HAL
open science

Amélioration d'une méthode de décomposition de domaine pour le calcul de structures électroniques

Guy Bencteux

To cite this version:

Guy Bencteux. Amélioration d'une méthode de décomposition de domaine pour le calcul de structures électroniques. Mathématiques [math]. Ecole des Ponts ParisTech, 2008. Français. NNT : . tel-00391801v2

HAL Id: tel-00391801

https://pastel.hal.science/tel-00391801v2

Submitted on 15 Jun 2009

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

1

THÈSE

présentée pour l'obtention du titre de

Docteur de l'Ecole Nationale des Ponts et Chaussées

Spécialité : Mathématique, Informatique

par

Guy BENCTEUX

Sujet : *Amélioration d'une méthode de décomposition de domaine pour le calcul de structures électroniques*

Soutenue le 18 décembre 2008 devant le jury composé de :

Président :	François-Xavier Roux
Rapporteurs :	Yves Achdou Michel Bercovier
Examinateurs :	François Jollet Eric Lorentz Bernard Philippe
Directeur de thèse :	Claude Le Bris

Remerciements

En 2004, après une dizaine d'années d'expérience en tant qu'ingénieur en R&D à EDF, j'ai souhaité avoir une activité de recherche plus approfondie, dans le cadre d'une thèse. Ce projet n'aurait pu aboutir sans le soutien immédiat de Jean-Louis Vaudescal : c'est donc à lui que je pense en priorité au moment d'écrire ces remerciements.

EDF est une entreprise au fonctionnement complexe et la volonté d'un seul ne suffit pas toujours. A la suite de Jean-Louis, d'autres m'ont épargné des perturbations, inhérentes à la recherche en milieu industriel, qui n'auraient pas été compatibles avec ce type de travail. J'en remercie Stéphane Bugat, Olivier Dubois, Sylvain Leclercq, Eric Lorentz, Jean-Paul Massoud et Françoise Waeckel.

Claude Le Bris a accepté de diriger cette thèse un peu particulière. J'ai apprécié la manière stimulante et efficace avec laquelle il a orienté mon travail. Je souhaite également lui témoigner ma gratitude pour les encouragements qu'il m'a apportés au cours de ces années.

Je suis reconnaissant à Eric Cancès pour sa disponibilité et son aide. Il a suivi de près les développements et m'a suggéré plusieurs améliorations significatives.

Je suis également redevable envers Maxime Barrault. Il a su prendre le temps de me faire partager son intuition du comportement de l'algorithme et m'aider à interpréter dans l'interprétation des résultats.

Je remercie William Hager pour son aide déterminante dans la justification numérique de l'algorithme.

Je suis très reconnaissant à Yves Achdou et à Michel Bercovier d'avoir pris le temps de rapporter sur ce mémoire ainsi qu'à François-Xavier Roux d'avoir présidé le jury de soutenance, et à François Jollet, Eric Lorentz et Bernard Philippe d'avoir accepté de faire partie du jury. Merci pour leurs nombreuses questions !

Au cours de ces années, j'ai pris un grand plaisir à venir travailler régulièrement dans les locaux du CERMICS. Il y règne une atmosphère de réflexion qui m'a beaucoup aidé. Je remercie donc tout particulièrement Serge Piperno de m'y avoir accueilli.

Le groupe "Analyse et Modèle Numérique" à EDF et l'équipe "Simulation Molécu-

laire" du CERMICS ont des fonctionnements et des membres bien différents. Pourtant, j'ai trouvé dans chacune des deux équipes une ambiance agréable où il est facile d'avoir des échanges techniques enrichissants et des moments de détente... eux aussi enrichissants ! Merci à tous. Je tiens à remercier plus particulièrement pour leur aide : Xavier Blanc, Olivier Boiteau, Sébastien Boyaval, Christophe Denis, David Doyen, Sandrine Dyèvre, Frank Hulsemann, Samuel Kortas, Bruno Lathuilière, Tony Lelièvre, Laurent Plagne, Gabriel Stoltz et Alice Tran.

Résumé :

Le travail a porté sur le développement d'une méthode de décomposition de domaine pour le calcul de structures électroniques avec les modèles de Hartree-Fock ou DFT (Density Functional Theory). La simulation de ces modèles passe traditionnellement par la résolution d'un problème aux valeurs propres généralisé, dont la complexité cubique est un verrou pour pouvoir traiter un grand nombre d'atomes.

La méthode MDD (Multilevel Domain Decomposition), introduite au cours de la thèse de Maxime Barrault (2005), est une alternative à cette étape bloquante. Elle consiste à se ramener à un problème de minimisation sous contraintes où on peut exploiter les propriétés de localisation de la solution.

Les résultats acquis au cours de la présente thèse sont :

- l'analyse numérique de la méthode : on a montré, sur un problème simplifié présentant les mêmes difficultés mathématiques, un résultat de convergence locale de l'algorithme ;
- l'augmentation de la vitesse de calcul et de la précision, pour les répartitions "1D" des sous-domaines, ainsi que la démonstration de la scalabilité jusqu'à 1000 processeurs ;
- l'extension de l'algorithme et de l'implémentation aux cas où les sous-domaines sont répartis en "2D/3D".

Abstract :

This work is about a domain decomposition method for electronic structure computations, with Hartree-Fock or DFT (Density Functional Theory) models. Usually, the numerical simulation of these models involve the solution of a generalized eigenvalue problem, which is a bottleneck due to the cubic scaling of the number of operations.

The MDD (Multilevel Domain Decomposition) method, that have been introduced in a previous PhD (Maxime Barrault, 2005), replace the generalized eigenvalue problem with a constrained minimization problem, for which it is easier to take benefit of the localization properties of the solution.

Results produced by the present work are :

- the numerical analysis of the algorithm : a local convergence result has been proved, on a simplified instance of the problem that exhibits the same mathematical difficulties ;
- improvement of speed and accuracy, with one-dimensional sub-domain arrangements, as well as demonstration of scalability up to one thousand processors ;
- extension of the algorithm and its numerical implementation to cases with 2D/3D subdomains arrangement.

Sommaire

1 Introduction générale **1**
1.1 Résolution numérique des modèles Hartree-Fock et Kohn-Sham . . . 2
1.1.1 Bases de discrétisation 6
1.1.2 Le calcul de la densité 7
1.2 L'algorithme MDD 10
1.2.1 Principe 10
1.2.2 Présentation du travail réalisé 12

2 Analyse numérique dans un cas simplifié **15**
2.1 Introduction 16
2.2 Optimality Conditions 20
2.3 The local step and continuity 24
2.4 Convergence of the decomposition algorithm 31
2.5 Numerical experiments 35
2.6 Conclusions 39

3 Domaines alignés : implémentation parallèle **41**
3.1 Introduction and motivation 43
3.2 A new domain decomposition approach 46
3.3 The Multilevel Domain Decomposition (MDD) algorithm 49
3.4 Parallel implementation 52
3.5 Numerical tests 53
3.5.1 General presentation 53
3.5.2 Sequential computations 55
3.5.3 Parallel computations 55
3.6 Conclusions and perspectives 58

4 Domaines alignés : comportement numérique **59**
4.1 Calcul distribué sur un grand nombre de processeurs 60
4.2 Relaxation de la contrainte d'orthogonalité 63
4.2.1 Localisation des orbitales moléculaires et orthogonalité 63
4.2.2 Orthogonalité et convergence de l'algorithme 67
4.2.3 Importance de l'algorithme d'orthogonalisation 70

4.3 Conclusion de la partie 1D 72
4.3.1 Synthèse des améliorations par rapport à [11] 72
4.3.2 Améliorations envisageables pour l'étape locale 72
4.3.3 Comparaison de deux algorithmes de calcul de la densité : MDD version 1D et diagonalisation de la matrice de Fock . . . 74

5 Répartition quelconque des domaines 79
5.1 Adaptation de l'étape globale 81
5.1.1 Retour sur la construction de l'étape globale dans le cas "1D" 81
5.1.2 Adaptation au cas "2D/3D" 84
5.2 Implémentation de la version multidimensionnelle de MDD 86
5.2.1 Structure des données 87
5.2.2 Implémentation 90
5.2.3 Estimation de la mémoire totale 94
5.2.4 Complexité des étapes 96

6 Conclusion générale 97

A Présentation succincte des méthodes d'ordre N 99
A.1 Les méthodes variationnelles 100
A.1.1 Minimisation sur les orbitales 101
A.1.2 Minimisation sur la densité 102
A.2 Les méthodes de projection 102
A.2.1 Approximation de l'opérateur de Fermi 103
A.2.2 Méthode de purification de la densité 103
A.3 Les méthodes de décomposition de domaine 105
A.4 Le passage au cas $S \neq I_{N_b}$ 106

B Calcul de dérivées pour l'étape globale 107
B.1 Notations et rappel des formules 107
B.2 Formules communes aux deux calculs 108
B.2.1 Dérivées partielles de J_i^{-1} 108
B.2.2 Expression des dérivées pour une étape globale à 2 blocs . . . 109
B.2.3 Simplification des traces de produit de matrices 109
B.3 Calcul du gradient 111
B.3.1 Le gradient en $U \neq 0$ 112
B.3.2 Le gradient en $U = 0$ 112
B.4 Calcul du hessien pour 2 blocs 113
B.4.1 Expression du hessien complet pour 2 blocs 117
B.5 Implémentation du calcul du gradient et du hessien pour 2 blocs . . . 119

C Mémoire et nombre d'opérations pour chaque étape 121
C.1 Mémoire 122
C.1.1 Données du problème et solution 123
C.1.2 Etape locale 123
C.1.3 Etape globale 125
C.1.4 Mémoire totale en fonction des options 127
C.2 Nombre d'opérations 127
C.2.1 Initialisation 128
C.2.2 Etape locale 129
C.2.3 Etape globale 129
C.2.4 Nombre d'opérations pour une itération de MDD 131

Bibliographie générale 133

Publications de l'auteur et communications orales

⋆ Publications de l'auteur

A1 G. Bencteux, M. Barrault, E. Cancès, W. W. Hager and C. Le Bris (2008) *Domain decomposition and electronic structure computations : a promising approach* in : Partial Differential Equations. Modeling and Numerical Simulation, 147-164, Roland Glowinski, Pekka Neittaanmaki (eds.), Springer.

A2 M. Barrault, G. Bencteux, E. Cancès, W. W. Hager, C. Le Bris (2007) *Domain Decomposition for Electronic Structure Computations*, in : NIC Series Volume 38 : Parallel Computing : Architectures, Algorithms and Applications, Proceedings ParCo 2007 Conference 4. - 7. September 2007 C. Bischof, M. Bücker, P. Gibbon, G.R. Joubert, T. Lippert, B. Mohr, F. Peters (Eds.)

A3 W. Hager, G. Bencteux, E. Cancès, C. Le Bris (2007) *Analysis of a quadratic programming decomposition algorithm*, INRIA report n°6288, soumis à publication dans : SIAM Journal on Numerical Analysis en septembre 2007.

⋆ Communications orales à des congrès et workshops

I1 4th International Workshop on Parallel Matrix Algorithms and Applications September 7-9, 2006, IRISA, Rennes, France.

I2 SMAI 2007, congrès national de mathématiques appliquées et industrielles. 4-8 juin 2007, Praz-sur-Arly.

I3 ParCo 2007, Parallel Computing, September 4-7, 2007, Jülich, Germany.

I4 Numerical Methods in Density Functional Theory 23. - 25. July 2008 TU Berlin.

Chapitre 1

Introduction générale

Modéliser, à l'échelle atomique, des phénomènes où la structure électronique des atomes joue un rôle, nécessite d'adopter une représentation quantique des électrons. C'est en particulier le cas dès que l'on veut décrire les liaisons chimiques entre atomes, la géométrie des molécules, la cohésion des solides, leur conductivité, etc. La chimie computationnelle offre une hiérarchie de modèles dans lesquels les électrons interviennent sous la forme d'une fonction d'onde. Certains modèles sont dits *ab initio* pour indiquer que les calculs se font sans introduction de paramètres d'ajustement du résultat à la réalité expérimentale.

Les modèles de Hartree-Fock (HF) [5] et de Kohn-Sham (Théorie de la Fonctionnelle de la Densité, DFT) [4] sont les plus utilisés par les chimistes et les physiciens. Dans de très nombreuses situations d'intérêt scientifique, ils permettent de décrire correctement l'évolution d'édifices atomiques (molécules ou cristaux) où tous les électrons sont dans leur état fondamental, les structures géométriques stables ou métastables, ou encore les constantes élastiques [3]. Les capacités de calcul actuelles (au sens large : algorithmes, logiciels, calculateurs) permettent de simuler au maximum quelques dizaines de milliers d'atomes[1]. C'est suffisant pour étudier des molécules ordinaires ou des défauts cristallins ponctuels. En revanche, la simulation de gros objets aussi complexes que les joints de grains en science des matériaux, ou les membranes biochimiques, est largement hors de portée à ce jour.

L'objet de ce travail est de poursuivre le développement de l'algorithme "Multilevel Domain Decomposition" (MDD) introduit dans [6, 11], dans la perspective d'augmenter significativement le nombre d'atomes calculables par les modèles HF et DFT.

La fin de ce chapitre est consacrée, dans un premier temps, à la présentation rapide des grandes caractéristiques de la résolution numérique des modèles HF et

[1] Une récente publication [67] décrit des simulations de DFT comportant quelques millions d'atomes. D'une part, cette performance a été obtenue avec des calculateurs parmi les 25 plus puissants du monde, (*top 500* des super-calculateurs [121], classement de juin 2008). D'autre part, l'algorithme d'ordre N utilisé introduit de grosses approximations dans le résultat, voir annexe A.

DFT, et en particulier le problème résolu par l'algorithme MDD. Une seconde section sert à présenter l'algorithme lui-même et à annoncer les principaux résultats obtenus au cours de cette thèse.

Dans tout le document, on note :
- $\mathcal{M}^{n,m}$ l'ensemble des matrices réelles à n lignes et m colonnes,
- $\mathcal{M}_S^n$ l'ensemble des matrices symétriques réelles d'ordre n.

1.1 Résolution numérique des modèles Hartree-Fock et Kohn-Sham

Pour un ensemble d'atomes ayant des positions données et comportant au total N électrons, les modèles DFT et HF permettent de calculer l'énergie et les forces à partir de la minimisation d'une fonctionnelle, appelée *énergie totale*, de la forme :

$$\mathcal{E}(\phi_1, \ldots, \phi_N), \tag{1.1}$$

faisant apparaître N fonctions $\phi_i \in H^1(\mathbb{R}^3)$ appelées les *orbitales moléculaires* du système, et soumises aux contraintes :

$$\int_{\mathbf{R}^3} \phi_i \phi_j = \delta_{ij}, \ \forall\, 1 \leq i, j \leq N. \tag{1.2}$$

Bien que ces modèles puissent se formuler mathématiquement de la même manière et soient tous deux dérivés de l'équation de Schrödinger pour les N électrons du système, ils diffèrent nettement sur le contenu physique et les performances. Les détails mathématiques de leur construction sont exposés dans [1]. On peut résumer grossièrement en disant que :
- le modèle Hartree-Fock est variationnel : on minimise l'énergie exacte (donnée par le modèle de Schrödinger) sur un sous-ensemble strictement plus petit que l'ensemble des fonctions d'ondes à N électrons,
- dans le modèle de Kohn-Sham, on récrit l'équation de Schrödinger sous une forme équivalente qui ne fait intervenir qu'une fonction inconnue $\rho \in L^1(\mathbb{R}^3)$, la densité électronique. Les approximations sont faites sur l'opérateur intervenant dans cette nouvelle équation en ρ : la fonctionnelle de la densité. Beaucoup de fonctionnelles ont été construites en pratique à partir d'arguments physiques [3].

