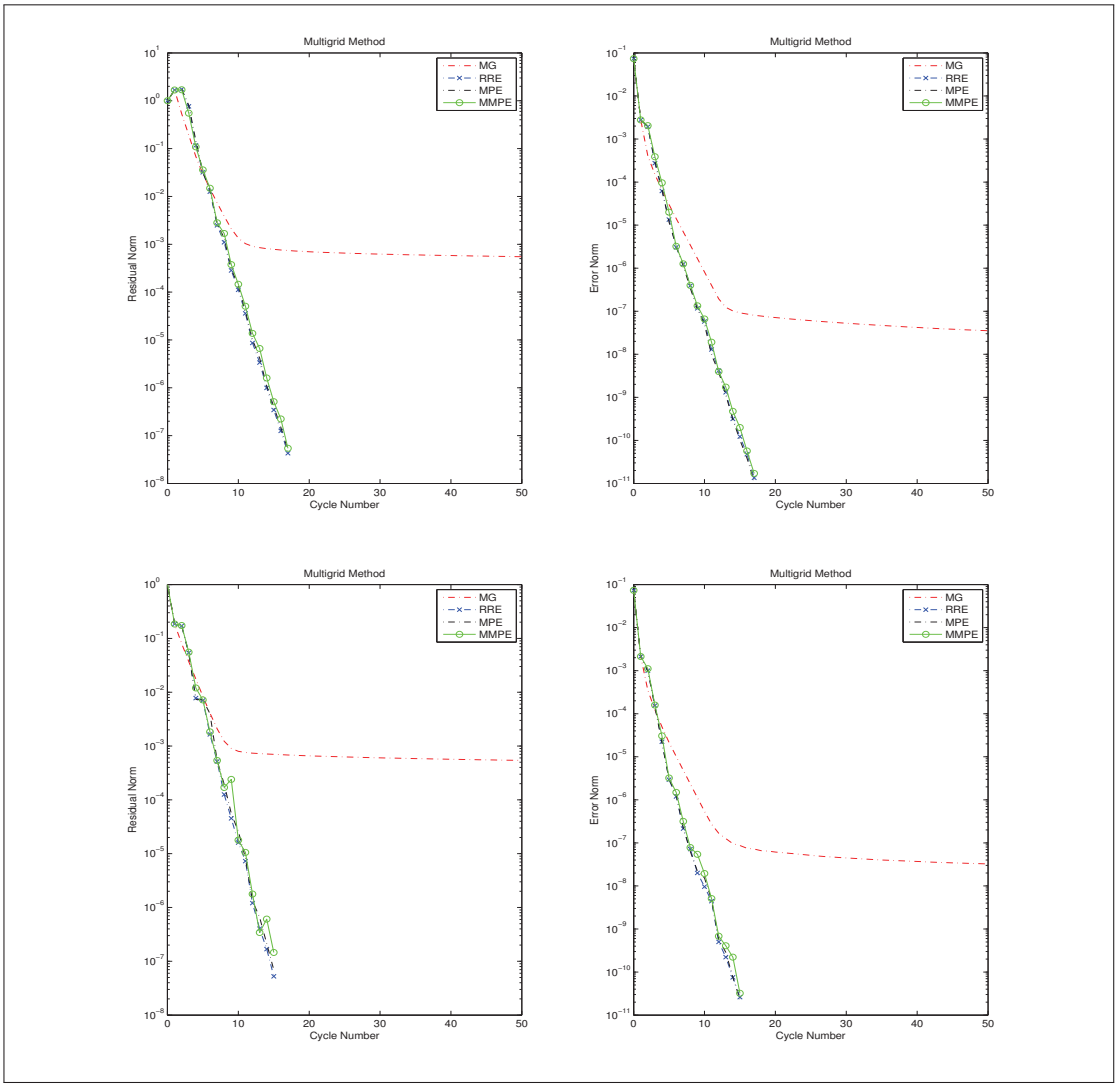


FIGURE 5.3 – Normes du résidu et de l'erreur pour $l = 4$

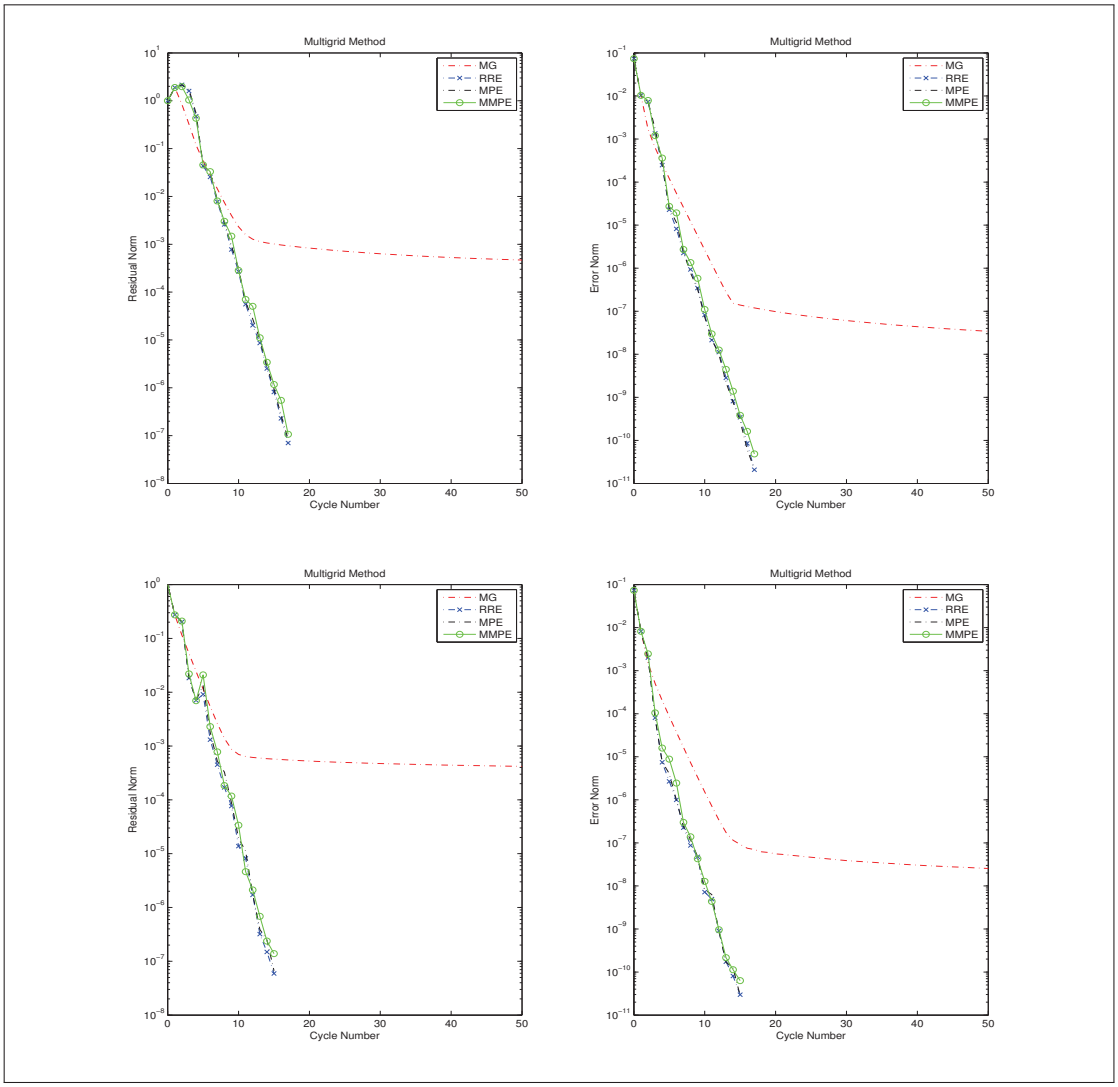


FIGURE 5.4 – Normes du résidu et de l'erreur pour $l = 5$

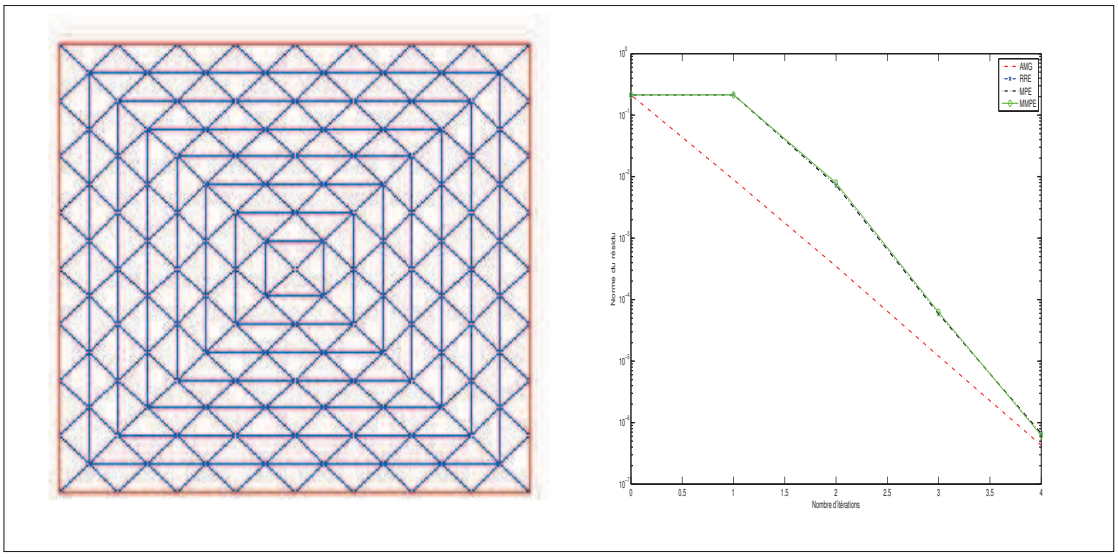


FIGURE 5.5 – Cas d'un maillage uniforme ($N = 145$)

les résultats en posant $\nu_1 = 1$ et $\nu_2 = 0$, les deux dernières représentent les résultats en posant $\nu_1 = 0$ et $\nu_2 = 1$.

Les résultats montrent que la méthode MG avec Jacobi possède une convergence très lente après quelques itérations. Cette stagnation arrive de plus en plus rapidement au fur et à mesure que le terme l diminue. Par contre, les méthodes d'extrapolation ont une convergence très rapide. Cette convergence n'est pratiquement pas modifiée par les différentes valeurs de l. Le fait d'utiliser la méthode de lissage au début ou à la fin du calcul de l'itération n'affecte pas les résultats.

5.4.2 Méthode Multigrille algébrique

On a énoncé dans la section 5.3, que la méthode AMG était intéressante dans le cas où le maillage était non uniforme. C'est pourquoi dans cette section, nous allons nous intéresser à la convergence des méthodes AMG dans le cas de maillages différents. On rappelle que la méthode de lissage que l'on utilise pour la méthode AMG est Gauss-Seidel symétrique.

La figure 5.5 montre le comportement des méthodes pour un maillage uniforme. On observe, dans ce cas, la convergence rapide de la méthode AMG. Les méthodes d'extrapolation sont moins efficaces.

Les figures 5.6, 5.7 et 5.8 montrent le comportement de la norme du résidu pour le cas d'un maillage où le pas se rétrécie au centre du carré pour différentes tailles ($N = 145$, $N = 545$ et $N = 2113$). Les courbes montrent que le nombre d'itérations nécessaires à la convergence de la méthode AMG augmente en fonction de la taille du problème. On observe aussi la grande efficacité des méthodes d'extrapolation. Le

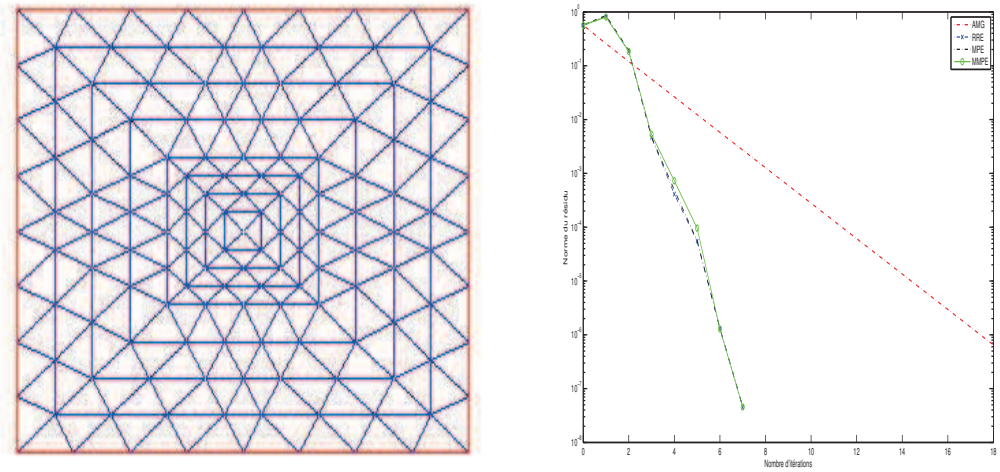


FIGURE 5.6 – Cas $N = 145$

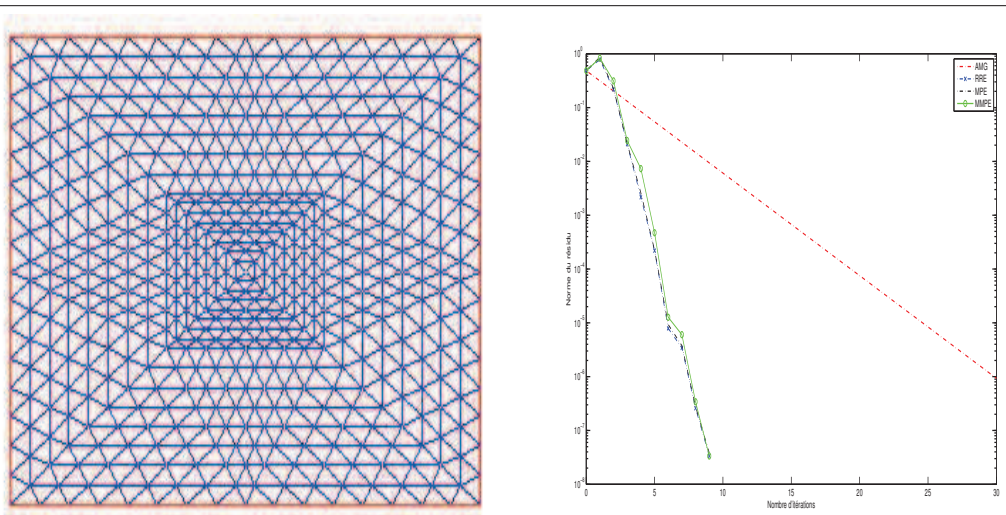


FIGURE 5.7 – Cas $N = 545$

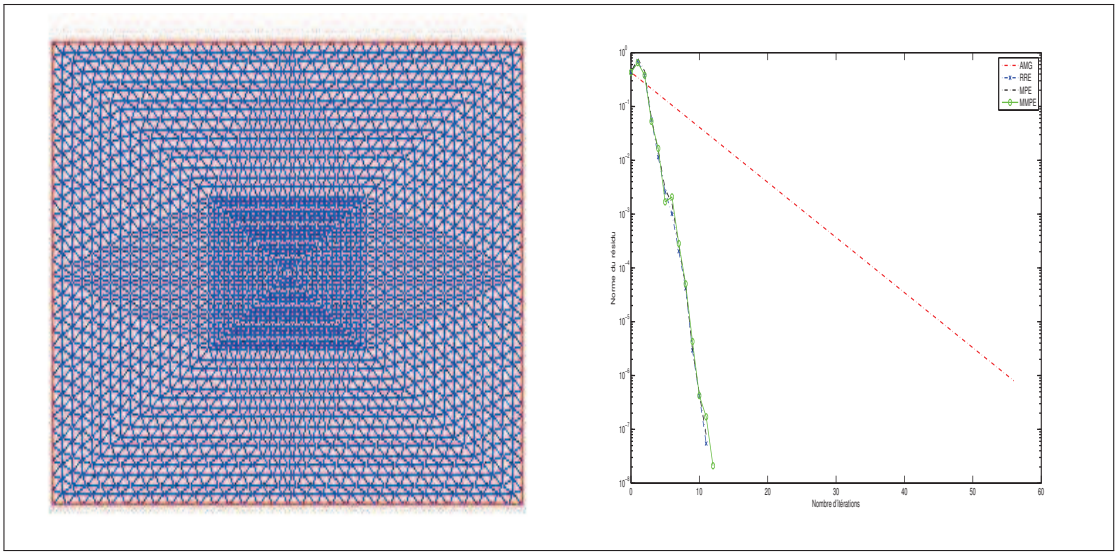


FIGURE 5.8 – Cas $N = 2113$

nombre d'itérations augmente un peu en fonction de la taille du problème.

Les figures 5.9 et 5.10 montrent les résultats pour des maillages non uniformes pour deux tailles de problème différentes. On observe encore l'accélération dûe à l'utilisation des méthodes d'extrapolation. On remarque aussi que l'efficacité de la méthode AMG dépend du maillage considéré.

Nous terminerons ces tests par un problème provenant de l'ingénierie. La matrice considérée est une matrice creuse de taille $N = 1224$ issue de la collection Matrix Market. La figure 5.11 montre la convergence de la méthode AMG, RRE et RRE cyclique. Les résultats montrent bien que pour ce problème difficile à résoudre, la méthode AMG converge très lentement tandis que l'application des méthodes RRE accélère de façon très significative la convergence. L'utilisation de la méthode cyclique permet d'améliorer cette efficacité.

5.5 Conclusion

Dans ce chapitre, nous nous sommes intéressés aux méthodes multigrilles. Nous avons défini les différentes méthodes (géométrique et algébrique) et les différents algorithmes. Ces méthodes sont très appréciées dans la résolution de problèmes provenant de la discrétisation des équations aux dérivées partielles. Nous avons voulu tester le comportement de la convergence de ces méthodes lorsque qu'elles étaient associées aux méthodes d'extrapolation. Nous avons donc testé toutes ces méthodes dans la résolution de l'équation de Poisson pour différentes tailles de problème, différents nombre de niveaux de raffinement, différents maillages. Tous ces tests ont montré la vraie efficacité de nos méthodes. D'autres tests, sur des exemples plus

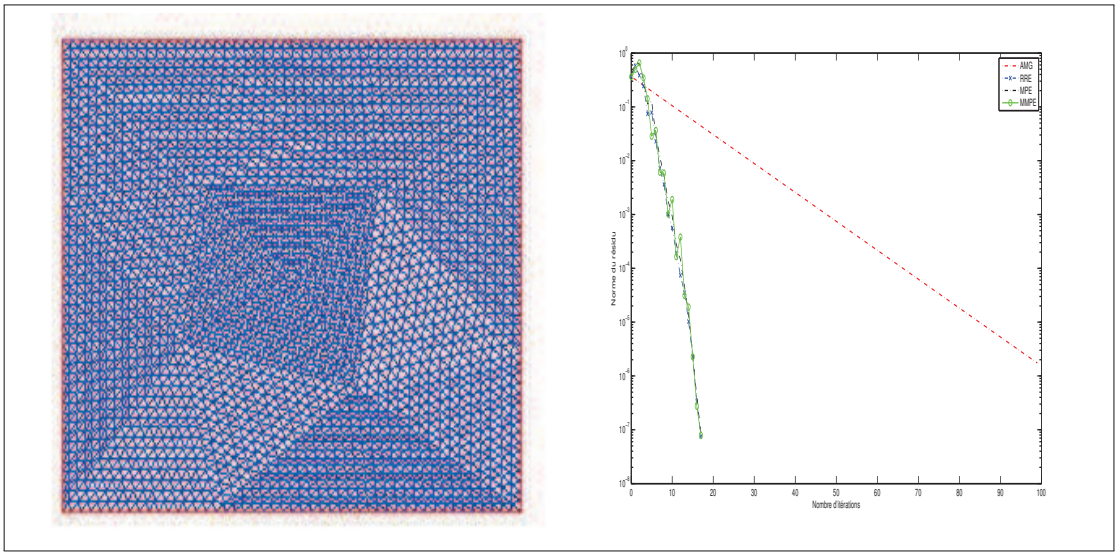


FIGURE 5.9 – Cas d'un maillage non uniforme ($N = 2657$)

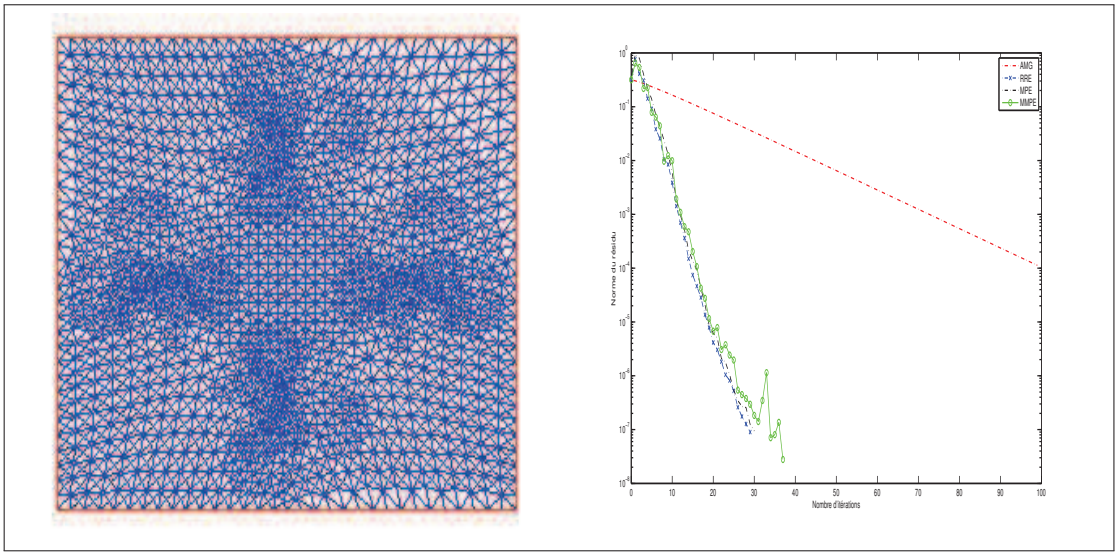


FIGURE 5.10 – Cas d'un maillage non uniforme ($N = 2445$)

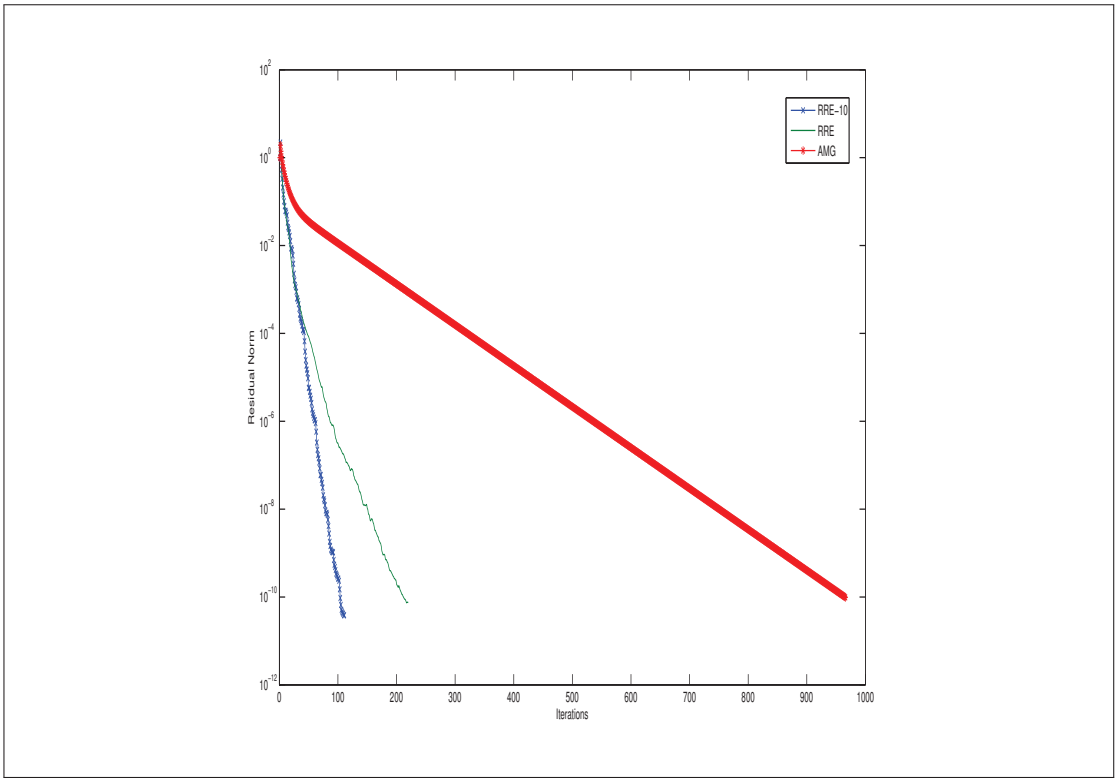


FIGURE 5.11 – Cas d'une matrice issue de Matrix Market ($N = 1224$)

grands, sur d'autres équations, pourraient valider encore plus cette efficacité.