Les deux modèles font intervenir de manière essentielle la densité électronique, qu'ils définissent par la formule :

$$\rho(x) = \sum_{i=1}^{N} \left[\phi_i(x)\right]^2. \tag{1.3}$$

Les fonctions ϕ_i sont parfois discrétisées par différences finies [21, 26], ce qui permet de mettre en œuvre des stratégies multigrilles [27], mais dans la grande

majorité des cas elles sont décomposées sur une base de Galerkin. On se donne une base de N_b fonctions $\{\chi_\mu\}_{\mu=1,N_b}$ et on cherche ϕ_i sous la forme :

$$\forall\, 1 \leq i \leq N, \quad \phi_i = \sum_{\mu=1}^{N_b} C_{\mu i}\chi_\mu. \tag{1.4}$$

Le problème discrétisé s'écrit alors :

$$\inf\Big\{\mathcal{E}_d(CC^t), C \in \mathcal{M}^{N_b,N},\ C^tSC = I_N\Big\} \tag{1.5}$$

où pour les modèles Hartree-Fock (HF) et de Kohn-Sham (DFT), on a respectivement[2] :

$$\left|\begin{array}{rcl} \mathcal{E}_d^{HF}(D) & = & 2\,\mathrm{Tr}(hD) + 2\,\mathrm{Tr}\big(J(D)D\big) - \mathrm{Tr}\big(K(D)D\big), \\ \mathcal{E}_d^{DFT}(D) & = & 2\,\mathrm{Tr}(hD) + 2\,\mathrm{Tr}\big(J(D)D\big) + \mathcal{E}_{xc}(D). \end{array}\right.$$

avec, $\forall\, 1 \leq \mu, \nu \leq N_b$,

$$\begin{array}{rcl} S_{\mu\nu} & = & \displaystyle\int_{\mathbf{R}^3} \chi_\mu\chi_\nu, \\ h_{\mu\nu} = h^L_{\mu\nu} + h^V_{\mu\nu} & = & \displaystyle\frac{1}{2}\int_{\mathbf{R}^3} \nabla\chi_\mu\nabla\chi_\nu + \int_{\mathbf{R}^3} V_{\bar{x}}\chi_\mu\chi_\nu, \quad V_{\bar{x}} = -\sum_{k=1}^{M}\frac{z_k}{|x-\bar{x}_k|}, \\ J(X)_{\mu\nu} & = & \displaystyle\sum_{\kappa,\lambda=1}^{N_b} (\mu\nu|\kappa\lambda)X_{\kappa\lambda}, \\ K(X)_{\mu\nu} & = & \displaystyle\sum_{\kappa,\lambda=1}^{N_b} (\mu\lambda|\nu\kappa)X_{\kappa\lambda}, \\ \forall\, 1 \leq \kappa, \lambda \leq N_b, \quad (\mu\nu|\kappa\lambda) & = & \displaystyle\int_{\mathbf{R}^3}\int_{\mathbf{R}^3} \frac{\chi_\mu(x)\chi_\nu(x)\chi_\kappa(x')\chi_\lambda(x')}{|x-x'|}\,dx\,dx'. \end{array} \tag{1.6}$$

La matrice S est la *matrice de recouvrement* : elle est symétrique définie positive. h est la *matrice hamiltonienne de cœur*, h^L est la matrice de l'énergie cinétique. $V_{\bar{x}}$ désigne le potentiel des noyaux agissant sur les électrons.

[2] On exprime ces fonctionnelles dans les unités atomiques :

$$m_e = 1, \quad e = 1, \quad \hbar = 1, \quad \frac{1}{4\pi\epsilon_0} = 1.$$

De plus, on omet les variables de spin, car elles ne jouent aucun rôle numérique. D'un point de vue physique, cela signifie que l'on travaille avec des modèles sans spin. Les équations présentées ici modélisent des systèmes où le nombre d'électrons est pair et où on ne calcule qu'une orbitale par paire d'électron.

$\mathcal{E}_{xc}(D)$ est l'*énergie d'échange-corrélation*. Cette fonctionnelle est généralement non convexe et non linéaire ; par exemple, pour une fonctionnelle de type LDA [3]

$$\mathcal{E}_{xc}(D) = \int_{\mathbf{R}^3} \rho(x)\varepsilon_{xc}^{LDA}\big(\rho(x)\big)\, dx \quad \text{avec } \rho(x) = 2\sum_{\mu,\nu=1}^{N_b} D_{\mu\nu}\chi_\mu(x)\chi_\nu(x). \tag{1.7}$$

Les coefficients $(\mu\nu|\kappa\lambda)$ sont appelés *intégrales biélectroniques*. Leur calcul est *a priori* de complexité $\mathcal{O}(N_b^4)$.

L'énergie $\mathcal{E}_d$ ne dépend que de la matrice $D = CC^t$, appelée *matrice densité*. Ceci est la conséquence de l'invariance de l'énergie de Hartree-Fock et de Kohn-Sham (1.1) par transformation orthogonale des ϕ_i. Afin de s'affranchir de cette dégénérescence, on peut écrire (1.5) sous la forme équivalente :

$$\inf\Big\{\mathcal{E}_d(D), D \in \mathcal{M}_S^{N_b},\ DSD = D,\ \mathrm{Tr}(DS) = N\Big\} \tag{1.8}$$

Une simplification largement répandue en pratique (essentiellement pour les calculs de DFT) consiste à ne pas simuler tous les électrons du système (on se restreint aux électrons de valence des atomes) et à représenter l'effet des électrons manquant par une contribution supplémentaire dans $\mathcal{E}_d$, appelée *pseudo-potentiel*. Cette approximation n'est ni utilisée ni étudiée dans ce travail. Une présentation mathématique en est donnée dans la thèse de M. Barrault [6].

Plusieurs algorithmes sont utilisés en pratique pour résoudre le problème (1.5), consistant :

- soit à résoudre les équations d'Euler-Lagrange associées,
- soit à minimiser directement la fonctionnelle $\mathcal{E}_d$.

Une revue de ces algorithmes, avec leur analyse mathématique, est donnée dans [1]. Les logiciels les plus utilisés sont basés sur la résolution des équations d'Euler-Lagrange, mais la recherche de nouveaux algorithmes de minimisation directe est toujours active [65, 77, 112], l'un d'eux ayant été récemment implémenté dans le logiciel BigDFT [82].

On précise maintenant les équations d'Euler-Lagrange associées au problème (1.5).

La contrainte dans (1.5) étant symétrique, la matrice des multiplicateurs de Lagrange l'est aussi. On peut utiliser cette remarque pour construire, à partir de tout point critique de (1.5) un point critique C de même énergie et vérifiant : il existe une matrice diagonale E d'ordre N telle que :

$$\left\{\begin{array}{rcl} F(D)C & = & SCE, \qquad E = \mathrm{Diag}(\epsilon_1, \ldots, \epsilon_N), \\ C^tSC & = & I_N, \\ D & = & CC^t. \end{array}\right. \tag{1.9}$$

La matrice $F(D)$ est appelée *matrice de Fock*. Pour les modèles HF et DFT, elle s'écrit :

$$F^{HF}(D) = h + 2\,J(D) - K(D), \quad F^{DFT}(D) = h + 2\,J(D) + F_{xc}(D) \tag{1.10}$$

avec :

$$\forall\, 1 \leq \mu, \nu \leq N_b, \quad \Big(F_{xc}(D)\Big)_{\mu\nu} = \frac{1}{2}\int_{\mathbf{R}^3} \mu_{xc}(\rho)\chi_\mu\chi_\nu \tag{1.11}$$

où $\mu_{xc}(\rho) = \partial\mathcal{E}_{xc}/\partial\rho$ et ρ est définie en (1.7).

Pour obtenir *tous* les points critiques de (1.5), il faut appliquer des transformations orthogonales aux solutions de (1.9), mais résoudre (1.9) est suffisant, car l'énergie et la densité D sont invariantes par ces transformations orthogonales. Les deux premières équations de (1.9) ont la structure d'un problème aux valeurs propres généralisé pour le couple (F, S). On peut supposer (sans perte de généralité si on modélise un isolant ou un semi-conducteur) que pour tout C minimiseur de (1.5) vérifiant (1.9), les nombres ϵ_i sont les N plus petites valeurs propres du couple (F, S) (principe *Aufbau*, [2]).

Le système d'équations (1.9), non linéaire, est toujours résolu de manière itérative. A partir d'une première approximation de F ou de D, on itère jusqu'à convergence les deux opérations suivantes :

- calcul de F à D fixé, suivant les formules (1.10),
- calcul de D à F fixé, en calculant une solution de (1.9) pour les N plus petites valeurs propres ϵ_i.

Cette procédure s'appelle l'algorithme *Self-Consistent Field* (SCF). Plusieurs versions existent, différant par la manière de calculer la matrice densité à l'itération n à partir des résultats obtenus aux itérations précédentes. La convergence n'est garantie que pour certains d'entre eux, comme par exemple l'*Optimal Damping Algorithm* [2, 19].

Les méthodes de calcul de F à D fixé sont spécifiques au modèle HF ou DFT. Pour les modèles Hartree-Fock, il faut calculer les intégrales bi-électroniques. L'utilisation des bases gaussiennes [17] permet d'accélérer le calcul de chacune des intégrales, mais la complexité globale reste en $O(N_b^4)$. La simplification supplémentaire apportée par la méthode multipôle rapide permet d'atteindre une complexité en $N_b \log(N_b)$ [20]. En DFT, l'étape limitante au cours du calcul de F consiste en la résolution d'une équation de Poisson et peut être réalisée en $O(N_b \log(N_b))$ opérations également.

On termine cette introduction du problème résolu par MDD en détaillant deux points importants pour le contexte : les différentes bases de Galerkin utilisées en pratique et le calcul de la densité à partir de la matrice de Fock.

1.1.1 Bases de discrétisation

Plusieurs types de bases de Galerkin sont utilisées. Certaines ont été introduites très récemment dans les simulations.

1.1.1.1 Ondes planes

La base d'ondes planes est le choix naturel pour la simulation de systèmes homogènes et périodiques en DFT[3]. On simule une maille élémentaire du système. Il faut utiliser d'autant plus de fonctions que le volume de la maille est grand. Ses avantages sont listés ci-dessous.

- Il est très simple de diminuer systématiquement l'erreur d'approximation, ce qui permet d'avoir une convergence aussi précise que l'on veut (dans la limite des moyens de calcul disponibles...).
  En contrepartie, même lorsqu'une précision modérée suffit, le nombre de fonctions de bases à prendre en compte est beaucoup plus important que pour les bases d'orbitales atomiques, qui sont présentées dans le paragraphe suivant : on a généralement $N_b \approx 100N$.
- Elle conduit à un opérateur d'énergie cinétique diagonal ($h^L_{\mu\nu}$ dans les équations (1.6)).
- Les fonctions sont orthogonales : $S = I_{N_b}$.
- Le calcul de la matrice de Fock est peu coûteux grâce à la transformée de Fourier rapide.

La base d'ondes planes est malheureusement assez mal adaptée aux systèmes qui ne sont pas périodiques dans toutes les directions d'espace. Dans ce cas, on force la périodicité en plaçant le système dans une supercellule qui est répliquée avec conditions aux bords périodiques. Pour que les répliques ne se perturbent pas mutuellement dans la direction où la périodicité a été introduite artificiellement, elles sont éloignées les unes des autres, ce qui agrandit la taille de la maille élémentaire et augmente d'autant le nombre d'ondes planes à utliser dans le calcul[4].

Enfin, les fonctions de bases étant complètement délocalisées, il n'existe pas, à ce jour, de méthode d'ordre N pour le calcul de la densité (voir 1.1.2).

1.1.1.2 Bases d'Orbitales Atomiques

Les bases d'Orbitales Atomiques sont constituées par la réunion de bases de fonctions associées à chaque atome du système. On parle alors d'approximation LCAO pour *Linear Combination of Atomic Orbitals*. Les Orbitales Atomiques ont été optimisées au préalable pour d'autres calculs sur de petites molécules par ajustement des résultats du modèle à des résultats de référence (des mesures expérimentales,

[3] On n'utilise pas cette base pour le modèle Hartree-Fock, car le coût de calcul du terme d'échange (la matrice K dans les équations (1.6)) est prohibitif.

[4] Une autre approche consiste à périodiser une supercellule minimale de la taille du système et à corriger *a posteriori* les effets d'interaction entre les différentes répliques [23].

ou des résultats numériques de modèles plus précis) [8, 75]. Ces bases sont localisées et permettent d'utiliser des algorithmes de complexité linéaire pour le calcul de la densité électronique à partir de la matrice de Fock (section 1.1.2 et annexe A).

Les orbitales atomiques les plus utilisées sont les gaussiennes contractées, car elles permettent de calculer très rapidement les intégrales bi-électroniques. Une autre particularité des gaussiennes qui rend leur utilisation intéressante est qu'on peut obtenir des simulations d'une bonne précision avec un nombre de fonctions de base assez faible : en général N_b est compris entre 2 et 4 fois le nombre d'électrons N et dans des situations très particulières N_b peut atteindre $10N$. Quelques éléments d'analyse de la grande efficacité des gaussiennes contractées sont développés dans [1, 2].

Cependant, l'utilisation des ces bases est limitée par le fait que l'on ne connaît pas de moyen systématique d'augmenter la précision des calculs, comme c'est le cas pour les ondes planes. Bien que la famille des gaussiennes génère l'espace $H^1(\mathbb{R}^3)$ tout entier ([2], chapitre 7), il est donc difficile d'obtenir des résultats où l'erreur de discrétisation tend effectivement vers 0.

1.1.1.3 Bases localisées systématiques

Les difficultés techniques rencontrées pour obtenir des calculs très précis avec les bases de gaussiennes ont conduit plusieurs équipes à développer des moyens d'utiliser des bases localisées pouvant se raffiner de manière systématique[5] au prix d'une augmentation de la quantité de calculs. On peut citer comme premier exemple l'utilisation de bases d'ondelettes [24, 33, 39].

Une autre technique consiste à développer les orbitales atomiques sur des bases systématiques en contraignant leur support à rester localisé dans une sphère. Les orbitales atomiques sont optimisées au cours de la simulation par minimisation de l'énergie totale (1.5). Dans [16, 50, 83], les orbitales sont développées sur des éléments finis, dans [72, 89] sur des fonctions sinus cardinal périodisées.

1.1.2 Le calcul de la densité

On rappelle ci-dessous les équations définissant la densité D à partir de la matrice de Fock, que l'on note H à partir de ce point et jusqu'à la fin du document, matrice fixée et ne dépendant plus de D :

$$\left\{ \begin{array}{rcl} HC & = & SCE, \qquad E = \mathrm{Diag}(\epsilon_1, \ldots, \epsilon_N), \\ C^t SC & = & I_N, \\ D & = & CC^t. \end{array} \right. \tag{1.12}$$

H est une matrice symétrique d'ordre N_b et S une matrice symétrique définie positive d'ordre N_b. Formellement, D est la matrice du projecteur S-orthogonal sur

[5] La discrétisation des orbitales moléculaires sur une base d'éléments finis répond à ce problème, elle est cependant rarement utilisée [56].

N vecteurs propres associés aux N plus petites valeurs propres du problème aux valeurs propres généralisé défini par H et S.

La matrice D n'est définie de manière unique que si le *gap* entre la N-ème et la $(N+1)$-ème valeur propre est non nul :

$$\gamma = \epsilon_{N+1} - \epsilon_N > 0.$$

Le cas d'égalité entre ces deux valeurs propres est rencontré au cours de la simulation de systèmes métalliques. L'étude des systèmes métalliques est hors du cadre de ce travail et on ne décrit donc pas la manière de résoudre les difficultés numériques posées par le cas $\gamma = 0$ [3].