Chapitre 6

Méthode CMRH : Préconditionnement et Implémentation Parallèle

6.1 Introduction

On s'intéresse, dans ce chapitre, à la résolution du système linéaire suivant :

$$Ax = b, \tag{6.1}$$

où A est une matrice carrée d'ordre N supposée inversible, x et b sont deux vecteurs de taille N. La résolution de ce système apparaît dans de nombreux domaines d'applications. La plupart des méthodes utilisées pour résoudre de manière efficace des systèmes linéaires de la forme (6.1) sont des méthodes de sous-espaces de Krylov. Ces méthodes construisent, à l'aide d'une approximation de départ, une suite de solutions approchées de la solution exacte du système considéré.

Les méthodes de sous-espaces de Krylov les plus connues sont la méthode du gradient conjugué [35], la méthode BCG (Bi-Conjugate Gradient) [27, 45], la méthode FOM (Full Orthogonalized Method) [54], la méthode GMRES (Generalized Minimal Residual Method) [56] et la méthode CMRH (Changing Minimal Residual Method based on the Hessenberg reduction process) [59].

Le principe de ces méthodes est la création d'une base de $K_k(A, v)$ et d'une matrice de Hessenberg supérieure. Par exemple, la méthode GMRES utilise le processus d'Arnoldi décrit dans l'algorithme 5 pour trouver cette base. La méthode GMRES minimise de façon optimale le résidu sur $K_{k+1}(A, r_0)$. La méthode CMRH utilise le processus de Hessenberg décrit dans l'algorithme 7. L'avantage de la méthode CMRH est qu'elle réduit le nombre de calculs et la place mémoire tout en ayant une convergence comparable à la méthode GMRES.

Dans ce chapitre, on commencera par rappeler les différentes étapes qui ont permis d'aboutir à l'algorithme 15 définie dans le chapitre 2.

Les avancées réalisées en informatique nous permettent de résoudre des systèmes de plus en plus grands, ce qui entraîne une augmentation du temps de calcul. Pour réduire ce temps, deux possibilités s'offrent à nous. La première consiste à chercher une matrice de préconditionnement qui permettra de réduire de façon conséquente le nombre d'itérations et ainsi de réduire le temps. Nous développons, dans ce chapitre, la méthode CMRH préconditionnée. Nous donnons son algorithme et montrons son efficacité sur un exemple issu d'un problème d'électrostatique.

La deuxième possibilité est le développement de méthodes parallèles. Celui-ci tient une grande place dans la recherche actuelle. Dans cette optique de réduction du temps, on continuera ce chapitre en développant la méthode CMRH en parallèle. On donnera les différentes étapes de la parallélisation qui permettront d'aboutir à une méthode CMRH en parallèle efficace. On montrera que notre méthode permet de bien répartir le nombre de calculs par processeur tout en évitant au mieux les échanges. Nous montrerons aussi qu'avec la même implémentation, la méthode CMRH est plus rapide que la méthode GMRES. Cette étude se fera sur quelques exemples numériques.

6.2 Méthode CMRH séquentielle

Commençons par donner l'implémentation séquentielle de la méthode CMRH. Comme on l'a énoncé dans le chapitre 2, la méthode CMRH utilise l'algorithme de Hessenberg, décrit dans l'algorithme 7, pour construire une base du sous espace de Krylov. Soit v un vecteur de $\mathbb{R}^N$ et A une matrice donnée. L'algorithme de Hessenberg construit une matrice unitaire trapézoïdale, que l'on notera $L_k = \{l^1, \cdots, l^k\}$, pour laquelle chaque colonne forme une base du sous espace $K_k(A, v)$ et $l^i,\ i = 1, \cdots, k$ est définie par

$$(l^i)_i = 1 \ \text{ et } \ (l^i)_j = 0,\ j = 1, \cdots, i-1.$$

Les colonnes sont construites en utilisant les formules suivantes :

$$\beta = ((e)_1, v)\ ; \quad l^1 = \frac{v}{\beta}$$

$$H_{k+1,k} l^{k+1} = A l^k - \sum_{j=1}^{k} H_{j,k} l^j \text{ pour } \ k = 1, \cdots, m$$

où chaque coefficient $H_{j,k}$ est déterminé de la manière suivante :

$$l^{k+1} \perp (e)_1, \cdots, (e)_k \text{ et } (l^{k+1})_{k+1} = 1. \tag{6.2}$$

Comme pour une décomposition LU d'une matrice, l'algorithme 7 devient très instable lorsque $H_{k+1,k} = 0$ ou est proche de 0. Ces situations entraînent pour l'une

un breakdown, pour l'autre une grande perte de précision. Pour éviter cela, il a été développé la méthode avec pivotage, comme pour la méthode d'élimination de Gauss. Ce changement consiste à remplacer la condition d'orthogonalité (6.2) par :

$$l^{k+1} \perp (e)_{(p)_1}, \cdots, (e)_{(p)_k} \text{ et } (l^{k+1})_{(p)_{k+1}} = 1, \tag{6.3}$$

où le vecteur p est un vecteur de permutation. L'algorithme 26 inclut cette modification. Dans cette algorithme, on a rajouté la condition $u \neq 0$. Celle-ci correspond

Algorithme 26: Algorithme du processus de Hessenberg avec pivotage

Initialisation $p = [1, 2, \cdots, N]^T$;
On détermine i_0 tel que $\left|(v)_{i_0}\right| = \|v\|_\infty$;
$\beta = (v)_{i_0}$, $l^1 = v/\beta$, $(p)_1 \leftrightarrow (p)_{i_0}$;
pour $k = 1, \cdots, m$ **faire**
  $u = Al^k$;
  pour $j = 1, \cdots, k$ **faire**
    $H_{j,k} = (u)_{(p)_j}$;
    $u = u - H_{j,k} l^j$;
  fin
  si ($k < N$ *et* $u \neq 0$) **alors**
    On détermine $i_0 \in \{k+1, \cdots, N\}$ tel que $\left|(u)_{(p)_{i_0}}\right| = \left\|u_{(p)_{k+1}:(p)_N}\right\|_\infty$;
    $H_{k+1,k} = (u)_{(p)_{i_0}}$;
    $l^{k+1} = u/H_{k+1,k}$;
    $(p)_{k+1} \leftrightarrow (p)_{i_0}$;
  fin
  $H_{k+1,k} = 0$, Stop;
fin

au cas où on a atteint le degré du polynôme minimal pour le vecteur v. Dans ce cas, l'algorithme peut être arrêté. On remarque que cette algorithme diminue le nombre de calculs par itérations par rapport à l'algorithme d'Arnoldi. En effet, calculer $u = Al^k$ demande $N(N-k)$ multiplications. Le calcul de $u = u - H_{j,k} l^j$ pour $j = 1, \cdots, k$ requiert $\frac{(k+1)(2N-k)}{2}$ multiplications. Si l'on compare à Arnoldi : $u = Av_k$ entraîne N^2 multiplications et $u = u - H_{j,k} v^j$ pour $j = 1, \cdots, k$ requiert Nk multiplications. Après avoir réduit le coût de calcul, il a été montré par M. Heyouni et H. Sadok [36] que la méthode CMRH réduisait aussi le coût de stockage. En effet, ils proposent l'algorithme de Hessenberg avec stockage des matrices L_k et H_k dans la matrice A. Ils ont remarqué qu'à la k^e itération de l'algorithme 26, L_k était une matrice unitaire trapézoïdale inférieure. Du coup, lorsque l'on calcule $u = Al^k$, les $k-1$ premières colonnes de A

ne sont pas utilisées. Ils ont donc proposé de remplacer la partie triangulaire supérieure de A par la partie triangulaire supérieure de la matrice H_k. De même, la partie inférieure de A est remplacée par la partie inférieure de la matrice L_k. Seul reste un vecteur h correspondant à la sous diagonale de la matrice H_k. On donne l'algorithme de Hessenberg avec pivotage et stockage dans l'algorithme 27. Cet algorithme sera très intéressant pour la parallélisation de la méthode que l'on donnera dans la section 6.4.

Algorithme 27: Algorithme de Hessenberg avec stockage

Initialisation : $p = [1, 2, \cdots, n]^T$;
On détermine i_0 tel que $\left|(v)_{i_0}\right| = \|v\|_\infty$;
$\beta = (v)_{i_0},\ v = v/\beta,$;
$p_1 \leftrightarrow (p)_{i_0},\ (v)_1 \leftrightarrow (v)_{i_0}$;
$A_{1,:} \leftrightarrow A_{i_0,:},\ A_{:,1} \leftrightarrow A_{:,i_0}$;
pour $k = 1, \cdots, m$ **faire**
  $u = A_{:,k} + A_{:,k+1:n}(v)_{k+1:n}$;
  $A_{k+1:n,k} = (v)_{k+1:n}$;
  pour $j = 1, \cdots, k$ **faire**
    $A_{j,k} = u_j,\ u_j = 0$;
    $u_{j+1:n} = u_{j+1:n} - A_{j,k}A_{j+1:n,j}$;
  fin
  On détermine i_0 tel que $\left|u_{i_0}\right| = \|u\|_\infty$;
  $(h)_{k+1} = u_{i_0}$;
  $v = u/(h)_{k+1}$;
  $(p)_{k+1} \leftrightarrow (p)_{i_0},\ (v)_{k+1} \leftrightarrow (v)_{i_0}$;
  $A_{k+1,:} \leftrightarrow A_{i_0,:},\ A_{:,k+1} \leftrightarrow A_{:,i_0}$;
fin

L'algorithme 15 de la méthode CMRH utilise le processus de Hessenberg avec pivotage et stockage des matrices dans A, appliqué à la matrice A et au vecteur x_0 comme défini dans la section 2.5.2.

6.3 Méthode CMRH préconditionnée

6.3.1 Algorithme de la méthode CMRH préconditionnée

Comme pour toutes les méthodes itératives, il est utile de combiner la méthode CMRH avec une technique de préconditionnement efficace. Avec la méthode CMRH, contrairement aux autres méthodes et si on veut garder un stockage optimal, on ne peut utiliser que la technique du préconditionnement à gauche. En effet, le principal avantage de la méthode CMRH est son coût en calcul et en mémoire. Donc l'utilisa-

tion d'un préconditionneur ne doit pas faire perdre cette avantage. C'est pourquoi, l'utilisation du préconditionnement à droite n'est pas envisagée. Avec le préconditionnement à gauche, le produit matrice-vecteur permet de garder les avantages à utiliser la méthode CMRH.

Algorithme 28: Méthode CMRH préconditionnée à gauche

Soit x_0;
$p = [1, 2, \cdots, N]^T$; Calculer $b = b - Ax_0$;
Résoudre $Mb = r_0$;
On détermine i_0 tel que $\left|(b)_{i_0}\right| = \|b\|_\infty$;
$\beta = (b)_{i_0}$, $b = b/\beta$, ;
$(p)_1 \leftrightarrow (p)_{i_0}$, $(b)_1 \leftrightarrow (b)_{i_0}$;
$A_{1,:} \leftrightarrow A_{i_0,:}$, $A_{:,1} \leftrightarrow A_{:,i_0}$;
pour $k = 1, \cdots,$ *jusqu'à convergence* **faire**
  $v = A_{:,k} + A_{:,k+1:N}(b)_{k+1:N}$;
  Résoudre $Mu = v$;
  $A_{k+1:N,k} = (b)_{k+1:N}$;
  pour $j = 1, \cdots, k$ **faire**
    $A_{j,k} = (u)_j$, $u_j = 0$;
    $(u)_{j+1:N} = (u)_{j+1:N} - A_{j,k}A_{j+1:N,j}$;
  fin
  On détermine $i_0 \in \{k+1, \cdots, N\}$ tel que $\left|(u)_{(p)_{i_0}}\right| = \left\|(u)_{(p)_{k+1}:(p)_N}\right\|_\infty$;
  $h = (u)_{(p)_{i_0}}$;
  $v = u/h$;
  $(p)_{k+1} \leftrightarrow (p)_{i_0}$, $(v)_{k+1} \leftrightarrow (v)_{i_0}$;
  $A_{k+1,:} \leftrightarrow A_{i_0,:}$, $A_{:,k+1} \leftrightarrow A_{:,i_0}$;
  Factorisation QR de $H_{k+1,k}$;
  On applique les rotations de Givens à $H_{k+1,k}$ et βe_1 ;
fin
On résout $H_k d_k = \beta e_1$, $(H_k = triu(A_{1:k,1:k}))$; ;
On calcule $x_k = x_0 + L_k d_k$, $(L_k = diag(ones(k,1)) + tril(A_{:,1:k}, -1))$; ;
On réordonne les coordonnées de x_k ;
pour $i = 1, \cdots, N$ **faire**
  $b_{(p)_i} = (x_k)_i$;
fin

Soit M une matrice inversible de taille $N \times N$ telle que le système $Mz = b$ peut être résolu facilement. Au lieu de résoudre le système (6.1), on applique la méthode CMRH au système équivalent suivant :

$$M^{-1}Ax = M^{-1}b. \tag{6.4}$$

Dans ce cas, le processus de Hessenberg construit une base du sous espace de Krylov préconditionné à gauche définie par :

$$K_k^M = K_k(M^{-1}A, r_0) = span\{r_0, M^{-1}Ar_0, \cdots, (M^{-1}A)^{k-1}r_0\}. \tag{6.5}$$

Tous les résidus et les normes calculés lors du processus correspondent à un résidu préconditionné, que l'on nommera $z_k = M^{-1}(b - Ax)$, à la place du résidu original $b - Ax$. L'algorithme 28 reprend la méthode CMRH préconditionnée à gauche.

6.3.2 Choix du préconditionneur

Comme nous l'avons expliqué dans le chapitre 1, il n'existe pas de matrice de préconditionnement universelle. Pour les exemples numériques de la section 6.5.3, nous allons utiliser la méthode de Cholesky pour trouver la matrice de préconditionnement. Cette méthode est généralement utilisée pour des préconditionnements à gauche et à droite. La méthode de Cholesky, définie dans l'algorithme 4, s'utilise uniquement pour des matrices définies positives. La matrice que nous allons considérer dans les résultats numériques n'étant pas définie positive, nous décidons donc d'appliquer la méthode de Cholesky à la matrice A^TA. Posons $B = A^TA$. Cette matrice admet une factorisation de Cholesky :

$$B = EE^T,$$

avec E une matrice triangulaire inférieure. La matrice de préconditionnement M sera donnée par :

$$M = E.$$

6.4 Méthode CMRH en parallèle

Dans cette section, nous nous intéressons à la parallélisation de la méthode CMRH. Nous voulons montrer que la méthode CMRH permet, en comparaison avec la méthode GMRES, de réduire sensiblement les temps de calcul et donc les temps de résolution du système (6.1). Il a été énoncé, dans [36, 59, 60] et nous le verrons dans les résultats numériques, que la méthode CMRH avait une convergence comparable à la méthode GMRES. On a aussi vu que la méthode CMRH réduisait le nombre de calculs et de stockage. Nous souhaitons maintenant montrer que la méthode CMRH est aussi bien parallélisable que la méthode GMRES.

On peut voir que l'algorithme 15 peut se restreindre en 2 grandes parties. D'une part, le produit matrice-vecteur, qui est le calcul le plus coûteux. D'autre part, l'échange

des lignes et des colonnes de la matrice A. Nous donnerons en détail, les techniques que nous avons choisies pour réduire le temps de calcul pour ces 2 problèmes.

Il existe de nombreuses bibliothèques parallèles, de nombreuses façons de paralléliser un code. Chaque machine est différente et demande donc une parallélisation adaptée. Dans ce chapitre, on considère uniquement la bibliothèque MPI [29]. La parallélisation de la méthode GMRES a été étudiée par différents auteurs, par exemple [4, 5, 18, 25, 53].

6.4.1 Distribution des données

Commençons par définir le mode de stockage que nous allons utiliser. En effet, on voit que dans l'algorithme 15, nous avons besoin de la matrice A et de trois vecteurs b, p, u pour calculer les itérations de la méthode CMRH. La parallélisation de l'algorithme sur des processeurs à mémoire distribuée requiert la distribution de la matrice A. Différentes distributions de la matrice sont possibles (par ligne, par colonne, par bloc, etc...). Nous pensons que la meilleure, dans notre cas, est la distribution par ligne. Ceci pour optimiser le produit matrice-vecteur. Nous verrons, dans la section suivante, pourquoi cette distribution nous semble être la mieux adaptée.

Pour ce qui est des vecteurs, le vecteur p nous sert exclusivement à réordonner le vecteur solution final, donc son stockage n'est pas très important et n'entraînera pas de transfert supplémentaire. Nous décidons donc, de le stocker sur un processeur. Ensuite pour les deux autres vecteurs, nous verrons que pour optimiser le produit matrice-vecteur, on stockera le vecteur b en entier sur chaque processeur et le vecteur u sera distribué.

Nous allons considérer 3 distributions différentes de la matrice A. Notons A_L la matrice distribuée par ligne, A_C la matrice distribuée par colonne et A_B la matrice distribuée par bloc. Notons Np, le nombre de processus. Supposons que la taille de la matrice soit proportionnelle au nombre de processus. Notons pl et pc, le nombre de blocs par ligne et par colonne respectivement tel que $pl \times pc = Np$. Notons $nl = \frac{N}{Np}$, $nlb = \frac{N}{pl}$ et $ncb = \frac{N}{pc}$. Alors, on définit :

$$A_L = \left(\begin{array}{c} A_{1:nl,:} \\ \hline \vdots \\ \hline A_{(p-1)nl+1:N,:} \end{array}\right) = \left(\begin{array}{c} A_1 \\ \hline \vdots \\ \hline A_{Np} \end{array}\right),$$

$$A_C = \left(\begin{array}{c|c|c} A^{1:nl,:} & \cdots & A^{(p-1)nl+1:N,:} \end{array}\right) = \left(\begin{array}{c|c|c} A^1 & \cdots & A^{Np} \end{array}\right),$$

$$A_B = \left(\begin{array}{c|c|c} A_{1:nlp,1:nlc} & \cdots & A_{1:nlp,(pc-1)nlc+1:N} \\ \hline \vdots & \cdots & \vdots \\ \hline A_{(pl-1)nlp+1:N,1:nlc} & \cdots & A_{(pl-1)nlp+1:N,(pc-1)nlc+1:N} \end{array}\right) = \left(\begin{array}{c|c|c} A_{1,1} & \cdots & A_{1,pc} \\ \hline \vdots & \cdots & \vdots \\ \hline A_{pl,1} & \cdots & A_{pl,pc} \end{array}\right).$$

6.4.2 Produit matrice-vecteur

Intéressons nous maintenant au produit matrice-vecteur. Nous avons rappelé précédemment que ce produit était moins coûteux en terme de calculs que la méthode GMRES. Nous allons montrer, qu'avec une distribution appropriée, le coût en temps peut devenir encore plus intéressant. Étudions les trois cas qui nous intéressent. La figure montre les différentes situations à l'étape k. Regardons plus en détail chaque cas et commençons par la distribution en colonne. On peut donner deux problèmes très importants qui vont prendre beaucoup de temps. D'une part, pour obtenir le vecteur u, on doit passer par une phase de réduction avec transfert. D'autre part, plus le nombre d'itérations augmente, moins on fera de calculs dans les premiers processeurs alors que dans le reste, le nombre d'opérations sera toujours le même. Nous devrons donc attendre que tous les calculs soient terminés dans les derniers processeurs avant de pouvoir commencer la phase de réduction, ce qui n'est pas très productif pour gagner du temps.

Nous avons les mêmes problèmes avec la distribution par bloc. L'avantage de cette distribution par rapport à la précédente, est la distribution du vecteur u. En effet, celle-ci est distribuée sur chaque processeur en un vecteur de taille correspond au nombre de lignes. Ce qui sera un avantage pour la suite du programme. Toutefois même si ce procédé demande moins de temps que le précédent, il n'est pas encore optimal.

Par contre, on voit que la distribution par ligne est bien appropriée pour ce style de calcul. En effet, le nombre d'opérations sur chaque processeur est le même à chaque itération. Nous n'avons plus besoin de faire de transfert de données car le vecteur u sera bien distribué sur chaque processeur. Donc, le gain de temps est optimal. Pour Np processeurs, le temps de calcul du produit matrice-vecteur sera divisé par Np. C'est pour cette raison que l'on a décidé de choisir ce type de distribution de matrice. À l'itération k, le produit matrice-vecteur, avec cette distribution, requiert $(N-k) \times nl$ multiplications sur chaque processeur. Si l'on compare avec la méthode GMRES, avec la même distribution, celle-ci requiert $N \times nl$ multiplications sur chaque processeur.

Comme le vecteur b est distribué sur chaque processeur à l'étape précédente, cela implique un coût en communication pour calculer u. Mais une simple implémentation, utilisant un programme de diffusion, permet de copier le vecteur b sur chaque processeur.

Dans la suite de ce chapitre, on considère uniquement la distribution par ligne de la matrice A.

6.4.3 Factorisation

Le vecteur u obtenu est distribué en vecteur de taille nl sur chaque processeur. Cette distribution locale de u permet d'exécuter la partie suivante de l'algorithme 15

sans aucun transfert de vecteur :

$$(u)_{j+1:N} = (u)_{j+1:N} - A_{j,k} A_{j+1:N,j}.$$

On doit juste utiliser le sous programme daxpy contenu dans la librairie BLAS. Pour cette étape, on doit uniquement transférer $A_{j,k}$ en utilisant une communication globale.

Comparons avec la méthode GMRES. À l'itération k, cette étape comporte k produits scalaires. Ce qui implique des k communications globales pour ce calcul.

Pour le calcul de la norme, euclidienne pour la méthode GMRES et infinie pour la méthode CMRH, les deux méthodes requièrent les mêmes communications. Finalement, les coûts pour le calcul des rotations de Givens sont aussi identiques.

6.4.4 Échanges

Cette étape est exclusivement utilisée par la méthode CMRH. On peut rappeler que cette étape est importante pour réduire les problèmes de rupture ou réduire les pertes de précision. On peut voir dans l'algorithme 15 que la méthode CMRH utilise deux échanges de vecteurs : un échange entre deux lignes et un échange entre deux colonnes. Avec notre distribution par ligne, l'échange entre les colonnes ne demande aucun transfert de données. L'échange s'exécute localement sur chaque processeur. Comme pour le produit matrice vecteur, cette étape réduit le temps par Np par rapport à l'exécution en séquentielle. L'utilisation de la subroutine swap, permet de faire cette étape.