La manière la plus simple de procéder au regard des équations (1.12) est :

1. Calculer les vecteurs propres C, définissant (par (1.4)) des ϕ_i appelées *orbitales canoniques* quand on calcule une molécule ou *fonctions de Bloch* dans le cas d'un solide périodique.
2. Former la matrice D en effectuant le produit de C par sa transposée.

C'est la voie classique encore proposée par défaut dans tous les codes de calcul.

L'algorithme utilisé pour le calcul des vecteurs propres dépend de la base de discrétisation. Dans les codes basés sur les orbitales atomiques, le rapport N/N_b est assez grand pour que l'utilisation des méthodes directes soit préférable [95]. Le calcul a donc une complexité en N_b^3 et on observe que cette étape domine le temps de calcul total à partir de quelques centaines d'atomes [59]. Dans les codes basés sur les ondes planes, le rapport N/N_b étant relativement petit, la diagonalisation doit être itérative, avec une complexité de $N_b\,N^2$. La résolution de (1.12) intervenant toujours au cours d'une boucle itérative, il n'est pas nécessaire de calculer les vecteurs propres avec une grande précision au cours des premières itérations ; quelques itérations peuvent être suffisantes. La complexité reste cependant en $N_b N^2$ puisqu'il faut orthonormaliser N vecteurs de taille N_b.

De nombreux algorithmes ont été proposés pour éviter la diagonalisation de la matrice de Fock et atteindre une complexité linéaire avec le nombre d'électrons. Ils partent nécessairement d'expressions équivalentes à (1.12) pour définir la densité sans faire intervenir les orbitales canoniques. On peut par exemple définir D comme la solution d'un problème de minimisation sous contraintes :

$$D = \operatorname{arginf}\left\{ \operatorname{Tr}\left(H\ \tilde{D}\right), \tilde{D} \in \mathcal{M}_S^{N_b}, \tilde{D}S\tilde{D} = \tilde{D}, \operatorname{Tr}\left(S\tilde{D}\right) = N \right\}, \tag{1.13}$$

ou comme une fonction de la matrice de Fock :

$$D = \mathcal{H}(H) \quad \text{dès que } \mathcal{H} \text{ vérifie : } \left| \begin{array}{ll} \mathcal{H}(\epsilon_i) = 1 & \forall\, 1 \leq i \leq N \\ \mathcal{H}(\epsilon_i) = 0 & \forall\, N+1 \leq i \leq N_b \end{array} \right. \tag{1.14}$$

On peut espérer obtenir la matrice densité en $O(N)$ opérations grâce au *principe de localité* [40, 45] : pour des systèmes non conducteurs (ce qui correspond à $\gamma > 0$) la densité électronique en un point n'est pas affectée par une perturbation du système localisée loin de ce point. Une conséquence de ce principe est qu'il existe des orbitales moléculaires ayant un support beaucoup plus restreint que les orbitales canoniques [32,48], ces dernières étant en général largement délocalisées. Numériquement, ces conséquences du principe de localité peuvent être plus facilement exploitées si on utilise une base de fonctions localisées dans l'espace réel, ou bien avec une discrétisation par différences finies[6]. Cela se traduit alors d'une part par le fait que la matrice densité est creuse, et d'autant plus creuse que γ est grand [55], et d'autre part par le fait que l'on peut représenter la matrice densité D par CC^t avec $C \in \mathcal{M}^{N_b,N}$ creuse.

De nombreux algorithmes d'ordre N ont été développés sur ce principe de localité (voir par exemple [15, 31, 37] pour des revues) et leur application est donc *a priori* restreinte aux matériaux isolants. L'annexe A présente brièvement les plus connus en les classant en trois catégories :

- *les méthodes variationnelles* transforment (1.13) en un problème sans contraintes. Les variantes viennent de différentes manières d'impliciter les contraintes. On peut citer les méthodes *Density Matrix Minimisation* [47] et *Orbital Minimisation* [53, 73],
- *les méthodes de projection* calculent la densité par la formule (1.14) en choisissant pour $\mathcal{H}$ des fonctions polynômiales ou rationnelles qui sont des approximations de la fonction de Heaviside. Les plus abouties dans cette classe sont les méthodes *Fermi Operator Expansion* [10] et *Density Matrix Purification* [51, 57]. Elles se réduisent numériquement à une suite de produits de matrices creuses, produits que l'on force à rester creux par troncature. Les stratégies de troncature sont difficiles à mettre en œuvre et sont souvent peu documentées dans la littérature. Une analyse numérique et un algorithme de troncature avec contrôle de l'erreur introduite sont proposés dans [61, 62]. Enfin, une analyse mathématique des méthodes de projection a été publiée récemment par Benzi et Razouk [101].
- les méthodes de type *décomposition de domaine*, introduites par Yang [80], consistent à calculer la densité en résolvant localement les équations (1.12) ou (1.9). Elles constituent une application directe du principe de localité mais leur justification mathématique est très incomplète. Le développement de l'algorithme MDD est une tentative pour disposer d'un algorithme de décomposition de domaine reposant sur des bases mathématiques rigoureuses.

[6] Elles sont très difficiles, si ce n'est impossibles, à mettre en œuvre dans une base délocalisée comme par exemple la base d'ondes planes.

1.2 L'algorithme MDD

La première partie de cette section résume les résultats obtenus dans [6, 11]. La seconde annonce les résultats de ce travail.

1.2.1 Principe

Le principe de l'algorithme est de calculer des orbitales moléculaires localisées *a priori*, dans une base de fonctions localisées. Concrètement, la matrice densité est construite à partir de la solution de :

$$\inf\left\{ \mathrm{Tr}\left(H\, C\, C^t\right), C \in \mathcal{M}^{N_b,N}, C^t\, S\, C = I_N \right\}, \tag{1.15}$$

où chaque colonne de C a une structure creuse fixée *a priori*. La réduction du nombre de degrés de liberté dans C permet d'atteindre une complexité linéaire, la contrepartie étant que l'on remplace le problème (1.15) par un problème approché.

Pour reprendre la terminologie des méthodes de décomposition de domaine, on appellera *domaine* un sous-ensemble des fonctions de base. On se donne p domaines non disjoints dont la réunion est l'ensemble des N_b fonctions de base du calcul. Une fois ces domaines fixés, on cherche N fonctions orthogonales, les orbitales moléculaires, chacune d'elle étant astreinte à se développer sur les fonctions de base d'un seul domaine. Cela revient à récrire (1.15) sous la forme :

$$\inf\left\{ \sum_{i=1}^{p} \mathrm{Tr}\left(H_i C_i C_i^t\right), \quad C_i \in \mathcal{M}^{n_i,m_i}(\mathbb{R}), \quad m_i \in \mathbb{N}, \right.$$
$$\left. C_i^t\, S\, C_j = \delta_{ij}\, I_{m_i}\ \forall\, 1 \le i,j \le p,\ \sum_{i=1}^{p} m_i = N \right\}, \tag{1.16}$$

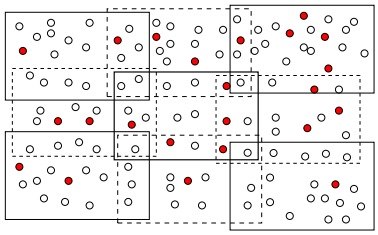
où :

- n_i est le nombre de fonctions de base du domaine i : $0 < n_i \le N_b$.
- m_i est le nombre d'orbitales se développant sur le domaine i : $0 \le m_i \le N$;
- H_i est la matrice de Fock restreinte au domaine i.

L'existence de N orbitales moléculaires pour un système non conducteur est un résultat qualitatif que l'on peut relier au principe de localité. Cependant, il n'existe pas de critère quantitatif permettant de connaître au début du calcul la taille et la disposition des domaines. Il faut une certaine intuition de la solution.

Dans (1.16), on s'est ramené à la minimisation d'une fonctionnelle séparable sous des contraintes qui couplent chacun des termes. Deux domaines i et j sont couplés par la contrainte d'orthogonalité dès que toutes leurs fonctions de base ne sont pas orthogonales deux à deux. On dira dans ce cas que les domaines i et j *se recouvrent* ou qu'ils sont *voisins*. Cette terminologie vient du cas particulier où les χ_μ sont des Orbitales Atomiques orthogonales. Dans ce cas, on peut représenter

les domaines dans l'espace réel (les fonctions d'un domaine sont constituées des Orbitales Atomiques des atomes qui y sont situées) et le recouvrement des domaines en termes de fonctions correspond au recouvrement spatial (voir figure ci-dessous). Au contraire, pour une base d'Orbitales Atomiques avec $S \neq I$, deux domaines peuvent se recouvrir même s'ils n'ont aucun atomes en commun. Enfin, il faut insister sur le fait que la propriété de recouvrement de deux domaines est bien distincte de la propriété de recouvrement de deux fonctions de base. Deux fonctions se recouvrent dès que l'intersection de leur support est non vide. Si $S = I$, deux domaines ne se recouvre pas (au sens défini précédemment) dès qu'ils sont disjoints. Pourtant, les supports des fonctions qui les composent ne sont pas forcément disjoints.



La séparabilité de la fonctionnelle dans (1.16) conduit naturellement à un algorithme de relaxation où on minimise séparément chacun des termes de la somme en tenant compte des contraintes :

$$\inf\left\{ \mathrm{Tr}\left(H_i C_i C_i^t\right), C_i \in \mathcal{M}^{n_i,m_i},\ m_i \in \mathbb{N},\ C_i^t\, S\, C_j = \delta_{ij}\, I_{m_i},\ \forall\, j \text{ voisin de } i \right\}. \quad (1.17)$$

Itérer la résolution de tous les problèmes locaux (1.17) ne permet pas de converger car les contraintes $C_i^t\, S\, C_j = 0$ sont mal traitées. Dans [6], cette difficulté a été résolue en introduisant un problème global couplant tous les domaines. L'idée de cette étape globale est de minimiser la fonctionnelle globale

$$\sum_{i=1}^{p} \mathrm{Tr}\left(H_i C_i C_i^t\right)$$

en combinant entre elles les orbitales moléculaires obtenues par la résolution des problèmes locaux, tout en conservant leur localisation et leur orthogonalité mutuelle. Ces transformations ont été formulées dans le cas où les domaines sont alignés et ne se recouvrent qu'entre premiers voisins, ce qui permet de traiter des systèmes où les atomes sont répartis spatialement au voisinage d'une seule direction : polymères non ramifiés ou nanotubes par exemple.

Il a été constaté sur les résultats numériques obtenus dans ce cadre [6] :

- que l'algorithme développé converge vers une solution proche du minimum de (1.16),
- que le temps de calcul pour atteindre cette solution évolue linéairement avec la taille de la molécule, pour une série de molécules de même nature en termes de localisation des orbitales.

1.2.2 Présentation du travail réalisé

Ce travail présente quelques contributions visant à améliorer la justification mathématique et étendre le domaine d'utilisation de MDD.

- Tout d'abord, on établit dans le chapitre 2 une preuve de convergence de l'algorithme. On ne s'intéresse pas à la pertinence du problème approché (1.16) comme moyen de calculer la matrice densité mais à la capacité de l'algorithme MDD pour minimiser (1.16). En fait, le résultat est établi pour un problème qui n'est pas tout à fait de la forme (1.16) mais qui présente les mêmes difficultés mathématiques : la minimisation d'une fonctionnelle séparable avec contraintes d'orthogonalité. On montre que, à extraction près, les itérés de MDD convergent vers un point stationnaire, pourvu que l'itéré initial soit suffisamment proche de la solution. Des tests numériques sur ce problème simplifié confirment la preuve de convergence.
- Dans les chapitres 3 et 4, on reste dans le cadre des domaines alignés : la parallélisation effective du code développé au cours de la thèse de Maxime Barrault révèle que l'algorithme a une très bonne scalabilité jusqu'à 1000 processeurs. De plus, quelques modifications dans les algorithmes utilisés pour la résolution de chaque étape de MDD permettent d'améliorer significativement les performances. Enfin, on montre, sur des tests numériques, qu'il est nécessaire de relaxer la contrainte d'orthogonalité entre les orbitales moléculaires. C'est la localisation des orbitales qui conduit à introduire cette tolérance. En effet, d'une part, la preuve de convergence du chapitre 2 est obtenue sans introduire d'écart à l'orthogonalité et la différence essentielle entre le problème simplifié de cette preuve et le problème (1.16) est la localisation des vecteurs. D'autre part, on met en évidence cette relation entre localisation et écart à l'orthogonalité sur des tests numériques dans le chapitre 4.
- Dans le chapitre 5, l'algorithme est étendu aux cas où on enlève toute contrainte sur la position relative des domaines, ce qui permet de simuler des situations où la répartition des atomes est homogène dans les trois directions de l'espace. On présente dans un premier temps comment on peut généraliser l'étape globale à ce cas. La fin du chapitre est consacrée à la description de l'implémentation.

Choix des cas tests dans la thèse. Dans ce document, toutes les simulations numériques sont faites à matrice H fixée. Les matrices utilisées correspondent :

- soit à des matrices obtenues à convergence de la boucle SCF pour un modèle *Restricted Hartree-Fock*,
- soit à des matrices H et S intervenant dans un modèle semi-empirique (Extended Hückel Theory [41]) où la matrice de Fock est paramétrée. Il est reconnu que les matrices de Fock des modèles semi-empiriques ont les mêmes caractéristiques numériques que les matrices rencontrées au cours de la boucle SCF [14, 55].

Dans tous les cas, on ne simule que des paires d'électrons. Les tests numériques correspondent à des molécules de polymères isolants car :

- la méthode MDD est limitée à des systèmes où les électrons sont localisés,
- elle a vocation à simuler des systèmes avec un très grand nombre d'atomes, typiquement des systèmes périodiques avec des défauts structuraux.

Pour la version "1D" de MDD, on a simulé des chaînes linéaires hydrocarbonées proches de situations d'intérêt pour les physiciens et les chimistes.

Jusque là on a toujours considéré des quantités réelles ϕ_i, H, ... Ce n'est pas toujours le cas en chimie quantique. La simulation de cristaux comportant une infinité d'atomes s'effectue par périodisation d'une cellule et calcul des ondes de Bloch [13] pour différents points k. Le calcul de la densité passe par la résolution d'une série de problèmes de la forme (1.12) (un pour chaque point k) avec H hermitienne à valeurs complexes.

La simulation des cristaux est une réelle ambition de MDD. Cependant, on ne s'intéressera qu'aux H matrices réelles car la méthode est destinée aux simulations comportant un grand nombre d'atomes. En effet, le nombre de problèmes indépendants à résoudre est d'autant plus petit que la cellule périodisée est grande et pour une cellule contenant quelques milliers d'atomes, le seul cas $k = 0$, pour lequel H est réelle, permet d'obtenir un résultat très précis.

Chapitre 2

Analyse numérique dans un cas simplifié

Ce chapitre reproduit un rapport de recherche [A3] réalisé en collaboration avec W. Hager, E. Cancès et C. Le Bris, qui a été soumis à publication dans : SIAM Journal on Numerical Analysis, en septembre 2007.

Le propos est de justifier mathématiquement l'approche choisie pour l'étape globale. On travaille ici sur une version simplifiée du problème approché (1.16) dans laquelle les orbitales ne sont pas localisées. En revanche, le problème modèle étudié ici présente les mêmes difficultés mathématiques : on minimise une fonctionnelle séparable avec une contrainte d'orthogonalité entre les différents termes.