Par contre, l'échange entre deux lignes est l'étape qui utilise le plus grand transfert de données. Si l'échange se situe entre deux processeurs différents, on doit transférer deux vecteurs de taille N.

Les tableaux 6.1 et 6.2 résument les coûts et les transferts possibles des méthodes CMRH et GMRES à l'itération k. Le tableau 6.1 donne le nombre maximum d'opérations et de transferts que peut faire un processeur, tandis que le tableau 6.2 donne le nombre minimum d'opérations et de transferts que peut faire un processeur.

Ces tableaux confirment la différence dans le nombre de calculs entre la méthode GMRES et la méthode CMRH. Cette réduction devient plus importante sur les premiers processeurs au fur et à mesure que k augmente. Il en est de même pour le nombre de transferts de données par processeur. S'il n'y a pas d'échanges, le volume de transferts entre les processeurs passe de $(k+1) \times NP$ pour la méthode GMRES à $k + Np$ pour la méthode CMRH.

	CMRH		GMRES	
	Opérations	Transferts	Opérations	Transferts
Produit matrice-vecteur	$2nl \times (N-k)$		$2nl \times N$	
Produit scalaire			$2k \times nl + Np$	$k \times Np$
$u = u - h_{j,k} v_j$	$2k \times nl$	k	$2k \times nl$	
$\|u\|_2$ ou $\|u\|_\infty$	nl	Np	$2nl + Np + 1$	Np
Échange de colonnes	$nl <->$			
Échange de lignes	$N <->$	N		

TABLE 6.1 – Résumé du coût maximum à l'itération k

	CMRH		GMRES	
	Opérations	Transferts	Opérations	Transferts
Produit matrice-vecteur	$2nl \times (N-k)$		$2nl \times N$	
Produit scalaire			$2k \times nl + Np$	$k \times Np$
$u = u - h_{j,k} v_j$		k	$2k \times nl$	
$\|u\|_2$ ou $\|u\|_\infty$	nl	Np	$2nl + Np + 1$	Np
Échange de colonnes				
Échange de lignes				

TABLE 6.2 – Résumé du coût minimum à l'itération k

6.5 Résultats numériques

Tous les tests ont été réalisés en Fortran 90 en utilisant MPI [29] sur une machine Bull Novascale 5160 avec 16 processeurs Itanium II de 1.5 GHz et 64 GB de mémoire de RAM au total. Dans cette section, nous allons comparer la méthode CMRH et la méthode GMRES sur quelques exemples. Nous commencerons par comparer la convergence des normes des résidus. Nous nous intéresserons ensuite au test du préconditionneur défini dans la section 6.3.2 dans le but d'accélérer la convergence de la méthode CMRH. Enfin, différents tests montreront le comportement de l'implémentation en parallèle que nous venons de définir.

6.5.1 Matrices

Dans cette section, nous définissons les matrices qui seront utilisées dans les différents tests proposés. Nous posons A, la matrice définie par :

$$A_{i,j} = \begin{cases} \frac{2j-1}{n-i+j} \text{ si } j \leq i, \\ \frac{2i-1}{n-i+j} \text{ si } j > i. \end{cases}$$

La matrice B sera la matrice définie par l'équation suivante :

$$B_{i,j} = \begin{cases} -log\left|z_i - z_j\right|, \ i \neq j, \\ -log|r_i| \end{cases} \tag{6.6}$$

où les N points z_i sont choisis de façon aléatoire dans un carré centré à l'origine dans le plan complexe et où les points r_i sont des nombres appartenant à l'intervalle $(0, d_i]$, d_i étant la distance entre le point z_i et son voisin le plus proche. Nous pouvons observer que calculer une sous diagonale de B correspond, à un facteur $-\frac{1}{2\pi}$ près, à évaluer la fonction de Green pour le Laplacien en 2D avec les arguments $z_i - z_j$. Pour plus de détails, voir [33].

Ensuite, nous choisissons une matrice issue de la collection de matrices de l'université de Floride [15] : TSOPF. Cette matrice provient d'un problème de réseau. Cette matrice est non symétrique et d'ordre $N = 38120$ avec $nz = 16171169$ entrées non nulles. Nous avons traité cette matrice comme une matrice dense.

Pour tous les exemples, nous avons choisi le vecteur b de telle sorte que la solution exacte du système est $x = (1, \cdots, 1)^T$ et le vecteur initiale $x_0 = (0, \cdots, 0)^T$. Pour toutes les figures, la courbe solide représente les résultats obtenus par la méthode CMRH tandis que la ligne pointillée est utilisée pour la méthode GMRES.

6.5.2 Analyse de la convergence

Nous comparons la convergence des deux méthodes sur chacune des matrices. Les figures 6.1 à 6.3 montrent le comportement de la norme du résidu en utilisant une échelle logarithmique. Pour les matrices A et B, nous posons $N = 20800$. Comme les figures l'indiquent et comme nous l'avons énoncé précédemment, les courbes restent très proches pour tous les exemples.

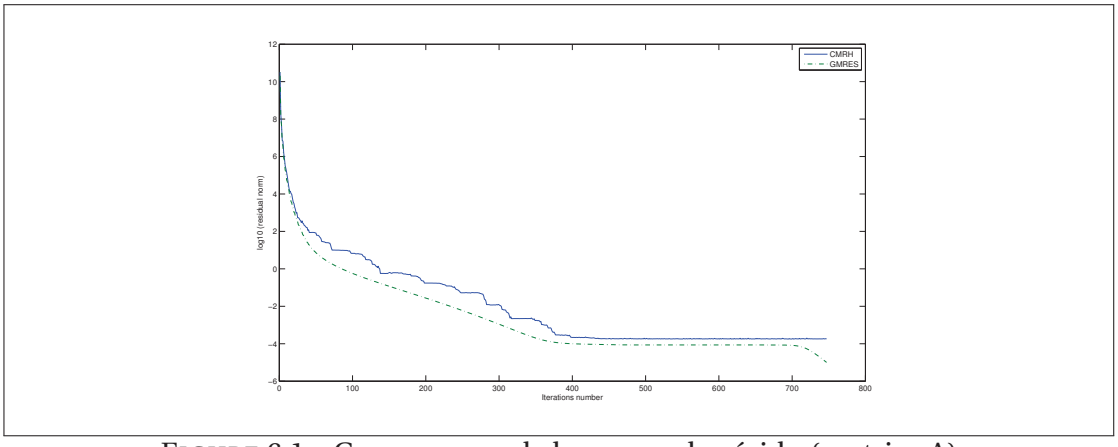


FIGURE 6.1 – Convergence de la norme du résidu (matrice A)

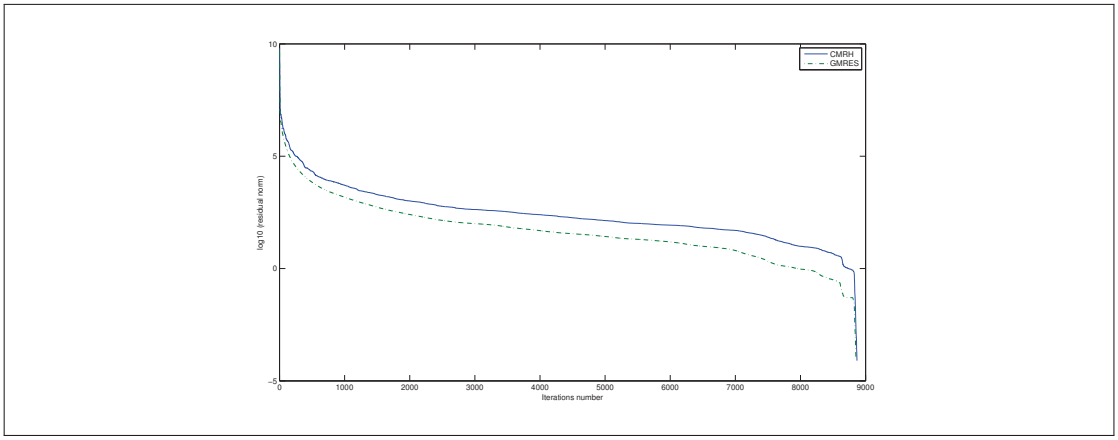


FIGURE 6.2 – Convergence de la norme du résidu (matrice B)

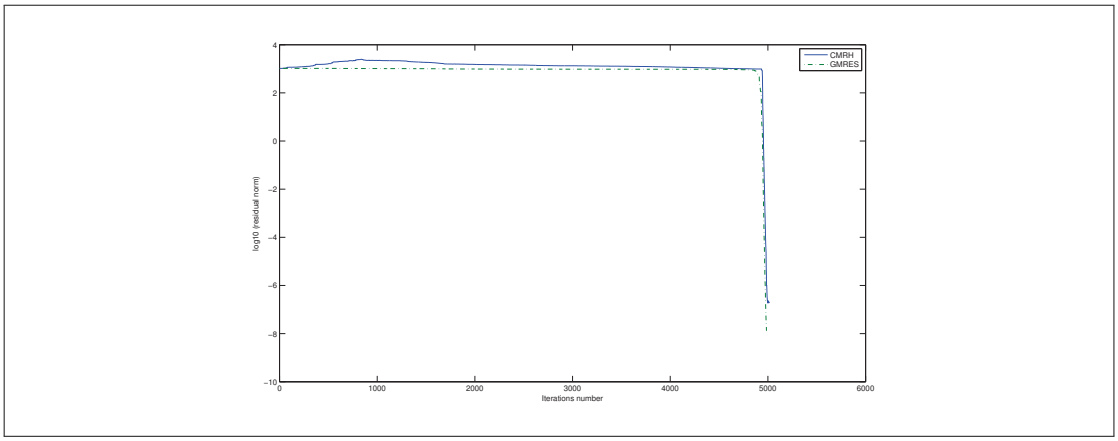


FIGURE 6.3 – Convergence de la norme du résidu (matrice TSOPF)

Pour la matrice A, les deux méthodes convergent en 747 itérations. Pour la matrice B, la méthode GMRES converge en 8851 itérations tandis que la méthode CMRH converge en 8868 itérations. Enfin, pour la matrice TSOPF, le nombre d'itérations est de 5029 pour la méthode CMRH et 4985 pour la méthode GMRES.

6.5.3 Analyse du préconditionnement

On s'intéresse ici à l'accélération de la convergence pour la matrice B. En effet, on a pu remarquer dans la figure 6.2 que le nombre d'itérations était assez important par rapport à la taille de la matrice. Dans cette section, on utilise l'algorithme 28 pour réduire le nombre d'itérations. On décide de préconditionner le système en utilisant la méthode définie dans la section 6.3.2. Les figures 6.4-6.7 montrent la convergence de la norme du résidu relatif $\|r_k\|/\|r_0\|$ des méthodes CMRH et CMRH préconditionnées pour différentes tailles de matrice. Le tableau 6.3 montre le temps CPU d'exécution des programmes. Le comportement des courbes montre l'efficacité du préconditionneur pour cette matrice. Le tableau montre que l'ajout de calculs dû à la résolution des systèmes $Mz = r$ n'affecte pas cette efficacité. Le fait d'augmenter la taille de la matrice permet encore d'améliorer cette accélération.

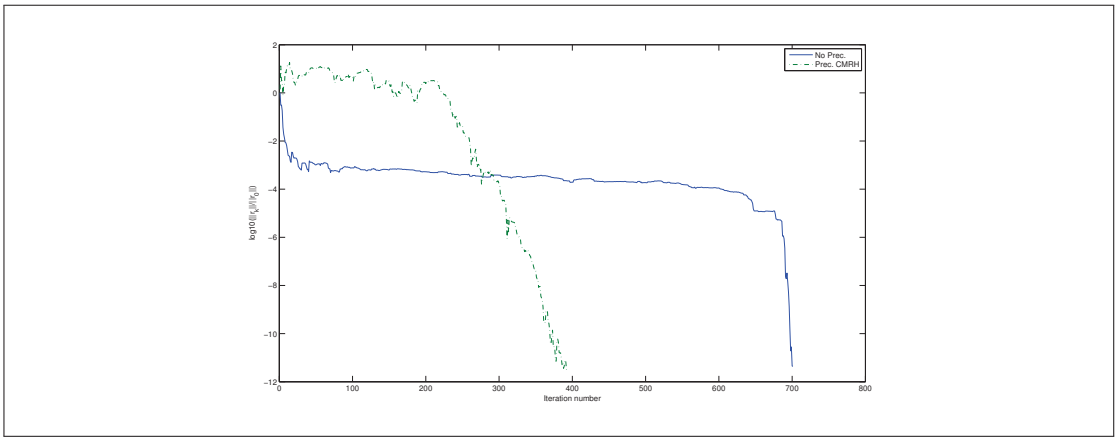


FIGURE 6.4 – Convergence de la méthode CMRH avec et sans préconditonneur (n=1000)

6.5.4 Comparaison de la taille

On a énoncé que la méthode CMRH permettait de résoudre un système linéaire en minimisant le coût en stockage mémoire. Nous allons voir les avantages de cette propriété. Nous avons cherché à connaître la taille de système maximale que l'on pouvait résoudre avec 64GB de RAM. Nous avons réalisé ce test avec la matrice B. Ce test nous a permis de résoudre un système de taille $N = 89600$ en double précision.

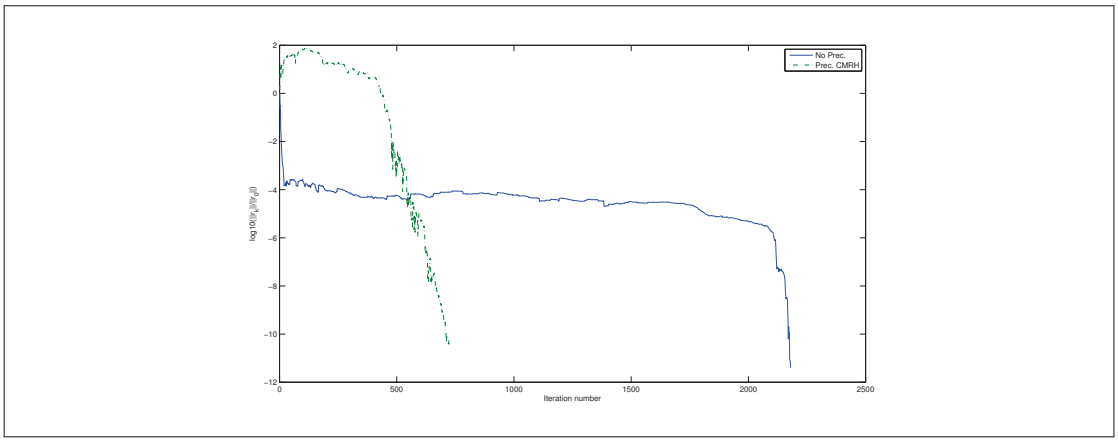


FIGURE 6.5 – Convergence de la méthode CMRH avec et sans préconditonneur (n=4000)

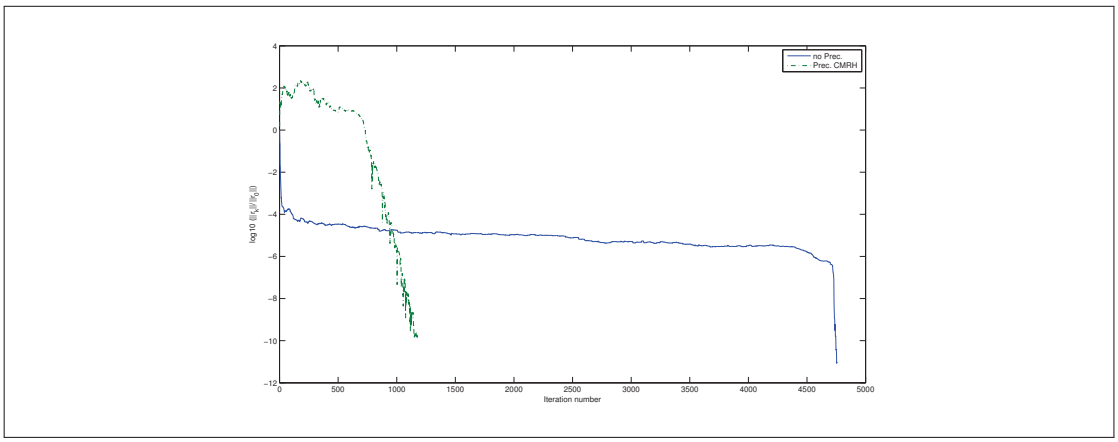


FIGURE 6.6 – Convergence de la méthode CMRH avec et sans préconditonneur (n=10000)

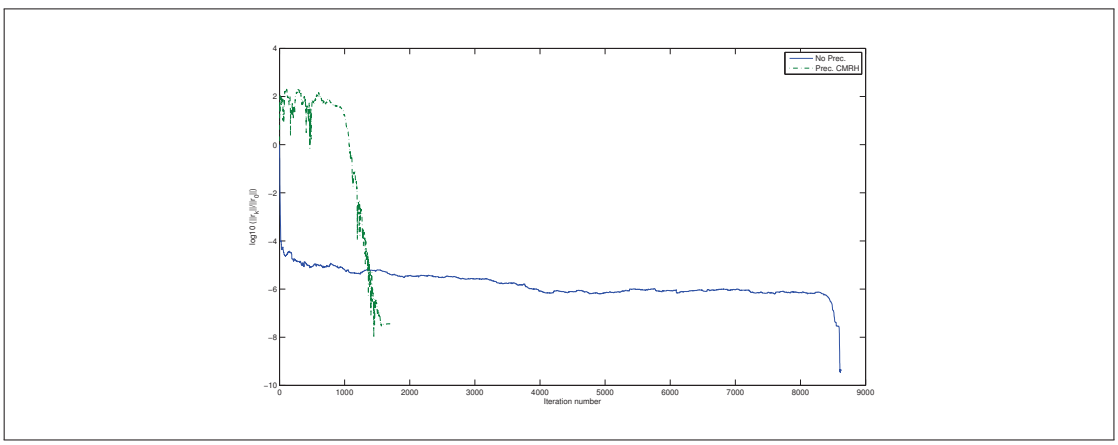


FIGURE 6.7 – Convergence de la méthode CMRH avec et sans préconditonneur (n=20000)

N	1000		4000	
Méthodes	CMRH	CMRH Prec	CMRH	CMRH Prec
Temps CPU (secondes)	10	8	462	222
N	10000		20000	
Méthodes	CMRH	CMRH Prec	CMRH	CMRH Prec
Temps CPU (secondes)	5758	2002	41463	10978

TABLE 6.3 – temps CPU (secondes) des méthodes CMRH et CMRH préconditionnée

Cette taille maximale sera forcément très inférieure pour la méthode GMRES. En effet, cette méthode requiert deux matrices supplémentaires : une de taille $N \times Nmax$ et une de taille $Nmax \times Nmax$, où $Nmax$ est le nombre d'itérations pour résoudre le système.

6.5.5 Comparaison du temps de calcul

Nous comparons le temps CPU de calcul des deux méthodes lorsqu'on utilise l'implémentation parallèle. Les tests ont été réalisés en utilisant 16 processeurs. Nous comparons les temps pour différentes tailles N de la matrice B. Le tableau 6.4 donne les résultats. Pour chaque test, le nombre d'itérations est le même pour les deux méthodes. Le tableau montre que la méthode CMRH est plus rapide que la méthode GMRES pour cet exemple. On peut remarquer que les échanges n'affectent pas la rapidité de la méthode.

N	Itérations	CPU CMRH	CPU GMRES
16000	6688	941	1525
32000	12846	6713	8229
48000	18181	20873	28217

TABLE 6.4 – temps CPU (secondes) des méthodes CMRH et GMRES pour différentes valeurs de N

Ensuite, nous comparons les temps d'exécution de nos deux méthodes en faisant varier le nombre de processeurs. Les tests sont réalisés sur la matrice B avec $N = 32000$ et la matrice $TSOPF$. Les figures 6.8 et 6.9 montrent les résultats. On peut remarquer que pour la première figure, le temps d'exécution est similaire dans le cas séquentiel. Par contre, lorsque le nombre de processeurs augmente, la méthode CMRH est plus rapide que la méthode GMRES. Ce qui confirme les résultats donnés dans le tableau 6.4.

Pour la matrice TSOPF, le comportement est sensiblement le même que pour la matrice B.

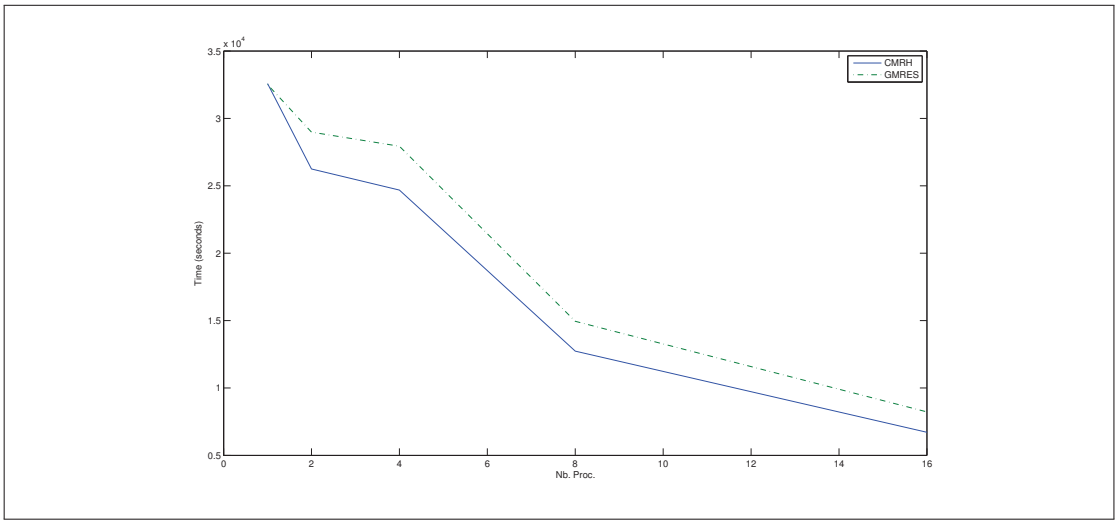


FIGURE 6.8 – Temps CPU (secondes) matrice B ($N = 32000$)

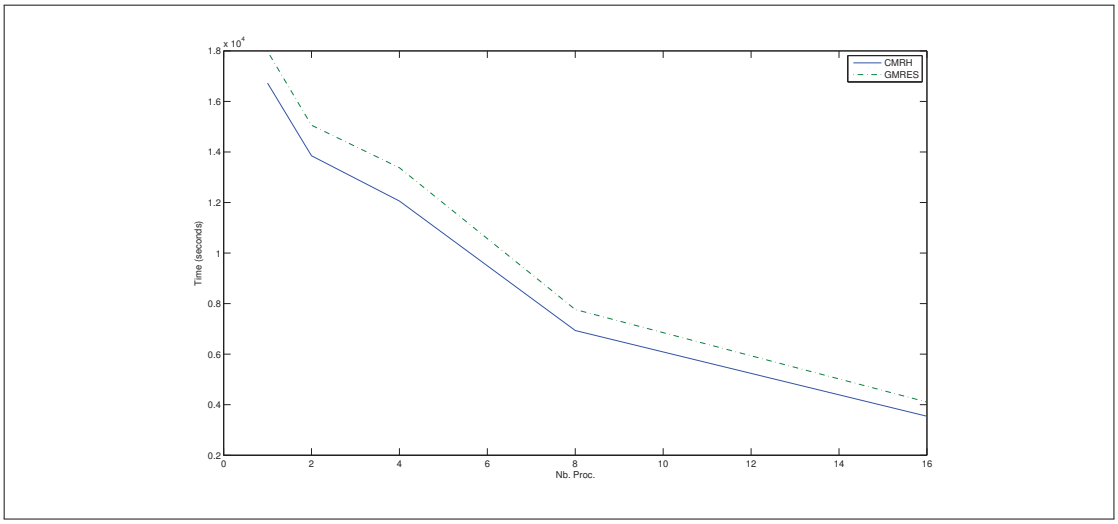


FIGURE 6.9 – Temps CPU (secondes) matrice TSOPF

6.5.6 Rapport de temps

Nous allons maintenant comparer le potentiel d'accélération du temps d'exécution en fonction du nombre de processeurs utilisés. Les figures 6.10 et 6.11 montrent les résultats obtenus pour les matrices B ($N = 32000$) et TSOPF. Dans les deux cas, on observe une meilleure accélération du temps pour la méthode CMRH par rapport à la méthode GMRES. Cette accélération est meilleure dans le cas de la matrice B. Cette accélération tend à être encore plus grande avec l'augmentation du nombre de processeurs.