Analysis of a Quadratic Programming Decomposition Algorithm [1]

W. W. Hager[2], G. Bencteux[3], E. Cancés[4] et C. Le Bris[5]

Abstract. We analyze a decomposition algorithm for minimizing a quadratic objective function, separable in $\mathbf{x}_1$ and $\mathbf{x}_2$, subject to the constraint that $\mathbf{x}_1$ and $\mathbf{x}_2$ are orthogonal vectors on the unit sphere. Our algorithm consists of a local step where we minimize the objective function in either variable separately, while enforcing the constraints, followed by a global step where we minimize over a subspace generated by solutions to the local subproblems. We establish a local convergence result when the global minimizers nondegenerate. Our analysis employs necessary and sufficient conditions and continuity properties for a global optimum of a quadratic objective function subject to a sphere constraint and a linear constraint. The analysis is connected with a new domain decomposition algorithm for electronic structure calculations.

Keywords quadratic programming, orthogonality constraints, domain decomposition method, electronic structure calculations

AMS subject classifications. 65K05, 90C20, 90C46

2.1 Introduction

In [11] and [12] we develop a multilevel domain decomposition algorithm for electronic structure calculations which has been extremely effective in computing electronic structure for large, linear polymer chains. Both the computational cost and memory requirement scale linearly with the number of atoms. Although this algorithm has been very effective in practice, a theory establishing convergence has

[1] August 29, 2007. This material is based upon work supported by the National Science Foundation under Grants 0619080 and 0620286.

[2] hager@math.ufl.edu, http ://www.math.ufl.edu/∼hager, PO Box 118105, Department of Mathematics, University of Florida, Gainesville, FL 32611-8105. Phone (352) 392-0281. Fax (352) 392-8357.

[3] EDF R&D, 1 Avenue du Général de Gaulle, 92141 Clamart Cedex, France and INRIA, MICMAC team project, BP 105, 78153 Rocquencourt, France.

[4] CERMICS, École Nationale des Ponts et Chaussées, 6 & 8, Avenue Blaise Pascal, Cité Descartes, 77455 Marne-La-Vallée Cedex 2, France and INRIA, MICMAC team project, BP 105, 78153 Rocquencourt, France.

[5] CERMICS, École Nationale des Ponts et Chaussées, 6 & 8, Avenue Blaise Pascal, Cité Descartes, 77455 Marne-La-Vallée Cedex 2, France and INRIA, MICMAC team project, BP 105, 78153 Rocquencourt, France.

not yet been developed. The algorithm in [11, 12] was motivated by a related decomposition algorithm for a quadratic programming problem with an orthogonality constraint. In this paper, we develop a convergence theory for the decomposition algorithm.

Let $\mathbf{H}_1$ and $\mathbf{H}_2$ be symmetric n by n matrices. We consider the following quadratic optimization problem :

$$\begin{aligned} \min \quad & F(\mathbf{x}_1, \mathbf{x}_2) := \mathbf{x}_1^\mathsf{T}\mathbf{H}_1\mathbf{x}_1 + \mathbf{x}_2^\mathsf{T}\mathbf{H}_2\mathbf{x}_2 \\ \text{subject to } & \mathbf{x}_1^\mathsf{T}\mathbf{x}_1 = 1 = \mathbf{x}_2^\mathsf{T}\mathbf{x}_2, \quad \mathbf{x}_1^\mathsf{T}\mathbf{x}_2 = 0. \end{aligned} \tag{2.1}$$

In other words, find orthogonal unit vectors $\mathbf{x}_1$ and $\mathbf{x}_2$ which minimize the separable quadratic objective function. Our algorithm for (2.1) consists of a "local step" where we minimize F over each variable separately, while enforcing the constraints, followed by a "global step" where we optimize over a subspace generated by the iterates of the local step. There are two modes of the local step, a "forward" and a "reverse" mode. In consecutive iterations, we employ the forward mode followed by the reverse mode. If $\mathbf{x}_k = (\mathbf{x}_{k1}, \mathbf{x}_{k2})$ is the iterate at step k, then the forward and the reverse modes of the local step are the following :

$$\begin{array}{ll} \text{forward} & \left\{ \begin{array}{lll} \mathbf{y}_{k1} & \in & \arg\min \{F(\mathbf{z}, \mathbf{x}_{k2}) : \|\mathbf{z}\| = 1, \quad \mathbf{z}^\mathsf{T}\mathbf{x}_{k2} = 0\}, \\ \mathbf{y}_{k2} & \in & \arg\min \{F(\mathbf{y}_{k1}, \mathbf{z}) : \|\mathbf{z}\| = 1, \quad \mathbf{z}^\mathsf{T}\mathbf{y}_{k1} = 0\}, \end{array} \right. \\ \text{reverse} & \left\{ \begin{array}{lll} \mathbf{y}_{k2} & \in & \arg\min \{F(\mathbf{x}_{k1}, \mathbf{z}) : \|\mathbf{z}\| = 1, \quad \mathbf{z}^\mathsf{T}\mathbf{x}_{k1} = 0\}, \\ \mathbf{y}_{k1} & \in & \arg\min \{F(\mathbf{z}, \mathbf{y}_{k2}) : \|\mathbf{z}\| = 1, \quad \mathbf{z}^\mathsf{T}\mathbf{y}_{k2} = 0\}. \end{array} \right. \end{array} \tag{2.2}$$

In general, the local steps converge to a limit which may not be a stationary point of (2.1). To achieve convergence to a stationary point for (2.1), each local step, either forward or reverse, is followed by a "global step" where we minimize F over the subspace spanned by the following four vectors, while enforcing the constraints of (2.1) :

$$\begin{bmatrix} \mathbf{y}_{k1} \\ \mathbf{0} \end{bmatrix}, \quad \begin{bmatrix} \mathbf{y}_{k2} \\ \mathbf{0} \end{bmatrix}, \quad \begin{bmatrix} \mathbf{0} \\ \mathbf{y}_{k1} \end{bmatrix}, \quad \begin{bmatrix} \mathbf{0} \\ \mathbf{y}_{k2} \end{bmatrix} \tag{2.3}$$

The global step amounts to minimizing F over the curve defined by

$$\mathbf{z}_k(s) = \frac{1}{\sqrt{1+s^2}}(\mathbf{y}_k + s\mathbf{d}), \tag{2.4}$$

where

$$\mathbf{d} = \pm \begin{bmatrix} \mathbf{y}_{k2} \\ -\mathbf{y}_{k1} \end{bmatrix} \quad \text{and} \quad \mathbf{y}_k = \begin{bmatrix} \mathbf{y}_{k1} \\ \mathbf{y}_{k2} \end{bmatrix}.$$

Let $F_k(s) = F(\mathbf{z}_k(s))$ be the objective function evaluated along the search direction. For convenience, the sign in the definition of $\mathbf{d}$ in the global step is chosen so that $F_k'(0) \leq 0$. At iteration k in the global step, we set

$$\mathbf{x}_{k+1} = \mathbf{z}(s_k), \tag{2.5}$$

for $k = 0, 1, 2, \ldots$

If k is even, perform a forward step :

$$\begin{aligned} \mathbf{y}_{k1} &\in \arg\min \{F(\mathbf{z}, \mathbf{x}_{k2}) : \|\mathbf{z}\| = 1, \quad \mathbf{z}^\mathsf{T}\mathbf{x}_{k2} = 0\}, \\ \mathbf{y}_{k2} &\in \arg\min \{F(\mathbf{y}_{k1}, \mathbf{z}) : \|\mathbf{z}\| = 1, \quad \mathbf{z}^\mathsf{T}\mathbf{y}_{k1} = 0\}, \end{aligned}$$

If k is odd, perform a reverse step :

$$\begin{aligned} \mathbf{y}_{k2} &\in \arg\min \{F(\mathbf{x}_{k1}, \mathbf{z}) : \|\mathbf{z}\| = 1, \quad \mathbf{z}^\mathsf{T}\mathbf{x}_{k1} = 0\}, \\ \mathbf{y}_{k1} &\in \arg\min \{F(\mathbf{z}, \mathbf{y}_{k2}) : \|\mathbf{z}\| = 1, \quad \mathbf{z}^\mathsf{T}\mathbf{y}_{k2} = 0\}. \end{aligned}$$

If k is either even or odd, perform a global step :

$$\mathbf{z}_k(s) = \tfrac{1}{\sqrt{1+s^2}}(\mathbf{y}_k + s\mathbf{d}), \quad \mathbf{d} = \pm \begin{bmatrix} \mathbf{y}_{k2} \\ -\mathbf{y}_{k1} \end{bmatrix}, \quad \mathbf{x}^{k+1} = \mathbf{z}(s_k),$$

where

$$s_k \in \arg\min \{F_k(s) : s \in [0, -\rho F_k'(0)]\}.$$

end

FIG. 2.1 – The decomposition algorithm.

where s_k is the stepsize.

Notice that when $\mathbf{H}_1$ and $\mathbf{H}_2$ are 2 by 2 matrices, the 4 vectors in (2.3) span $\mathbb{R}^4$. Hence, in the 2 by 2 case, the global step yields a global optimum for (2.1). More generally, we find that the local steps steer the iterates into a subspace associated with the eigenvectors of $\mathbf{H}_1$ and $\mathbf{H}_2$ corresponding to the smallest eigenvalues, while the global step finds the best point within this low dimensional subspace.

Ideally, the stepsize s_k is the global minimum of $F_k(s)$ over all s. However, the convergence analysis for this "optimal step" is not easy since $\|\mathbf{x}_k - \mathbf{x}_{k+1}\|$ could be on the order of 1 for all k. For example, if $\mathbf{H}_1 = \mathbf{H}_2$, then $F_k(s)$ is constant, independent of s ; consequently, any choice of s is optimal, and there is no control over the iteration change. To ensure global convergence of the algorithm, we restrict the stepsize to an interval $[0, -\rho F_k'(0)]$, where ρ is a fixed positive scalar. In other words, we take

$$s_k \in \arg\min \{F_k(s) : s \in [0, -\rho F_k'(0)]\}. \tag{2.6}$$

Notice that $s_k = 0$ when $F_k'(0) = 0$ and the global step is skipped. In practice, we observe convergence when s_k is a global minimizer of F_k. The constraint on the stepsize is needed to rigorously prove convergence of the iteration. For reference, the complete algorithm is recapped in Figure 2.1.

Since F is a pure quadratic, the objective function satisfies

$$F(\mathbf{x}_1, \mathbf{x}_2) = F(-\mathbf{x}_1, \mathbf{x}_2) = F(\mathbf{x}_1, -\mathbf{x}_2) = F(-\mathbf{x}_1, -\mathbf{x}_2).$$

Hence, if $\mathbf{y}_{kj}$ is a minimum in a subproblem at iteration k, then so is $-\mathbf{y}_{kj}$. In order to carry out the analysis, it is convenient to choose the signs so that following

inequalities hold :

$$\begin{cases} \mathbf{x}_{k2}^{\mathsf{T}}\mathbf{H}_1\mathbf{y}_{k1} \geq 0 \quad \text{and} \quad \mathbf{y}_{k1}^{\mathsf{T}}\mathbf{H}_2\mathbf{y}_{k2} \geq 0 \quad \text{(forward mode)} \\ \mathbf{x}_{k1}^{\mathsf{T}}\mathbf{H}_2\mathbf{y}_{k2} \geq 0 \quad \text{and} \quad \mathbf{y}_{k2}^{\mathsf{T}}\mathbf{H}_1\mathbf{y}_{k1} \geq 0 \quad \text{(reverse mode)} \end{cases} \tag{2.7}$$

With this sign convention, the multipliers associated with the orthogonality constraints in the local step are always nonnegative as shown in Section 2.4.

Our analysis establishes local, and in some cases global, convergence of the decomposition algorithm of Figure 2.1 to a stationary point. Recall that $\mathbf{y} = (\mathbf{y}_1, \mathbf{y}_2)$ is a stationary point if there exist scalars λ_1, λ_2, and μ such that

$$\begin{bmatrix} \mathbf{H}_1 & \mathbf{0} \\ \mathbf{0} & \mathbf{H}_2 \end{bmatrix} \begin{bmatrix} \mathbf{y}_1 \\ \mathbf{y}_2 \end{bmatrix} = \begin{bmatrix} \lambda_1\mathbf{I} & \mu\mathbf{I} \\ \mu\mathbf{I} & \lambda_2\mathbf{I} \end{bmatrix} \begin{bmatrix} \mathbf{y}_1 \\ \mathbf{y}_2 \end{bmatrix}. \tag{2.8}$$

The condition (2.8) is often called the KKT (Karush-Kuhn-Tucker) condition.

The local step in the decomposition algorithm requires the solution of a quadratic program of the following form :

$$\min \ \mathbf{x}^{\mathsf{T}}\mathbf{H}\mathbf{x} \quad \text{subject to } \|\mathbf{x}\| = 1, \quad \mathbf{a}^{\mathsf{T}}\mathbf{x} = 0, \tag{2.9}$$

where $\mathbf{a} \in \mathbb{R}^n$ with $\|\mathbf{a}\| = 1$ and $\mathbf{H}$ is symmetric. In the decomposition algorithm, $\mathbf{H}$ is $\mathbf{H}_i$ and $\mathbf{a}$ is either $\mathbf{x}_{ki}$ or $\mathbf{y}_{ki}$, $i = 1$ or 2. Since $\mathbf{H}$ is symmetric, we can perform an orthogonal change of variables to diagonalize $\mathbf{H}$. Hence, *without loss of generality*, we can assume that $\mathbf{H}$ is diagonal with the ordered eigenvalues

$$\epsilon_1 \leq \epsilon_2 \leq \ldots \leq \epsilon_n \tag{2.10}$$

The analysis of the decomposition algorithm is based on an analysis of how the multiplier for the constraint $\mathbf{a}^{\mathsf{T}}\mathbf{x} = 0$ in (2.9) depends on $\mathbf{a}$. For almost every choice of $\mathbf{a}$ on the unit sphere, the multiplier is unique and depend continuously on $\mathbf{a}$. Let ϵ_s denote the smallest eigenvalue of $\mathbf{H}$ which is strictly larger than ϵ_1. The degenerate choices of $\mathbf{a}$, where uniqueness and continuity are lost, correspond to those $\mathbf{a}$ which satisfy the equations

$$\sum_{\epsilon_i=\epsilon_1} \frac{a_i^2}{\epsilon_s - \epsilon_1} = \sum_{\epsilon_i>\epsilon_s} \frac{a_i^2}{\epsilon_i - \epsilon_s}, \quad \|\mathbf{a}\| = 1, \quad a_i = 0 \text{ when } \epsilon_i = \epsilon_s. \tag{2.11}$$

We say that $\mathbf{a}$ is degenerate for $\mathbf{H}$ if (2.11) holds, and conversely, $\mathbf{a}$ is nondegenerate if (2.11) is violated. The degenerate choices of $\mathbf{a}$ compose a set of measure 0 on the unit sphere. We say that $(\mathbf{y}_1, \mathbf{y}_2)$ is nondegenerate for (2.1) if $\mathbf{y}_1$ is nondegenerate for $\mathbf{H}_2$ and $\mathbf{y_2}$ is nondegenerate for $\mathbf{H}_1$. If $\mathbf{H}_1$ and $\mathbf{H}_2$ commute, then the solution to (2.1), given in Section 2.3, is nondegenerate.

Our main result is the following :

Theorem 2.1.1 *If the global minimizers of* (2.1) *are all nondegenerate, then for any starting guess sufficiently close to the solution set, there exists a subsequence of the iterates of the decomposition algorithm of Figure* 2.1 *which approaches a stationary point for* (2.1).

Remark 1. In the special case where $\mathbf{H}_1 = \mathbf{H}_2$ and $\epsilon_3 - \epsilon_2 \geq \epsilon_2 - \epsilon_1$, it is shown in [6] that the decomposition algorithm is globally convergent for any starting point. On the other, we observe in Section 2.5 that when $\epsilon_3 - \epsilon_2 < \epsilon_2 - \epsilon_1$, then for specially chosen starting points, the algorithm could converge to a stationary point which is not a global minimum.