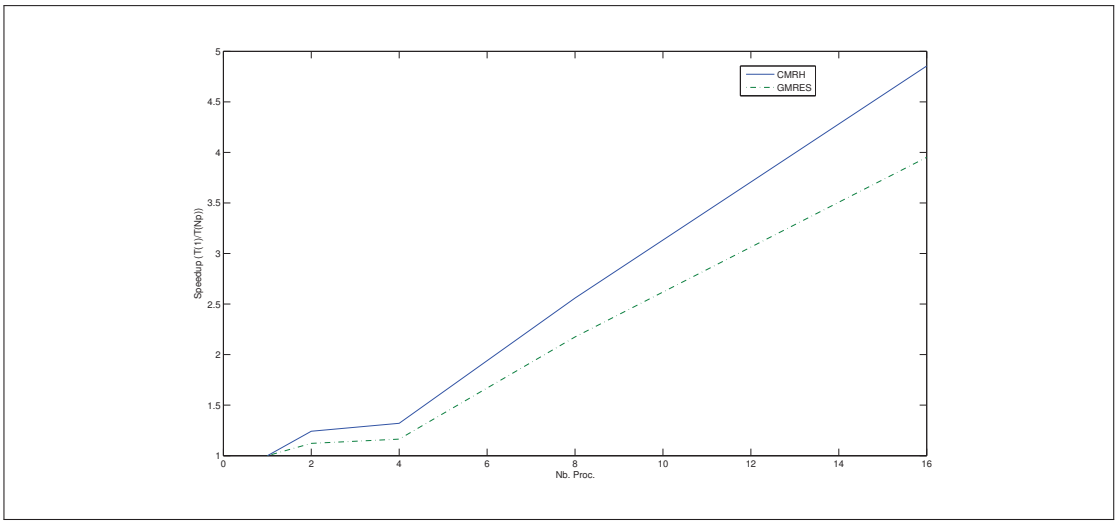


FIGURE 6.10 – Rapport de temps pour la matrice B ($N = 32000$)

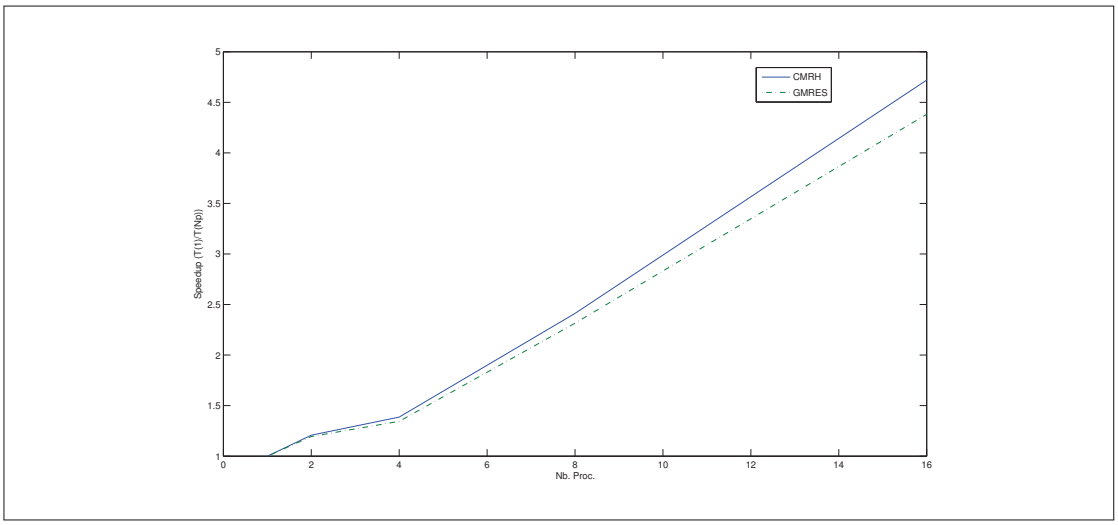


FIGURE 6.11 – Rapport de temps pour la matrice TSOPF

6.6 Conclusion

Dans ce chapitre, on s'est intéressé à montrer l'efficacité de la méthode CMRH par comparaison avec la méthode GMRES. Cette efficacité avait déjà été montrée avec la réduction du coût de calculs et de stockage tout en ayant une convergence similaire.

La recherche de préconditionneurs et la parallélisation des codes représentent un enjeu important pour la recherche actuelle. On a montré, dans ce chapitre, que la méthode CMRH pouvait réduire le temps de résolution de systèmes linéaires en utilisant ces deux techniques. En comparaison avec la méthode GMRES, la méthode CMRH possède une meilleure accélération du temps dans le cas de la résolution en parallèle. Une utilisation sur des machines possédant plus de processeurs permettrait de confirmer les premiers résultats obtenus ici.

Les résultats de ce chapitre ont été publié dans [20].

Conclusion générale et Perspectives

Dans cette thèse, nous nous sommes intéressés à la résolution de systèmes linéaires et non linéaires en utilisant des méthodes d'extrapolation vectorielles polynômiales. Nous avons appliqué ces méthodes à la résolution d'équations aux dérivées partielles telles que l'équation de Schrödinger et les équations de Navier-Stokes. Toutes ces applications ont montré l'intérêt des méthodes d'extrapolation dans l'accélération de la convergence des méthodes de points-fixes (simplemix, Picard) et des méthodes multi-grilles (géométriques et algébriques).

Le grand avantage des méthodes d'extrapolation est qu'elles utilisent uniquement la suite de vecteurs pour créer une nouvelle suite qui peut admettre une convergence quadratique. L'étude de ces méthodes à la résolution de systèmes d'équations linéaires montre qu'elles sont équivalentes à des méthodes de type Krylov. La méthode MMPE étant comparable à la méthode CMRH, on s'est ensuite intéressé plus en détail à cette dernière. Son efficacité avec l'utilisation de préconditionneurs et sa capacité à être paralléliser nous poussent à compléter les premiers résultats observés.

Suite aux travaux réalisés dans cette thèse, nous allons, tout d'abord, poursuivre l'étude sur les méthodes multi-grilles. En effet, nous nous sommes intéressés uniquement à l'étude sur l'équation de Poisson. Nous allons étendre cette étude sur d'autres équations aux dérivées partielles. Ces équations pouvant être linéaires ou non linéaires. Nous pouvons, ensuite, nous intéresser aux méthodes permettant de résoudre des systèmes d'équations ordinaires.

D'autre part, l'étude du préconditionnement de la méthode CMRH peut encore être développée, comme par ailleurs l'étude de la parallélisation. L'utilisation de plus de processeurs permettrait de confirmer les résultats présentés dans cette thèse. L'application à d'autres bibliothèques parallèles ou à des machines à mémoire partagée ou encore à des machines GPU représente quelques uns des nombreux projets à réaliser dans ce domaine.

Bibliographie

[1] AITKEN A.C. « On Bernoulli's numerical solution of algebraic equations ». Dans : *Proc. R. Soc. Edinb.* vol. 46 (1926), p. 289–305 (cf. p. 22).

[2] ANDERSON D.G. « Iterative procedures for nonlinear integral equations ». Dans : *J. Assoc. Comput. Mach.* vol. 12 (1965), p. 547–560 (cf. p. 94).

[3] ARNOLDI W.E. « The principle of minized iterations in the solution of the matrix eigenvalue problem ». Dans : *Quart. Appl. Math.* vol. 9 (1951), p. 17–29 (cf. p. 7).

[4] BAI Z., HU D. et REICHEL L. « A Newton basis GMRES implementation ». Dans : *IMA J. Numeric. Anal.* vol. 14 (1994), p. 563–581 (cf. p. 53, 143).

[5] BAI Z., HU D. et REICHEL L. « Implementation of GMRES method using QR factorisation ». Dans : *Proc. Fifth SIAM Conference on Parallel Processing for Scientific Computing* (1992), p. 84–91 (cf. p. 143).

[6] BATCHELOR G. *An introduction to fluid dynamics*. Cambridge Mathematical Library (second paperback ed.), 2000 (cf. p. 64).

[7] BJORCK A. « Least Squares Methods ». Dans : *Handbook of Numerical Analysis*. T. I. Solution of Equations in R^n. Part 1. Amsterdam : Elsevier/North Holland, 1990, p. 466–647 (cf. p. 4).

[8] BORDNER J. et SAIED F. *MGLab : an interative multigrid environment*. 2007 (cf. p. 126).

[9] BRANDT A. « Multi-level adaptative solutions to boundary-value problems ». Dans : *Math Comput* vol. 31 (1977), p. 333–390 (cf. p. 117).

[10] BREZINSKI C. « Généralisation de la transformation de Shanks, de la table de Padé et de l'epsilon-algorithm ». Dans : *Calcolo* vol. 12 (1975), p. 317–360 (cf. p. 19, 34).

[11] BRIGGS W.L., HENSON V.E. et MCCORMICK S.F. *A multigrid tutorial*. SIAM, 2000 (cf. p. 117).

[12] BROWN P. et SAAD Y. « Hybrid Krylov methods for nonlinear systems of equations ». Dans : *SIAM J. Sci. Stat. Comput.* vol. 11 (1990), p. 450–481 (cf. p. 37).

[13] BROYDEN C.G. « A class of methods for solving nonlinear simulteneous equations ». Dans : *Math. Comp.* vol. 19 (1965), p. 577–593 (cf. p. 89).

[14] CABAY S. et JACKSON L.W. « A polynomial extrapolation method for finding limits and antilimits for vector sequences ». Dans : *SIAM J. Numer. Anal.* vol. 13 (1976), p. 734–752 (cf. p. 19, 30).

[15] DAVIS T. A. et HU Y. « The University of Florida Sparse Matrix Collection ». Dans : *ACM T. Math. Software* vol. 38 (2011), p. 1–25 (cf. p. 147).

[16] DEMBO R.S., EISENTAT S.C. et STEIHAUG T. « Inexact Newton methods ». Dans : *J. Numer. Anal.* vol. 19 (1982), p. 400–408 (cf. p. 15).

[17] DENNIS J.E. et SCHNABEL R.B. *Numerical methods for unconstrained optimization and nonlinear equations*. T. 16. SIAM, Classics in applied mathematics, 1996 (cf. p. 2, 13).

[18] DI BROZZOLO J.J. et ROBERT Y. « Parallel conjugate gradient-like algorithms for solving sparse nonsymmetric linear systems on a vector multiprocessor ». Dans : *Parallel Comput.* vol. 11 (1989), p. 84–91 (cf. p. 143).

[19] DUMINIL S.and Sadok H. « Reduced rank extrapolation applied to electronic structure computations ». Dans : *Electron. T Numer. Ana.* vol. 38 (2011), p. 347–362 (cf. p. 106).

[20] DUMINIL S. « A parallel implementation of the CMRH method for dense linear systems ». Dans : *Numer. Algorithms* (2012). DOI : 10.1007/s11075-012-9616-4 (cf. p. 154).

[21] DUMINIL S., SADOK H. et SILVESTER D. « Fast solvers for discretized Navier-Stokes problems using vector extrapolation (soumis) ». Dans : *J. Comput. Appl. Math.* (2011) (cf. p. 83).

[22] EDDY R.P. « Extrapolation to the limit of a vector sequence ». Dans : *P. C. C. Wang, ed., Information Linkage Between Applied Mathematics and Industry* (1979), p. 387–396 (cf. p. 19, 32).

[23] ELMAN H., RAMAGE A. et SILVESTER D. « Algorithm 866 : IFISS, a Matlab toolbox for modelling incompressible flow ». Dans : *ACM Trans. Math. Softw.* vol. 33 (2007), p. 2–14 (cf. p. 58).

[24] ELMAN H., RAMAGE A. et SILVESTER D. *Finite Elements and Fast Iterative Solvers : with applications in incompressible fluid dynamics*. Oxford University Press, USA, 2005. ISBN : ISBN : 978-0-19-852868-5 ; 0-19-852868-X (cf. p. 58, 63).

[25] ERHEL J. « A parallel GMRES version for general sparse matrices ». Dans : *Electron. T. Numer. Ana.* vol. 3 (1995), p. 307–323 (cf. p. 143).

[26] FANG H. et SAAD Y. « Two classes of Multisecant Methods for Nonlinear Acceleration ». Dans : *Numer. Linear Algebra with Appl.* vol. 16 (2009), p. 197–221 (cf. p. 85, 88, 91, 94).

[27] FLETCHER R. « Conjugate gradient methods for indefinite systems in numerical analysis ». Dans : *G. A. Watson ed. LNM 506, Springer Verlag, Berlin* (1976), p. 73–89 (cf. p. 137).

[28] FORD W.D. et SIDI A. « Recursive algorithms for vector extrapolation methods ». Dans : *Appl. Numer. Math.* vol. 4 (1988), p. 477–489 (cf. p. 19).

[29] FORUM Message Passing Interface. « MPI : A message-passing interface standard ». Dans : *Int. J. Supercomput Ap* (1994) (cf. p. 143, 146).

[30] GRAUSCHOPF T., GRIEBEL M. et REGLER H. « Additive multilevel preconditioners based on bilinear interpolation, matrix dependent geometric coarsening and algebraic multigrid coarsening for second order elliptic PDEs ». Dans : *Institut fur Informatik, Technische Universitat Munchen* (1996) (cf. p. 117, 124, 126).

[31] HACKBUSCH W. *Iterative solution of large sparse systems of equations*. T. 95. Applied mathematical sciences, 1993 (cf. p. 117).

[32] HACKBUSCH W. *Multigrid methods and applications*. T. 4. Springer series in computational mathematics, 1985 (cf. p. 117).

[33] HELSING J. « Approximate inverse preconditioners for some large dense random electrostatic interaction matrices ». Dans : *BIT Numer. Math.* vol. 46 (2006), p. 307–323 (cf. p. 147).

[34] HESSENBERG K. « Behandlung der linearen Eigenwert-Aufgaben mit Hilfe der Hamilton-Cayleychen Gleichung ». Dans : *Darmstadt Dissertation* (1940) (cf. p. 8).

[35] HESTENES M.R. et STIEFEL E.L. « Methods of conjugate grandients for solving linear systems ». Dans : *J. Res. Nat. Bur. Stand.* vol. 49 (1952), p. 409–436 (cf. p. 137).

[36] HEYOUNI M. et SADOK H. « A new implementation of the CMRH method for solving dense linear systems ». Dans : *J. Comput. Appl. Math.* vol. 213 (2008), p. 307–323 (cf. p. 51, 139, 142).

[37] JBILOU K. « A general projection algorithm for solving linear systems of equations ». Dans : *Numer. Algorithms* vol. 4 (1993), p. 361–377 (cf. p. 34).

[38] JBILOU K. et SADOK H. « LU-implementation of the modified minimal polynomial extrapolation method ». Dans : *IMA J. Numer. Anal.* vol. 19 (1999), p. 549–561 (cf. p. 19, 34).

[39] JBILOU K. et SADOK H. « Some results about vector extrapolation methods and related fixed point iterations ». Dans : *J. Comput. Appl. Math.* vol. 36 (1991), p. 385–398 (cf. p. 19, 41).

[40] JBILOU K. et SADOK H. « Vector extrapolation methods. Applications and numerical comparison ». Dans : *J. Comp. Appl. Math.* vol. 122 (2000), p. 149–165 (cf. p. 5, 19, 29, 50).

[41] KELLEY C.T. *Iterative Methods for Linear and Nonlinear Equations*. T. 16. SIAM, Frontiers in Applied Mathematics, 1995 (cf. p. 13, 16, 89–91).

[42] KELLEY C.T. « Solution of the Chandreasekhar H-equation by Newton method ». Dans : *J. Math. Phys.* vol. 21 (1980), p. 1625–1628 (cf. p. 13).

[43] KELLEY C.T. *Solving nonlinear equations with Newton's method*. SIAM, 2003 (cf. p. 90).

[44] KOHN W. et SHAM L.J. « Self-consistent equations including exchange and correlation effects ». Dans : *Phys. Rev.* vol. 140 (1965), p. 1133–1138 (cf. p. 86).

[45] LANCZOS C. « Solution of systems of linear equations by minimized iterations ». Dans : *J. res. Nat. Bur. Stand.* vol. 49 (1952), p. 33–53 (cf. p. 137).

[46] MESINA M. « Convergence acceleration for the iterative solution of x=Ax+f ». Dans : *Comput. Methods Appl. Mech. Eng.* vol. 10 (1977), p. 165–173 (cf. p. 19, 32).

[47] MEURANT G. *Computer solutions of large linear systems*. T. 28. Studies in Mathematics et its Applications, 1999 (cf. p. 117, 119).

[48] NI P. « Anderson acceleration of fixed-point iteration with applications to electronic structure computations ». Thèse de doct. Worcester Polytechnic Institute, 2009 (cf. p. 85, 94).

[49] ORTEGA J.M. et RHEINBOLDT W.C. *Iterative Solution of Nonlinear Equations in Several Variables*. Academic Press, Harcourt Brace Jovanovich, 1970 (cf. p. 2, 13, 42).

[50] OSEEN C.W. « Über die Stokes'sche formel, und über eine verwandte Aufgabe in der Hydrodynamik ». Dans : *Arkiv för matematik* vol. 29 (1910) (cf. p. 64).

[51] PUGATCHEV B.P. « Acceleration of the convergence of iterative processes and a method for solving systems of nonlinear equations ». Dans : *U.S.S.R. Comput. Math. and Math. Phys.* vol. 17 (1978), p. 199–207 (cf. p. 19, 34).

[52] RUGE J.W. et STUBEN K. « Efficient solution of finite difference and finite element equations by algebraic multigrid (AMG) ». Dans : *Multigrids methods for integral and differential equations* (1985), p. 169–212 (cf. p. 117, 124).

[53] SAAD Y. *Iterative methods for sparse linear systems (2nd edition)*. SIAM, PWS, 2003 (cf. p. 5, 8, 143).

[54] SAAD Y. « Krylov subspace methods for solving large unsymmetric linear systems ». Dans : *Math. Comput.* vol. 37 (1981), p. 105–126 (cf. p. 15, 20, 137).

[55] SAAD Y., CHELIKOWSKY J. et SHONTZ S. « Numerical methods for electronic structure calculations of materials ». Dans : *Siam Review* vol. 52 (2010), p. 3–54 (cf. p. 86).

[56] SAAD Y. et SCHULTZ M.H. « GMRES : A generalized minimal residual algorithm for solving nonsymmetric linear systems ». Dans : *SIAM J. Sci. Statist. Comput.* vol. 7 (1986), p. 856–869 (cf. p. 8, 20, 137).

[57] SAAD Y. et al. « Diagonalization methods in PARSEC ». Dans : *Physica Status Solidi (b)* vol. 243 (2006), p. 2188–2197 (cf. p. 86).

[58] SAAD Y. et al. « Solution of large eigenvalue problems in electronic structure calculations ». Dans : *BIT* vol. 36 (1995), p. 199–207 (cf. p. 87).

[59] SADOK H. « CMRH : A new method for solving nonsymmetric linear systems based on the Hessenberg reduction algorithm ». Dans : *Numer. Algorithms* vol. 20 (1999), p. 303–321 (cf. p. 20, 50, 137, 142).

[60] SADOK H. et SZYLD D. B. « A new look at CMRH and its relation to GMRES ». Dans : *BIT Numer. Math.* vol. 52 (2012), p. 485–501 (cf. p. 50, 51, 142).

[61] SHANKS D. « Nonlinear transformations of divergent and slowly convergent sequences ». Dans : *J. Math. Phys* vol. 34 (1955), p. 1–42 (cf. p. 23).

[62] SHAW R. « Optimum Form of a Modified Heine-Abarenkov Model Potential for the Theory of Simple Metals ». Dans : *Phys. Rev.* vol. 174 (1968), p. 769–781 (cf. p. 86).

[63] SIDI A. « Efficient implementation of minimal polynomial and reduced rank extrapolation methods ». Dans : *J. Comput. Appl. Math.* vol. 36 (1991), p. 305–337 (cf. p. 32–34).

[64] SIDI A. « Extrapolation vs. projection methods for linear systems of equations ». Dans : *J. Comput. Appl. Math.* vol. 22 (1988), p. 71–88 (cf. p. 49).

[65] SIDI A., FORD W.F. et SMITH D.A. « Acceleration of convergence of vector sequences ». Dans : *SIAM J. Numer. Anal.* vol. 23 (1986), p. 178–196 (cf. p. 19, 34).

[66] SUN W. et YUAN Y.X. *Optimization theory and methods*. Springer, 2006 (cf. p. 15).

[67] TROTTENBERG U., OOSTERLEE C.W. et SCHÜLLER A. *Multigrid*. London et San Diego, 2001 (cf. p. 117).

[68] YANG C., MEZA J.C. et WANG L.W. « A trust region direct constrained minimization algorithm for the Kohn-Sham equation ». Dans : *SIAM J. Sci. Comput.* vol. 29 (2007), p. 1854–1875 (cf. p. 85, 94).

[69] YANG C. et al. « KSSOLV - A MATLAB toolbox for solving the Kohn-Sham equations ». Dans : *ACM Trans. Math. Softw.* vol. 36.nº 10 (2009), p. 1–35 (cf. p. 85, 94).

Résumé

Nous nous intéressons, dans cette thèse, à l'étude des méthodes d'extrapolation polynômiales et à l'application de ces méthodes dans l'accélération de méthodes de points fixes pour des problèmes donnés. L'avantage de ces méthodes d'extrapolation est qu'elles utilisent uniquement une suite de vecteurs qui n'est pas forcément convergente, ou qui converge très lentement pour créer une nouvelle suite pouvant admettre une convergence quadratique. Le développement de méthodes cycliques permet, de plus, de limiter le coût de calculs et de stockage.

Nous appliquons ces méthodes à la résolution des équations de Navier-Stokes stationnaires et incompressibles, à la résolution de la formulation Kohn-Sham de l'équation de Schrödinger et à la résolution d'équations elliptiques utilisant des méthodes multigrilles. Dans tous les cas, l'efficacité des méthodes d'extrapolation a été montrée.

Nous montrons que lorsqu'elles sont appliquées à la résolution de systèmes linéaires, les méthodes d'extrapolation sont comparables aux méthodes de sous espaces de Krylov. En particulier, nous montrons l'équivalence entre la méthode MMPE et CMRH. Nous nous intéressons, enfin, à la parallélisation de la méthode CMRH sur des processeurs à mémoire distribuée et à la recherche de préconditionneurs efficaces pour cette même méthode.

Mots clés : Extrapolation vectorielle, RRE, MPE, MMPE, Systèmes linéaires, Méthodes de Krylov, CMRH, Implémentation parallèle, CMRH préconditionnée, Systèmes non linéaires, Équations de Navier-Stokes, Équations de Schrödinger, Méthodes multigrilles