In our local convergence result Theorem 2.1.1, the requirement for the starting point ensures that the iterates avoid degenerate points for either $\mathbf{H}_1$ or $\mathbf{H}_2$. Let C_d denote the minimum value for the objective function of (2.1) subject to the additional constraint that either $\mathbf{x}_1$ is degenerate for $\mathbf{H}_2$ or $\mathbf{x}_2$ is degenerate for $\mathbf{H}_1$. Since the global minimizers of (2.1) are nondegenerate, C_d is strictly larger than the minimum value for the objective function. If the objective function at the starting point is strictly less than C_d, then the iterates are bounded away from degenerate points for either $\mathbf{H}_1$ or $\mathbf{H}_2$.

The paper is organized as follows. In Section 2.2 we develop necessary and sufficient optimality conditions for a quadratic optimization problem with both a sphere and an affine constraint, and we develop necessary optimality conditions for (2.1). In Section 2.3 we apply the optimality theory to obtain an optimal solution for the local subproblem, and we show that the multipliers in the subproblems possess a continuity property. The optimality theory also yields the solution to the original problem (2.1) when $\mathbf{H}_1$ and $\mathbf{H}_2$ commute. In Section 2.4 we prove our local convergence result Theorem 2.1.1. In Section 2.5 we investigate the global convergence of the decomposition algorithm using a series of numerical examples.

2.2 Optimality Conditions

Each step of the domain decomposition algorithm requires the solution of a sphere constrained, quadratic programming problem with a linear constraint. This leads us to consider a problem with the structure

$$\min f(\mathbf{x}) := \frac{1}{2}\mathbf{x}^\mathsf{T}\mathbf{H}\mathbf{x} - \mathbf{h}^\mathsf{T}\mathbf{x} \quad \text{subject to } \mathbf{x}^\mathsf{T}\mathbf{x} = 1, \quad \mathbf{A}\mathbf{x} = \mathbf{b}, \tag{2.12}$$

where $\mathbf{A}$ is m by n, $\mathbf{h} \in \mathbb{R}^n$, and $\mathbf{b} \in \mathbb{R}^m$. The local steps of our decomposition algorithm correspond to the case $\mathbf{h} = \mathbf{0}$, $m = 1$, and $b = 0$. Our analysis in this section, however, applies to the more general quadratic cost function and linear constraints appearing in (2.12).

The following result gives necessary and sufficient conditions for a point to be a global minimum. Without the linear constraint, this result is known (see [110]). We give a slightly different analysis which also takes into account linear constraints.

Recall that at a local minimizer where a constraint qualification holds, the Hessian of the Lagrangian is typically positive semidefinite over the tangent space associated with all the constraints, both the linear constraint $\mathbf{Ax} = \mathbf{b}$ and the sphere constraint $\mathbf{x}^\mathsf{T}\mathbf{x} = 1$. If $\mathbf{y}$ is a global minimizer for (2.12) and λ is the Lagrange multiplier for the sphere constraint, then the second-order necessary optimality condition is

$$\mathbf{d}^\mathsf{T}(\mathbf{H} - \lambda\mathbf{I})\mathbf{d} \geq \mathbf{0} \quad \text{whenever } \mathbf{Ad} = \mathbf{0} \quad \text{and} \quad \mathbf{y}^\mathsf{T}\mathbf{d} = 0.$$

In (2.14), we claim that the condition $\mathbf{y}^\mathsf{T}\mathbf{d} = 0$ can be dropped and the Hessian of the Lagrangian is positive semidefinite over a larger space, the null space of $\mathbf{A}$.

Proposition 2.2.1 *Suppose that* $\mathbf{y}$ *is feasible in* (2.12). *A necessary and sufficient condition for* $\mathbf{y}$ *to be a global minimizer is that there exist* $\lambda \in \mathbb{R}$ *and* $\boldsymbol{\mu} \in \mathbb{R}^m$ *such that*

$$\mathbf{Hy} = \mathbf{h} + \mathbf{y}\lambda + \mathbf{A}^\mathsf{T}\boldsymbol{\mu} \tag{2.13}$$

and

$$\mathbf{d}^\mathsf{T}(\mathbf{H} - \lambda\mathbf{I})\mathbf{d} \geq \mathbf{0} \quad \textit{whenever } \mathbf{Ad} = \mathbf{0}. \tag{2.14}$$

Proof. Let $\mathcal{L} : \mathbb{R} \times \mathbb{R}^m \times \mathbb{R}^n \to \mathbb{R}$ be the Lagrangian defined by

$$\mathcal{L}(\lambda, \boldsymbol{\mu}, \mathbf{x}) = f(\mathbf{x}) + \frac{\lambda}{2}(1 - \mathbf{x}^\mathsf{T}\mathbf{x}) + \boldsymbol{\mu}^\mathsf{T}(\mathbf{b} - \mathbf{Ax}).$$

First, suppose that there exist $\lambda \in \mathbb{R}$ and $\boldsymbol{\mu} \in \mathbb{R}^m$ such that (2.13) and (2.14) hold. For any feasible $\mathbf{x}$ for (2.12), a Taylor expansion of $\mathcal{L}$ around $\mathbf{y}$ yields

$$\begin{aligned} f(\mathbf{x}) &= \mathcal{L}(\lambda, \boldsymbol{\mu}, \mathbf{x}) \\ &= \mathcal{L}(\lambda, \boldsymbol{\mu}, \mathbf{y}) + \nabla_x \mathcal{L}(\lambda, \boldsymbol{\mu}, \mathbf{y})(\mathbf{x} - \mathbf{y}) + \frac{1}{2}(\mathbf{x} - \mathbf{y})^\mathsf{T}(\mathbf{H} - \lambda\mathbf{I})(\mathbf{x} - \mathbf{y}) \\ &= f(\mathbf{y}) + \frac{1}{2}(\mathbf{x} - \mathbf{y})^\mathsf{T}(\mathbf{H} - \lambda\mathbf{I})(\mathbf{x} - \mathbf{y}). \end{aligned} \tag{2.15}$$

The first-derivative term in (2.15) vanishes due to (2.13). Since $\mathbf{x}$ is feasible, $\mathbf{A}(\mathbf{x} - \mathbf{y}) = \mathbf{0}$. Hence, (2.14) and (2.15) imply that $f(\mathbf{x}) \geq f(\mathbf{y})$, which shows that $\mathbf{y}$ is a global minimizer for (2.12).

Conversely, suppose that $\mathbf{y}$ is a global minimizer for (2.12). Condition (2.13) is the usual first-order optimality condition at $\mathbf{y}$. This condition holds if the following "constraint qualification" is satisfied (e. g. see [97]) : For each vector $\mathbf{d}$ in the tangent space $\mathcal{T}$ at $\mathbf{y}$, there exists a feasible curve approaching $\mathbf{y}$ along the direction $\mathbf{d}$, where

$$\mathcal{T} = \{\mathbf{d} \in \mathbb{R}^n : \mathbf{y}^\mathsf{T}\mathbf{d} = 0, \quad \mathbf{Ad} = \mathbf{0}\}$$

Given $\mathbf{d} \in \mathcal{T}$, such a feasible curve is given by the formula

$$\mathbf{x}(t) = \mathbf{x}_0 + \left(\frac{\mathbf{y} + t\mathbf{d} - \mathbf{x}_0}{\|\mathbf{y} + t\mathbf{d} - \mathbf{x}_0\|}\right)\|\mathbf{x}_0 - \mathbf{y}\|, \tag{2.16}$$

where t is a scalar and $\mathbf{x}_0$ is the point satisfying the linear equation $\mathbf{Ax} = \mathbf{b}$ which is closest to the origin. Since the expression in parentheses in (2.16) lies in the null space of $\mathbf{A}$ and since $\mathbf{Ax}_0 = \mathbf{b}$, it follows that $\mathbf{Ax}(t) = \mathbf{b}$ for each choice of t. Since $\mathbf{x}_0$ is orthogonal to the null space of $\mathbf{A}$, it follows from the Pythagorean theorem that $\mathbf{x}(t)$ is a unit vector for each choice of t. Differentiating $\mathbf{x}(t)$, we obtain

$$\mathbf{x}'(0) = \mathbf{d} - (\mathbf{y} - \mathbf{x}_0)\left(\frac{(\mathbf{y} - \mathbf{x}_0)^\mathsf{T}\mathbf{d}}{\|\mathbf{y} - \mathbf{x}_0\|^2}\right).$$

Since $\mathbf{d} \in \mathcal{T}$, $\mathbf{y}^\mathsf{T}\mathbf{d} = 0$. Since $\mathbf{x}_0$ is orthogonal to the null space of $\mathbf{A}$ and $\mathbf{Ad} = \mathbf{0}$, we have $\mathbf{x}_0^\mathsf{T}\mathbf{d} = \mathbf{0}$. Hence, $\mathbf{x}'(0) = \mathbf{d}$ and there exists a feasible curve approaching $\mathbf{y}$ in the direction $\mathbf{d}$. This verifies the constraint qualification for (2.12) ; consequently, the first-order condition (2.13) is satisfied for some $\lambda \in \mathbb{R}$ and $\boldsymbol{\mu} \in \mathbb{R}^m$.

By (2.15), the first-order optimality condition (2.13), and the global optimality of $\mathbf{y}$, we have

$$(\mathbf{x} - \mathbf{y})^\mathsf{T}(\mathbf{H} - \lambda\mathbf{I})(\mathbf{x} - \mathbf{y}) = 2(f(\mathbf{x}) - f(\mathbf{y})) \geq 0 \tag{2.17}$$

whenever $\mathbf{x}$ is feasible in (2.12). Suppose that $\mathbf{Ad} = \mathbf{0}$. If in addition, $\mathbf{d}^\mathsf{T}\mathbf{y} = 0$, then $\mathbf{d} \in \mathcal{T}$. Earlier we observed that when $\mathbf{d} \in \mathcal{T}$, $\mathbf{x}(t)$ is feasible in (2.12) for all choices of t. Since $\mathbf{x}(t)$ is feasible, we can substitute $\mathbf{x} = \mathbf{x}(t)$ in (2.17). Since $\mathbf{x}(t) - \mathbf{y} = t\mathbf{d} + O(t^2)$, it follows from (2.17), after dividing by t^2 and letting t approach 0^+, that (2.14) holds. If $\mathbf{d}^\mathsf{T}\mathbf{y} \neq 0$, then $\mathbf{d} \notin \mathcal{T}$, and $\|\mathbf{y} + t\mathbf{d}\| < 1$ for a suitable choice of t near 0. Increase the magnitude of t until $\|\mathbf{y} + t\mathbf{d}\| = 1$. Substituting $\mathbf{x} = \mathbf{y} + t\mathbf{d}$ in (2.17) gives (2.14). □

We now obtain bounds on the location of the multiplier λ associated with (2.12).

Proposition 2.2.2 *If the eigenvalues of* $\mathbf{H}$ *are arranged in increasing order as in* (2.10) *and if* $\mathbf{A}$ *has rank* $k \geq 1$*, then* $\lambda \leq \epsilon_{k+1}$ *when* (2.14) *holds. Moreover, if* $\mathbf{h} = \mathbf{0}$ *and* $\mathbf{b} = \mathbf{0}$ *and* $\mathbf{y}$ *is a global minimizer in* (2.12)*, then* $\lambda \geq \epsilon_1$.

Proof. If $\mathbf{V}$ is an n by k matrix whose columns are an orthonormal basis for the rows of $\mathbf{A}$, then the matrix $\mathbf{P} = \mathbf{I} - \mathbf{VV}^\mathsf{T}$ projects a vector into the null space of $\mathbf{A}$. Define $\mathbf{B} = \mathbf{H} - \lambda\mathbf{I}$. Condition (2.14) is equivalent to

$$\mathbf{x}^\mathsf{T}\mathbf{PBPx} = \mathbf{x}^\mathsf{T}(\mathbf{I} - \mathbf{VV}^\mathsf{T})\mathbf{B}(\mathbf{I} - \mathbf{VV}^\mathsf{T})\mathbf{x} \geq 0 \tag{2.18}$$

for all $\mathbf{x} \in \mathbb{R}^n$. In other words, the matrix $\mathbf{PBP}$ is positive semidefinite.

Expanding in (2.18), we have

$$\begin{aligned}\mathbf{PBP} &= \mathbf{B} - \mathbf{VV}^\mathsf{T}\mathbf{B} - \mathbf{BVV}^\mathsf{T} + \mathbf{VV}^\mathsf{T}\mathbf{BVV}^\mathsf{T} \\ &= \mathbf{B} + \mathbf{WV}^\mathsf{T} + \mathbf{VW}^\mathsf{T} \\ &= \mathbf{B} + \mathbf{PP}^\mathsf{T} - \mathbf{QQ}^\mathsf{T} \end{aligned} \tag{2.19}$$

where

$$\mathbf{W} = -\mathbf{B}\mathbf{V} + \frac{1}{2}\mathbf{V}\mathbf{V}^{\mathsf{T}}\mathbf{B}\mathbf{V}, \quad \mathbf{P} = \frac{\mathbf{W}+\mathbf{V}}{\sqrt{2}}, \quad \text{and} \quad \mathbf{Q} = \frac{\mathbf{W}-\mathbf{V}}{\sqrt{2}}.$$

By an old result, first written down by Weyl [111] (see Parlett [98, p. 192]), the i-th smallest eigenvalue of $\mathbf{B} - \mathbf{Q}\mathbf{Q}^{\mathsf{T}}$ is less than or equal to the i-th smallest eigenvalue of $\mathbf{B}$ for any i. Since $\mathbf{P}$ has rank k, the same result of Weyl also implies (see [98, p. 193]) that the smallest eigenvalue of $(\mathbf{B} - \mathbf{Q}\mathbf{Q}^{\mathsf{T}}) + \mathbf{P}\mathbf{P}^{\mathsf{T}}$ is less than or equal to the $(k+1)$-st smallest eigenvalue of $\mathbf{B} - \mathbf{Q}\mathbf{Q}^{\mathsf{T}}$. Since the $(k+1)$-st smallest eigenvalue of $\mathbf{B} = \mathbf{H} - \lambda\mathbf{I}$ is $\epsilon_{k+1} - \lambda$, the $(k+1)$-st smallest eigenvalue of $\mathbf{B} - \mathbf{Q}\mathbf{Q}^{\mathsf{T}}$ is less than or equal to $\epsilon_{k+1} - \lambda$. Hence, if $\lambda > \epsilon_{k+1}$, $\mathbf{P}\mathbf{B}\mathbf{P}$ is not positive semidefinite and (2.14) cannot hold. Conversely, if (2.14) holds, then $\lambda \le \epsilon_{k+1}$.

Now, suppose that $\mathbf{h} = \mathbf{b} = \mathbf{0}$ and $\lambda < \epsilon_1$. By (2.13), we conclude that $\mathbf{y} = (\mathbf{H} - \lambda\mathbf{I})^{-1}\mathbf{A}^{\mathsf{T}}\boldsymbol{\mu}$. Utilizing the constraint $\mathbf{A}\mathbf{y} = \mathbf{0}$, it follows that

$$\mathbf{A}(\mathbf{H} - \lambda\mathbf{I})^{-1}\mathbf{A}^{\mathsf{T}}\boldsymbol{\mu} = \mathbf{0}.$$

Hence, $\boldsymbol{\mu}$ lies in the null space of $\mathbf{A}^{\mathsf{T}}$. Since $\mathbf{h} = \mathbf{0}$ and $\lambda < \epsilon_1$, it follows that $\mathbf{y} = \mathbf{0}$, which violates the sphere constraint $\|\mathbf{y}\| = 1$. Consequently, $\lambda \ge \epsilon_1$. □

Next, we focus on the original 2-variable problem (2.1).

Corollary 1 *If $(\mathbf{y}_1, \mathbf{y}_2)$ is a local minimizer for (2.1), then there exist λ_1, λ_2, and μ such that the KKT condition (2.8) holds. If $\mathbf{y}$ is a global minimizer for (2.1), then for $i = 1, 2$, we have $\lambda_i \in [\epsilon_{i1}, \epsilon_{i2}]$, where ϵ_{ij} is the j-smallest eigenvalue of $\mathbf{H}_i$,*

$$\epsilon_{i1} \le \epsilon_{i2} \le \ldots \le \epsilon_{in}. \tag{2.20}$$

Proof. The gradients of the constraints for (2.1) at $\mathbf{y}$ are multiples of the 3 vectors

$$\begin{bmatrix} \mathbf{y}_1 \\ \mathbf{0} \end{bmatrix}, \quad \begin{bmatrix} \mathbf{0} \\ \mathbf{y}_2 \end{bmatrix}, \quad \begin{bmatrix} \mathbf{y}_2 \\ \mathbf{y}_1 \end{bmatrix}.$$

Since these vectors are orthogonal, they are linearly independent. Since the "linear independence constraint qualification" is satisfied, the first-order optimality condition (2.8) holds for suitable choices of λ_1, λ_2, and μ. If $\mathbf{y}$ is a global minimizer of (2.1), then $\mathbf{y}_i$ is a global minimizer of the problem

$$\min \ \mathbf{x}^{\mathsf{T}}\mathbf{H}_i\mathbf{x} \quad \text{subject to } \mathbf{x}^{\mathsf{T}}\mathbf{x} = 1, \quad \mathbf{z}^{\mathsf{T}}\mathbf{x} = 0,$$

where $\mathbf{z} = \mathbf{y}_2$ when $i = 1$ and $\mathbf{z} = \mathbf{y}_1$ when $i = 1$. We apply Proposition 2.2.2 with $k = 1$ to obtain $\lambda_i \in [\epsilon_{i1}, \epsilon_{i2}]$, $i = 1, 2$. □

2.3 The local step and continuity

In each step of the domain decomposition algorithm, we must solve a quadratic programming problem of the form (2.9). After an orthogonal change of variables, we can assume, without loss of generality, that $\mathbf{H}$ is diagonal with the ordered eigenvalues (2.10) on the diagonal and $\|\mathbf{a}\| = 1$. Using Propositions 2.2.1 and 2.2.2, we now determine the optimal solutions to (2.9). In the special case $\mathbf{h} = \mathbf{0}$ and $\mathbf{A} = \mathbf{a}^\mathsf{T}$, the first-order optimality conditions (2.13) reduce to

$$\mathbf{H}\mathbf{y} = \mathbf{y}\lambda + \mathbf{a}\mu \tag{2.21}$$

Case 1 : $\epsilon_1 = \epsilon_2$. By Proposition 2.2.2, the multiplier λ of Proposition 2.2.1 is $\lambda = \epsilon_1 = \epsilon_2$. Define the set

$$\mathcal{E}_i = \{j : \epsilon_j = \epsilon_i\}.$$

If $\mu \neq 0$, then by (2.21) we must have $a_i = 0$ for all $i \in \mathcal{E}_1$. If $i \notin \mathcal{E}_1$, then

$$y_i = \frac{\mu a_i}{\epsilon_i - \epsilon_1} \quad \text{and} \quad \mathbf{a}^\mathsf{T}\mathbf{y} = \mu \sum_{i \notin \mathcal{E}_1} \frac{a_i^2}{\epsilon_i - \epsilon_1} \neq 0, \tag{2.22}$$

which violates the orthogonality condition $\mathbf{a}^\mathsf{T}\mathbf{y} = 0$. Hence, $\mu = 0$ and all $\mathbf{y}$ satisfying the following conditions are solutions to (2.9) :

$$y_i = 0 \text{ if } i \notin \mathcal{E}_1, \quad \mathbf{a}^\mathsf{T}\mathbf{y} = 0, \quad \|\mathbf{y}\| = 1. \tag{2.23}$$

Observe that there is an infinite set of solutions $\mathbf{y}$ while the multipliers λ and μ are unique.

Case 2 : $\epsilon_1 < \epsilon_2$ and $a_1 = 0$. If $\lambda > \epsilon_1$, then the second-order condition (2.14) is violated by the vector $d_1 = 1$ and $d_i = 0$ for $i > 1$. Hence, $\lambda = \epsilon_1$. As in Case 1, the orthogonality condition $\mathbf{a}^\mathsf{T}\mathbf{y} = 0$ is violated unless $\mu = 0$. The solution is again given by (2.23) and the multipliers are $\lambda = \lambda_1$ and $\mu = 0$.

Case 3 : $\epsilon_1 < \epsilon_2$ and $a_1 \neq 0$. We first show that $\lambda > \epsilon_1$. Suppose, to the contrary, that $\lambda = \epsilon_1$. The first component of (2.13) implies that $\mu = 0$. Hence, (2.13) reduces to $\mathbf{H}\mathbf{y} = \epsilon_1\mathbf{y}$. Since $\mathbf{H}$ is diagonal and $\epsilon_i > \epsilon_1$ for $i > 1$, we conclude that $y_i = 0$ for $i > 1$. Hence, $y_1 = \pm 1$ since $\mathbf{y}$ is a unit vector. However, a vector of this form violates the orthogonality condition $\mathbf{a}^\mathsf{T}\mathbf{y} = 0$ when $a_1 \neq 0$. This gives a contradiction, so we have $\lambda > \epsilon_1$.

(a) **$a_i \neq 0$ for some $i \in \mathcal{E}_2$.** We show that $\lambda < \epsilon_2$. Suppose, to the contrary, that $\lambda = \epsilon_2$. Since $a_1 \neq 0$ and $a_i \neq 0$ for some $i \in \mathcal{E}_2$, the second-order condition (2.14) is violated by taking $\mathbf{d}$ to be completely zero except for components 1 and i. Since $\epsilon_1 < \lambda < \epsilon_2$, (2.21) can be solved for $\mathbf{y}$:

$$\mathbf{y} = \mu(\mathbf{H} - \lambda\mathbf{I})^{-1}\mathbf{a}. \tag{2.24}$$

If $\mu = 0$, then $\mathbf{y} = \mathbf{0}$, which violates the constraint $\mathbf{y}^\mathsf{T}\mathbf{y} = 1$. We combine the expression (2.24), with the orthogonality condition $\mathbf{a}^\mathsf{T}\mathbf{y} = 0$, and the fact that $\mu \neq 0$ to obtain the equation

$$g(\lambda) := \sum_{i=1}^{n} \frac{a_i^2}{\epsilon_i - \lambda} = 0. \tag{2.25}$$

Observe that g is strictly monotone increasing on the interval (ϵ_1, ϵ_2) and $g(\epsilon_1^+) = -\infty$ since $a_1 \neq 0$ while $g(\epsilon_2^-) = +\infty$ since $a_2 \neq 0$. There exists a unique zero λ of g in (ϵ_1, ϵ_2). The solution to (2.9) is

$$y_i = \frac{\mu a_i}{\epsilon_i - \lambda} \quad \text{where} \quad \mu^2 = \left(\sum_{i=1}^{n} \frac{a_i^2}{(\epsilon_i - \lambda)^2} \right)^{-1} = g'(\lambda)^{-1}. \tag{2.26}$$

The equation for μ^2 is obtained from the requirement that $\mathbf{y}^\mathsf{T}\mathbf{y} = 1$. Notice that both the solution $\mathbf{y}$ and the multiplier μ are unique to within sign.

(b) $a_i = 0$ for all $i \in \mathcal{E}_2$ and $g(\epsilon_2) < 0$. We show that $\lambda = \epsilon_2$ and $\mu = 0$. By (2.21), we have

$$y_i = \frac{\mu a_i}{\epsilon_i - \lambda} \quad \text{when } i \notin \mathcal{E}_2.$$

If $\mu \neq 0$, then the orthogonality condition $\mathbf{a}^\mathsf{T}\mathbf{y} = 0$ reduces to (2.25), which has no solution on (ϵ_1, ϵ_2) since g is monotone on this interval, $g(\epsilon_1^+) = -\infty$, and $g(\epsilon_2) < 0$. Hence, $\mu = 0$. If $\lambda < \epsilon_2$, then (2.24) implies that $\mathbf{y} = \mathbf{0}$, which violates the constraint $\mathbf{y}^\mathsf{T}\mathbf{y} = 1$. Hence, $\lambda = \epsilon_2$ and $\mu = 0$. The solution consists of all vectors $\mathbf{y}$ satisfying

$$y_i = 0 \text{ if } i \notin \mathcal{E}_2, \quad \|\mathbf{y}\| = 1. \tag{2.27}$$

Notice that λ and μ are again unique.

(c) $a_i = 0$ for all $i \in \mathcal{E}_2$ and $g(\epsilon_2) > 0$. First, suppose that $\mu \neq 0$. Since $g(\epsilon_1^+) = -\infty$ while $g(\epsilon_2) > 0$, g in (2.25) has a unique zero on (ϵ_1, ϵ_2). Hence, one solution to (2.21) is given by (2.26). We now consider the possibility that $\mu = 0$ at a global minimum. We will show that this leads to a contradiction. Consequently, there is a unique (to within sign) global minimizer for (2.9) given by (2.26). If $\mu = 0$, then by (2.21), $(\epsilon_i - \lambda)y_i = 0$ for all i, which implies that $y_i = 0$ for $i \notin \mathcal{E}_2$ since $\epsilon_1 < \lambda \leq \epsilon_2$. Since $\|\mathbf{y}\| = 1$, it follows that $\lambda = \epsilon_2$ (or else $\mathbf{y} = \mathbf{0}$, violating the condition $\|\mathbf{y}\| = 1$). We now show that the second-order condition (2.14) is violated for the choice

$$d_i = \frac{a_i}{\epsilon_i - \gamma},$$

where γ is the unique zero of g on the interval (ϵ_1, ϵ_2). This choice for $\mathbf{d}$ satisfies the condition $\mathbf{a}^\mathsf{T}\mathbf{d} = 0$ since $g(\gamma) = 0$. Since $\gamma < \epsilon_2$, we have

$$\begin{aligned}\mathbf{d}^\mathsf{T}(\mathbf{H} - \lambda\mathbf{I})\mathbf{d} &= \mathbf{d}^\mathsf{T}(\mathbf{H} - \epsilon_2\mathbf{I})\mathbf{d} = \sum_{i=1}^{n} d_i^2(\epsilon_i - \epsilon_2) \\ &= \sum_{i\notin\mathcal{E}_2} \frac{a_i^2(\epsilon_i - \epsilon_2)}{(\epsilon_i - \gamma)^2} < \sum_{i\notin\mathcal{E}_2} \frac{a_i^2(\epsilon_i - \gamma)}{(\epsilon_i - \gamma)^2} = g(\gamma) = 0.\end{aligned}$$

This violates the second-order condition (2.14).

(d) $\boldsymbol{a_i = 0}$ **for all** $\boldsymbol{i \in \mathcal{E}_2}$ **and** $\boldsymbol{g(\epsilon_2) = 0}$**.** This is the degenerate case introduced in Section 2.1. We first observe that $\lambda = \epsilon_2$. Suppose, to the contrary, that $\lambda < \epsilon_2$. By (2.21), $\mathbf{y}$ is given by (2.24). If $\mu \neq 0$, then the orthogonality condition gives (2.25), which has no solution on (ϵ_1, ϵ_2) since g is monotone and $g(\epsilon_2) = 0$. Consequently, $\mu = 0$ and (2.24) implies that $\mathbf{y} = \mathbf{0}$, violating the constraint $\mathbf{y}^\mathsf{T}\mathbf{y} = 1$. Thus $\lambda = \epsilon_2$.
By (2.21),

$$y_i = \frac{\mu a_i}{\epsilon_i - \epsilon_2} \quad \text{for } i \notin \mathcal{E}_2. \tag{2.28}$$

For $i \in \mathcal{E}_2$, the first-order condition (2.13) provides no information concerning y_i since both sides of the equation vanish identically :

$$(\epsilon_i - \epsilon_2)y_i = \mu a_i = 0$$

The general solution is the following. First, choose any value for y_i, $i \in \mathcal{E}_2$, such that

$$\sum_{i\in\mathcal{E}_2} y_i^2 \leq 1.$$

Then choose μ in (2.28) such that $\|\mathbf{y}\| = 1$. In other words, we choose μ so that

$$\mu^2 = \frac{1 - \sum_{i\in\mathcal{E}_2} y_i^2}{g'(\epsilon_2)}.$$

Notice that λ is unique in the degenerate case, while both μ and $\mathbf{y}$ have multiple values.

Lemma 2.3.1 *For the optimization problem* (2.9) *and a global minimizer* $\mathbf{y}$*, the multiplier* λ *associated with the constraint* $\mathbf{y}^\mathsf{T}\mathbf{y} = 1$ *is a Lipschitz continuous function of* $\mathbf{a}$ *on the unit sphere. With appropriate sign, the corresponding multiplier* μ *associated with the orthogonality constraint* $\mathbf{a}^\mathsf{T}\mathbf{y} = 0$ *is continuous at any nondegenerate* $\mathbf{a}$*.*

Proof. In Case 1, Lipschitz continuity is trivially satisfied, so we focus on the situation where $\epsilon_1 < \epsilon_2$. Since the intersection of the hyperplanes $a_1 = 0$ or $a_2 = 0$ with the unit sphere are sets of measure zero on the surface of the sphere, Lipschitz

continuity over the complement implies Lipschitz continuity over the entire sphere (by continuity). Hence, we restrict our attention to $\epsilon_1 < \epsilon_2$, $a_1 \neq 0$, and $a_2 \neq 0$. In this case, λ is the unique solution to (2.25) on the interval (ϵ_1, ϵ_2). Differentiating (2.25) gives

$$\frac{\partial \lambda}{\partial a_i} = \frac{2a_i}{(\lambda - \epsilon_i)g'(\lambda)}. \tag{2.29}$$

If for some i, we have

$$\sum_{j \in \mathcal{E}_i} a_j^2 \geq 1/2, \tag{2.30}$$

then

$$\begin{aligned} |\epsilon_i - \lambda| g'(\lambda) &= \left(\frac{1}{|\epsilon_i - \lambda|}\right)\left(\sum_{j \in \mathcal{E}_i} a_j^2\right) + |\epsilon_i - \lambda| \sum_{j \notin \mathcal{E}_i} \frac{a_j^2}{(\epsilon_j - \lambda)^2} \\ &\geq \left(\frac{1}{|\epsilon_i - \lambda|}\right) \sum_{j \in \mathcal{E}_i} a_j^2 \geq \frac{1}{2|\epsilon_i - \lambda|} \geq \frac{1}{2(\epsilon_n - \epsilon_1)}. \end{aligned} \tag{2.31}$$

It follows from (2.29) that when (2.30) holds,

$$\left|\frac{\partial \lambda}{\partial a_i}\right| \leq 4(\epsilon_n - \epsilon_1).$$

If $a_i = 0$, then $\partial\lambda/\partial a_i = 0$ by (2.29). Now, suppose that $a_i \neq 0$ and (2.30) is violated. By (2.29) and (2.31), we have

$$\begin{aligned} \left|\frac{\partial \lambda}{\partial a_i}\right| &= \frac{2|a_i|}{\dfrac{1}{|\epsilon_i - \lambda|}\left(\displaystyle\sum_{j \in \mathcal{E}_i} a_j^2\right) + |\epsilon_i - \lambda| \left(\displaystyle\sum_{j \notin \mathcal{E}_i} \frac{a_j^2}{(\epsilon_j - \lambda)^2}\right)} \\ &\leq \frac{2}{\left(\dfrac{|a_i|}{|\epsilon_i - \lambda|}\right) + \dfrac{|\epsilon_i - \lambda|}{|a_i|}\left(\displaystyle\sum_{j \notin \mathcal{E}_i} \frac{a_j^2}{(\epsilon_n - \epsilon_1)^2}\right)} \\ &\leq \frac{2}{\left(\dfrac{|a_i|}{|\epsilon_i - \lambda|}\right) + \left(\dfrac{|\epsilon_i - \lambda|}{|a_i|}\right)\left(\dfrac{1}{2(\epsilon_n - \epsilon_1)^2}\right)}. \end{aligned} \tag{2.32}$$

The last inequality is due to the assumption that (2.30) is violated, which implies that

$$\sum_{j \notin \mathcal{E}_i} a_j^2 \geq 1/2.$$

The denominator contains both $|a_i|/|\epsilon_i - \lambda|$ and its reciprocal $|\epsilon_i - \lambda|/|a_i|$. Suppose that

$$\frac{|a_i|}{|\epsilon_i - \lambda|} \geq 1. \tag{2.33}$$

By (2.32), the partial derivative $\partial\lambda/\partial a_i$ is bounded by 2 in magnitude since both terms in the denominator of (2.32) are positive and one of the terms is greater than or equal to 1. Conversely, if (2.33) is violated, then we drop the first term in the denominator of (2.32) to obtain

$$\left|\frac{\partial\lambda}{\partial a_i}\right| \leq 4(\epsilon_n - \epsilon_1)^2.$$

At this point, we have shown that there exists a constant β with the property that if $\mathbf{a}$ lies on the unit sphere with $a_1 \neq 0$ and $a_2 \neq 0$, then

$$\left|\frac{\partial\lambda}{\partial a_i}\right| \leq \beta$$

for each i. Observe that the solution λ to (2.25) does not change if $\mathbf{a}$ is multiplied by a nonzero scalar. Hence, for any nonzero $\mathbf{a}$ (not necessarily on the unit sphere) with $a_1 \neq 0$ and $a_2 \neq 0$, we have

$$\left|\frac{\partial\lambda}{\partial a_i}\right| \leq \beta/\|\mathbf{a}\|.$$

Consequently, there exists constants $r_1 < 1 < r_2$ with the property that

$$\left|\frac{\partial\lambda}{\partial a_i}\right| \leq 2\beta$$

whenever $r_1 \leq \|\mathbf{a}\| \leq r_2$, $a_1 \neq 0$, and $a_2 \neq 0$. Given two arbitrary points on the unit sphere, we can construct a piecewise linear path between them with the property that the line segments all lie within the shell formed by the spheres of radius r_1 and r_2. Due to the bound on the partial derivatives of λ, the change in λ across each line segment is bounded by 2β times the length of the line segment. Since the number of line segments is bounded, independent of the location of the points, we deduce that λ is a Lipschitz continuous function of $\mathbf{a}$ on the unit sphere.

Now consider the multiplier μ. If $\epsilon_1 = \epsilon_2$, then $\mu = 0$ (Case 1) and there is nothing to prove. Next, we focus on Case 3a where μ is given by (2.26). For $\lambda \in (\epsilon_1, \epsilon_2)$, g' is bounded away from zero. For example,

$$g'(\lambda) = \sum_{i=1}^{n} \frac{a_i^2}{(\epsilon_i - \lambda)^2} \geq \frac{1}{(\epsilon_n - \epsilon_1)^2}.$$

Let $\lambda(\mathbf{a})$ denote the unique multiplier associated with any given $\mathbf{a}$. Since λ is a Lipschitz continuous function of $\mathbf{a}$, it follows that μ is continuous at any point $\mathbf{a}$ where $\lambda(\mathbf{a}) \neq \epsilon_i$ for all i. Since $\lambda \in [\epsilon_1, \epsilon_2]$, the only potential points of discontinuity are those points $\mathbf{b}$ for which $\lambda(\mathbf{b}) = \epsilon_i$, $i = 1$ or $i \in \mathcal{E}_2$. If $b_1 \neq 0$ or $b_i \neq 0$ for any $i \in \mathcal{E}_2$, then $\mu(\mathbf{a})$ approaches 0 as $\mathbf{a}$ approaches $\mathbf{b}$ due to the pole in the denominator of g'. Hence, μ is continuous at $\mathbf{b}$.

If $b_1 = 0$ and $\lambda(\mathbf{b}) = \epsilon_1$, then we use (2.25) to solve for $a_1^2/(\epsilon_1 - \lambda)$:

$$\frac{a_1^2}{\lambda - \epsilon_1} = \sum_{i>1} \frac{a_i^2}{\epsilon_i - \lambda} \tag{2.34}$$

By assumption, $\lambda(\mathbf{a})$ approaches ϵ_1 as $\mathbf{a}$ approaches $\mathbf{b}$. Since the right side of (2.34) is continuous when λ is near ϵ_1, the limit of the left side as $\mathbf{a}$ approaches $\mathbf{b}$ is

$$\sum_{i>1} \frac{b_i^2}{\epsilon_i - \epsilon_1} > 0$$

since $b_1 = 0$ and $\|\mathbf{b}\| = 1$. Consequently, by (2.26), μ tends to 0 as $\mathbf{a}$ approaches $\mathbf{b}$, the same limit given in Case 2.

Finally, suppose that $b_i = 0$ for all $i \in \mathcal{E}_2$ and $\lambda(\mathbf{b}) = \epsilon_2$. Again, by (2.25), we have

$$\left(\frac{1}{\lambda - \epsilon_2}\right) \sum_{i \in \mathcal{E}_2} a_i^2 = \sum_{i \notin \mathcal{E}_2} \frac{a_i^2}{\epsilon_i - \lambda} \tag{2.35}$$

According to the statement of the lemma, we only need to prove continuity at nondegenerate $\mathbf{b}$, in which case the right side does not vanish at $\lambda = \epsilon_2$. Hence, as $\mathbf{a}$ approaches $\mathbf{b}$, the right side approaches the limit

$$\sum_{i \notin \mathcal{E}_2} \frac{b_i^2}{\epsilon_i - \epsilon_2} \neq 0.$$

Consequently, by (2.26), μ tends to 0 as $\mathbf{a}$ approaches $\mathbf{b}$, the same limit given in Case 3b. Note that Case 3c is not a point of discontinuity of μ since $\lambda < \epsilon_2$. This completes the proof. □

Now let us consider the original problem (2.1). If $\mathbf{H}_1$ and $\mathbf{H}_2$ commute, then they are simultaneously diagonalizable by the same eigenvector matrix [99, p. 249]. In this case, we can perform an orthogonal change of variables to reduce $\mathbf{H}_1$ and $\mathbf{H}_2$ to diagonal matrices. The solution is as follows :

Corollary 2 *Suppose* $\mathbf{H}_i$, $i = 1$ *and* 2, *are diagonal with diagonal element* ϵ_{ij}, $1 \leq j \leq n$, *arranged in increasing order. The minimum cost in* (2.1) *is*

$$\epsilon_{11} + \epsilon_{22} \quad \textit{or} \quad \epsilon_{12} + \epsilon_{21},$$

whichever is smaller. In the first case, an associated solution to (2.1) *is*

$$y_{11} = 1, \quad y_{22} = 1, \quad \textit{and} \quad y_{ij} = 0 \ \textit{otherwise.}$$

In the latter case, an associated solution to (2.1) *is*

$$y_{12} = 1, \quad y_{21} = 1, \quad \textit{and} \quad y_{ij} = 0 \ \textit{otherwise.}$$

Proof. Without loss of generality, we assume that the diagonal elements of $\mathbf{H}_i$ are strictly separated :

$$\epsilon_{i1} < \epsilon_{i2} < \ldots < \epsilon_{in}$$

for $i = 1, 2$. If some of the diagonal elements were equal, then we could establish the corollary by considering a sequence of perturbed problems and letting the perturbation go to zero. By the first-order optimality conditions (2.8) and by the diagonal structure of the $\mathbf{H}_i$, we have

$$(\epsilon_{1j} - \lambda_1)y_{1j} = \mu y_{2j} \quad \text{and} \quad (\epsilon_{2j} - \lambda_2)y_{2j} = \mu y_{1j}.$$

We combine these equations to obtain

$$\left[(\epsilon_{1j} - \lambda_1)(\epsilon_{2j} - \lambda_2) - \mu^2\right]y_{1j} = 0 = \left[(\epsilon_{1j} - \lambda_1)(\epsilon_{2j} - \lambda_2) - \mu^2\right]y_{2j}. \tag{2.36}$$

By Corollary 1, the multipliers λ_1 and λ_2 satisfy $\lambda_i \in [\epsilon_{i1}, \epsilon_{i2}]$. Hence, the coefficient of y_{1j} and y_{2j} in (2.36) are strictly increasing functions of $j \in [2, n]$. It follows that these coefficients can vanish for at most one $j \in [2, n]$ and possibly for $j = 1$. When the coefficients of y_{1j} and y_{2j} do not vanish in (2.36), we must have $y_{1j} = y_{2j} = 0$. In summary, at the global optimum, all the components of y_{ij} vanish except possibly y_{11}, y_{21}, y_{1j}, and y_{2j} for some $j \in [2, n]$. We focus on the case $j = 2$ since $j > 2$ leads to a larger cost.

Define $x_{1j}^2 = v_j$ and $x_{2j}^2 = w_j$ for $j = 1, 2$. The optimization problem (2.1) with $\mathbf{H}_i$ diagonal and $x_{ij} = 0$ for $j > 2$ reduces to

$$\begin{aligned} \min \quad & v_1\epsilon_{11} + v_2\epsilon_{12} + w_1\epsilon_{21} + w_2\epsilon_{22} \\ \text{subject to} \quad & v_1 + v_2 = 1 = w_1 + w_2, \\ & v_1 w_1 = v_2 w_2, \quad v_1, v_2, w_1, w_2 \geq 0. \end{aligned}$$

The equation $v_1 w_1 = v_2 w_2$ is the orthogonality condition $x_{11}x_{21} = -x_{12}x_{22}$ squared. We substitute $v_1 = 1 - v_2$ and $w_1 = 1 - w_2$ to reduce the optimization problem to

$$\begin{aligned} \min \quad & \epsilon_{11} + \epsilon_{21} + v_2(\epsilon_{12} - \epsilon_{11}) + w_2(\epsilon_{22} - \epsilon_{21}) \\ \text{subject to} \quad & v_2 + w_2 = 1, \quad v_2 \geq 0, \quad w_2 \geq 0. \end{aligned} \tag{2.37}$$

We substitute $v_2 = 1 - w_2$ in the objective function to further reduce the optimization problem to

$$\begin{aligned} \min \quad & \epsilon_{21} + \epsilon_{12} + w_2(\epsilon_{11} + \epsilon_{22} - \epsilon_{21} - \epsilon_{12}) \\ & \text{subject to} \quad 0 \leq w_2 \leq 1. \end{aligned}$$

Since the cost function is linear in w_2, the minimum is achieved at either $w_2 = 0$ ($w_1 = 1$, $v_1 = 0$, $v_2 = 1$) with objective function value $\epsilon_{21} + \epsilon_{12}$ or $w_1 = 1$ ($w_2 = 0$, $v_1 = 1$, $v_2 = 0$) with objective function value $\epsilon_{11} + \epsilon_{22}$. □

Remark 2. Assuming the eigenvalues are all distinct, then the degenerate choices for $\mathbf{a}$ in (2.9) correspond to those vectors $\mathbf{a}$ for which

$$\frac{a_1^2}{\epsilon_2 - \epsilon_1} = \sum_{i=3}^{n} \frac{a_i^2}{\epsilon_i - \epsilon_2}, \quad a_2 = 0, \quad \sum_{i \neq 2} a_i^2 = 1. \tag{2.38}$$

The solution to (2.1) given by Corollary 2 has the property that the nonzeros lie in the first two components of the vectors, while a degenerate $\mathbf{a}$ must have nonzero in components greater than or equal to 3. In fact, it follows from (2.38) that

$$\sum_{i=3}^{n} a_i^2 \geq \frac{\epsilon_3 - \epsilon_2}{\epsilon_3 - \epsilon_1}.$$

Remark 3. Let us consider the special case $\mathbf{H}_1 = \mathbf{H}_2 = \mathbf{H}$. Since $\mathbf{H}_1$ and $\mathbf{H}_2$ commute, we can apply Corollary 2. Let ϵ_j denote the j-th smallest eigenvalue of $\mathbf{H}$. As shown in (2.37), the optimization problem (2.1) reduces to

$$\begin{aligned} \min \quad & 2\epsilon_1 + (v_2 + w_2)(\epsilon_2 - \epsilon_1) \\ \text{subject to} \quad & v_2 + w_2 = 1, \quad v_2 \geq 0, \quad w_2 \geq 0. \end{aligned}$$

Since $v_2 + w_2 = 1$, the objective function value is $\epsilon_1 + \epsilon_2$, independent of the choice of v_2 and w_2 satisfying the constraints. Hence, when $\mathbf{H}_1 = \mathbf{H}_2$ there are an infinite number of solutions to (2.1). $\mathbf{y}_1$ is any unit vector in the span of the eigenvectors associated with ϵ_1 and ϵ_2, and $\mathbf{y}_2$ is any orthogonal unit vector in the same eigenspace. Note that if $\mathbf{H}$ is 2 by 2, then all feasible points are optimal and $F(\mathbf{x}_1, \mathbf{x}_2)$ is the trace of $\mathbf{H}$ whenever $\mathbf{x}_1$ and $\mathbf{x}_2$ are feasible in (2.1).

2.4 Convergence of the decomposition algorithm

The proof of Theorem 2.1.1 is organized into four steps. In Step 1, we analyze the global step and show that the iteration difference $\|\mathbf{x}_{k+1} - \mathbf{y}_k\|$ tends to 0. In Step 2, we show that the multipliers μ_{kj} in the local step are almost monotone decreasing since the violation in monotonicity decays to zero as the iteration number k tends to infinity. Step 3 and 4 focus on the limit of the multipliers μ_{kj} as k tends to infinity. Step 3 considers the limit 0, while Step 4 considers a positive limit.

Step 1. Analysis of the global step

Suppose that iteration k corresponds to the forward mode. Let $\mathbf{y}_k$ denote the result of the local step, and let $\mathbf{x}_{k+1}$ be the result of the global step based on the starting point $\mathbf{y}_k$. Since $\mathbf{x}_{k1}$ is feasible in the first subproblem of the forward mode (2.2), we have

$$F(\mathbf{y}_{k1}, \mathbf{x}_{k2}) \leq F(\mathbf{x}_{k1}, \mathbf{x}_{k2}).$$

Since $\mathbf{x}_{k2}$ is feasible in the second subproblem, we have

$$F(\mathbf{y}_{k1}, \mathbf{y}_{k2}) \leq F(\mathbf{y}_{k1}, \mathbf{x}_{k2}).$$

Combining these relations gives

$$F(\mathbf{y}_k) \leq F(\mathbf{x}_k). \tag{2.39}$$

A similar analysis for the reverse mode also gives $F(\mathbf{y}_k) \leq F(\mathbf{x}_k)$.

The components of $\mathbf{z}_k(s)$ lie on the unit sphere for all choices of s. Consequently, $F_k''(s)$ is bounded by a finite constant M, uniformly in k and s. Define the constants

$$\delta = \min\{\rho, 1/M\} \quad \text{and} \quad \bar{s}_k = -\delta F_k'(0).$$

Since $\bar{s}_k$ lies on the interval $[0, -\rho F_k'(0)]$ appearing in (2.6), we have

$$F(\mathbf{x}_{k+1}) = F_k(s_k) \leq F_k(\bar{s}_k). \tag{2.40}$$

Expanding in a Taylor series around $s = 0$, there exists $\xi_k \in [0, s_k]$ such that

$$\begin{aligned} F_k(\bar{s}_k) &= F_k(0) + F_k'(0)\bar{s}_k + \tfrac{1}{2}\bar{s}_k^2 F_k''(\xi_k) \\ &\leq F_k(0) + F_k'(0)\bar{s}_k + \tfrac{1}{2}\bar{s}_k^2 M \\ &= F_k(0) + \delta F_k'(0)^2(\tfrac{1}{2}\delta M - 1) \leq F_k(0) - \tfrac{\delta}{2}F_k'(0)^2 \\ &= F(\mathbf{y}_k) - \tfrac{\delta}{2}F_k'(0)^2 \leq F(\mathbf{x}_k) - \tfrac{\delta}{2}F_k'(0)^2, \end{aligned}$$

where the last inequality is (2.39). Combining this with (2.40) gives

$$F(\mathbf{x}_{k+1}) \leq F(\mathbf{x}_k) - \left(\frac{\delta}{2}\right) F_k'(0)^2.$$

Summing this inequality over k yields

$$F(\mathbf{x}_k) \leq F(\mathbf{x}_0) - \left(\frac{\delta}{2}\right) \sum_{i=0}^{k-1} F_i'(0)^2.$$

Since the feasible points for (2.1) lie on the unit sphere, the objective function value is bounded from below. Hence, we have

$$\lim_{k\to\infty} F_k'(0) = 0. \tag{2.41}$$

By the definition of $\mathbf{z}_k$ and the fact that $s_k \in [0, -\rho F_k'(0)]$ where $F_k'(0)$ approaches 0, we also conclude that

$$\|\mathbf{x}_{k+1} - \mathbf{y}_k\| \leq c|F_k'(0)| \tag{2.42}$$

where c is a constant which is independent of k.

Step 2. The change in the multiplier μ.

The first-order optimality conditions associated with the subproblems (2.2) can be stated in the following way : There exist scalars λ_{k1}, μ_{k1}, λ_{k2}, and μ_{k2} such that

$$\begin{aligned} &\text{forward} \quad \begin{bmatrix} \mathbf{H}_1 & \mathbf{0} \\ \mathbf{0} & \mathbf{H}_2 \end{bmatrix} \begin{bmatrix} \mathbf{y}_{k1} \\ \mathbf{y}_{k2} \end{bmatrix} = \begin{bmatrix} \lambda_{k1}\mathbf{y}_{k1} + \mu_{k1}\mathbf{x}_{k2} \\ \mu_{k2}\mathbf{y}_{k1} + \lambda_{k2}\mathbf{y}_{k2} \end{bmatrix}, \\ &\text{reverse} \quad \begin{bmatrix} \mathbf{H}_1 & \mathbf{0} \\ \mathbf{0} & \mathbf{H}_2 \end{bmatrix} \begin{bmatrix} \mathbf{y}_{k1} \\ \mathbf{y}_{k2} \end{bmatrix} = \begin{bmatrix} \lambda_{k1}\mathbf{y}_{k1} + \mu_{k1}\mathbf{y}_{k2} \\ \mu_{k2}\mathbf{x}_{k1} + \lambda_{k2}\mathbf{y}_{k2} \end{bmatrix}. \end{aligned} \tag{2.43}$$

By the orthogonality between $\mathbf{y}_{k1}$ and $\mathbf{x}_{k2}$ (forward), between $\mathbf{y}_{k1}$ and $\mathbf{y}_{k2}$ (forward and reverse), and between $\mathbf{x}_{k1}$ and $\mathbf{y}_{k2}$ (reverse), we have :

$$\text{forward}\begin{cases} \lambda_{k1} &= \mathbf{y}_{k1}^{\mathsf{T}}\mathbf{H}_1\mathbf{y}_{k1} \\ \mu_{k1} &= \mathbf{x}_{k2}^{\mathsf{T}}\mathbf{H}_1\mathbf{y}_{k1} \\ \lambda_{k2} &= \mathbf{y}_{k2}^{\mathsf{T}}\mathbf{H}_2\mathbf{y}_{k2} \\ \mu_{k2} &= \mathbf{y}_{k1}^{\mathsf{T}}\mathbf{H}_2\mathbf{y}_{k2} \end{cases} \qquad \text{reverse}\begin{cases} \lambda_{k1} &= \mathbf{y}_{k1}^{\mathsf{T}}\mathbf{H}_1\mathbf{y}_{k1} \\ \mu_{k1} &= \mathbf{y}_{k2}^{\mathsf{T}}\mathbf{H}_1\mathbf{y}_{k1} \\ \lambda_{k2} &= \mathbf{y}_{k2}^{\mathsf{T}}\mathbf{H}_2\mathbf{y}_{k2} \\ \mu_{k2} &= \mathbf{x}_{k1}^{\mathsf{T}}\mathbf{H}_2\mathbf{y}_{k2} \end{cases} \tag{2.44}$$

By our sign convention (2.7), the multipliers μ_{kj} are nonnegative.

By the definition of $F_k(s)$, we have

$$F_k'(0) = \pm(\mathbf{y}_{k1}^{\mathsf{T}}\mathbf{H}_1\mathbf{y}_{k2} - \mathbf{y}_{k2}^{\mathsf{T}}\mathbf{H}_2\mathbf{y}_{k1}). \tag{2.45}$$

We multiply the first equation in (2.43) by $\mathbf{y}_{k2}^{\mathsf{T}}$ to obtain $\mathbf{y}_{k1}^{\mathsf{T}}\mathbf{H}_1\mathbf{y}_{k2} = \mu_{k1}\mathbf{y}_{k2}^{\mathsf{T}}\mathbf{x}_{k2}$. We multiply the second equation by $\mathbf{y}_{k1}^{\mathsf{T}}$ to obtain $\mathbf{y}_{k1}^{\mathsf{T}}\mathbf{H}_2\mathbf{y}_{k2} = \mu_{k2}$. Hence, in the forward mode,

$$\begin{aligned} \mu_{k2} &= \mathbf{y}_{k1}^{\mathsf{T}}\mathbf{H}_2\mathbf{y}_{k2} \\ &= \mathbf{y}_{k1}^{\mathsf{T}}\mathbf{H}_1\mathbf{y}_{k2} \mp F_k'(0) \\ &= \mu_{k1}\mathbf{y}_{k2}^{\mathsf{T}}\mathbf{x}_{k2} \mp F_k'(0), \end{aligned} \tag{2.46}$$

which implies that

$$\mu_{k2} \le |\mu_{k1}\mathbf{y}_{k2}^{\mathsf{T}}\mathbf{x}_{k2}| + |F_k'(0)| \le \mu_{k1} + |F_k'(0)| \tag{2.47}$$

since $\mathbf{y}_{k2}$ and $\mathbf{x}_{k2}$ are unit vectors. In a similar fashion, for the reverse mode at iteration $k+1$, we have

$$\mu_{k+1,1} \le \mu_{k+1,2} + |F_{k+1}'(0)|. \tag{2.48}$$

If iteration k corresponds to the forward mode, then the multiplier μ_{k1} corresponds to $\mathbf{a} = \mathbf{x}_{k2}$ and $\mathbf{H} = \mathbf{H}_1$ in (2.9). The multiplier $\mu_{k-1,1}$ corresponds to $\mathbf{a} = \mathbf{y}_{k-1,2}$ and $\mathbf{H} = \mathbf{H}_1$ in (2.9). By (2.42) $\|\mathbf{x}_{k2} - \mathbf{y}_{k-1,2}\| \le c|F_{k-1}'(0)|$. We apply Lemma 2.3.1 and (2.41). For any (small) $\eta > 0$, we have

$$|\mu_{k1} - \mu_{k-1,1}| \le \eta$$

when k is sufficiently large, which implies that

$$\mu_{k1} \le \mu_{k-1,1} + \eta. \tag{2.49}$$

The analogous result for the reverse mode is

$$\mu_{k+1,2} \le \mu_{k2} + \eta \tag{2.50}$$

for k sufficiently large.

Combining (2.47)–(2.50), it follows that when k is large enough that $|F_j'(0)| \leq \eta$ for all $j \geq k$, we have

$$\mu_{k-1,1} \succeq \mu_{k1} \succeq \mu_{k2} \succeq \mu_{k+1,2} \succeq \mu_{k+1,1} \tag{2.51}$$

where the notation $\mu_{k1} \succeq \mu_{k2}$ means that $\mu_{k2} \leq \mu_{k1} + \eta$. Hence, in each iteration, the μ multiplier either decreases or makes an increase which is bounded by η.

Step 3. The case $\liminf \mu_{k1} = 0$.

When $\liminf \mu_{k1} = 0$, there exists a subsequence of the iterates with the property that μ_{k1} tends to 0. By (2.51) and the fact that η can be taken arbitrarily small, we conclude that the corresponding subsequence of the multipliers μ_{k2} also approaches 0. Since $\mathbf{y}_k$ lies in a compact set, we can extract subsequences converging to a limit denote $\mathbf{y}$. By (2.44), the corresponding subsequence of multipliers λ_{k1} and λ_{k2} also approach limits denoted λ_1 and λ_2. By (2.43), we have

$$\begin{bmatrix} \mathbf{H}_1 & \mathbf{0} \\ \mathbf{0} & \mathbf{H}_2 \end{bmatrix} \begin{bmatrix} \mathbf{y}_1 \\ \mathbf{y}_2 \end{bmatrix} = \begin{bmatrix} \lambda_1 \mathbf{y}_1 \\ \lambda_2 \mathbf{y}_2 \end{bmatrix}.$$

Since $\mathbf{y}_1$ and $\mathbf{y}_2$ are orthogonal unit vectors, we conclude that $\mathbf{y}$ is a stationary point for (2.1) corresponding to the multiplier $\mu = 0$.

Step 4. The case $\mu = \liminf \mu_{k1} > 0$.

Let us focus on a subsequence of the iterates, also denoted μ_{k1} for convenience, with the property that μ_{k1} approaches μ. Given any $\eta > 0$, choose K large enough that (2.51) holds for all $k \geq K$. Also, choose K large enough that

$$\left| \mu - \inf_{k \geq K} \mu_{k1} \right| \leq \frac{\eta}{2}. \tag{2.52}$$

Hence, for some $k \geq K$, we have

$$\mu - \eta \leq \mu_{k1} \leq \mu + \eta.$$

By (2.51) it follows that

$$\mu_{k2} \leq \mu_{k1} + \eta \leq \mu + 2\eta.$$

Also, by (2.51) and by (2.52), we have

$$\mu - \eta \leq \mu_{k+1,1} \leq \mu_{k+1,2} + \eta \leq \mu_{k2} + 2\eta.$$

Combining these inequalities gives

$$\mu - 3\eta \leq \mu_{k2} \leq \mu + 2\eta \quad \text{and} \quad \mu - \eta \leq \mu_{k1} \leq \mu + \eta.$$

Since η is arbitrary, it follows that μ_{k1} and μ_{k2} approach the same limit μ.

Again, by extracting subsequences, there exist limits $\mathbf{y}_1$, $\mathbf{y}_2$, λ_1, and λ_2 such that

$$\begin{bmatrix} \mathbf{H}_1 & \mathbf{0} \\ \mathbf{0} & \mathbf{H}_2 \end{bmatrix} \begin{bmatrix} \mathbf{y}_1 \\ \mathbf{y}_2 \end{bmatrix} = \begin{bmatrix} \lambda_1 \mathbf{y}_1 & + & \mu \mathbf{x}_2 \\ \mu \mathbf{y}_1 & + & \lambda_2 \mathbf{y}_2 \end{bmatrix}. \tag{2.53}$$

By (2.46), we have

$$\mu = \mu \mathbf{y}_2^\mathsf{T} \mathbf{x}_2,$$

where $\mu > 0$. Since $\mathbf{y}_2$ and $\mathbf{x}_2$ are unit vectors, we deduce that $\mathbf{x}_2 = \mathbf{y}_2$. When $\mathbf{x}_2$ is replaced by $\mathbf{y}_2$ in (2.53), we see that $\mathbf{y}$ is a stationary point.

Remark 4. In the special case $\mathbf{H}_1 = \mathbf{H}_2 = \mathbf{H}$, both the analysis and the algorithm simplify. As noted earlier, $F_k'(0) = 0$ in this case so the global step is skipped. Moreover, the monotonicity property (2.51) holds without the reverse iteration. Hence, the decomposition algorithm can simply employ forward steps, for which the associated first-order optimality condition is

$$\begin{bmatrix} \mathbf{H} & \mathbf{0} \\ \mathbf{0} & \mathbf{H} \end{bmatrix} \begin{bmatrix} \mathbf{y}_{k1} \\ \mathbf{y}_{k2} \end{bmatrix} = \begin{bmatrix} \lambda_{k1}\mathbf{y}_{k1} + \mu_{k1}\mathbf{y}_{k-1,2} \\ \mu_{k2}\mathbf{y}_{k1} + \lambda_{k2}\mathbf{y}_{k2} \end{bmatrix}. \tag{2.54}$$

Multiplying the first equation by $\mathbf{y}_{k2}^\mathsf{T}$ gives

$$\begin{aligned} \mu_{k1}\mathbf{y}_{k2}^\mathsf{T}\mathbf{y}_{k-1,2} &= \mathbf{y}_{k2}^\mathsf{T}\mathbf{H}\mathbf{y}_{k1} \\ &= \mathbf{y}_{k1}^\mathsf{T}(\mu_{k2}\mathbf{y}_{k1} + \lambda_{k2}\mathbf{y}_{k2}) = \mu_{k2}. \end{aligned}$$

Since $\mathbf{y}_{k2}$ and $\mathbf{y}_{k-1,2}$ are unit vectors, this shows that $\mu_{k2} \leq \mu_{k1}$. Multiplying the first equation in (2.54) by $\mathbf{y}_{k-1,2}$ gives

$$\begin{aligned} \mu_{k1} &= \mathbf{y}_{k-1,2}^\mathsf{T}\mathbf{H}\mathbf{y}_{k1} \\ &= \mathbf{y}_{k1}^\mathsf{T}(\mu_{k-1,2}\mathbf{y}_{k-1,1} + \lambda_{k-1,2}\mathbf{y}_{k-1,2}) \\ &= \mu_{k-1,2}\mathbf{y}_{k1}^\mathsf{T}\mathbf{y}_{k-1,1}. \end{aligned}$$

Since $\mathbf{y}_{k1}$ and $\mathbf{y}_{k-1,1}$ are unit vectors, we deduce that $\mu_{k1} \leq \mu_{k-1,2}$. Hence, (2.51) holds with $\succeq$ replaced by $\geq$.

2.5 Numerical experiments

A series of numerical experiments were performed to investigate the convergence rate of the decomposition algorithm and to explore the connections between the theoretical analysis and the practical convergence. The experiments we describe were performed using Scilab (www.scilab.org). The solution of each local step (2.9) was obtained by computing an eigenvector associated with the smallest non-zero eigenvalue of the matrix $\mathbf{PHP}$, where $\mathbf{P} = \mathbf{I} - \mathbf{a}\mathbf{a}^\mathsf{T}$ is the projection into the subspace perpendicular to $\mathbf{a}$. The global step was implemented using the Scilab routine "optim" with default parameter values.

Recall that in our theoretical analysis, the stepsize was restricted to an interval $[0, -\rho F_k'(0)]$, for some fixed $\rho > 0$, to ensure that the iterates approach each other in the limit. However, in all our numerical experiments, we found that there was no need to restrict the stepsize to obtain convergence. Hence, it appears that the

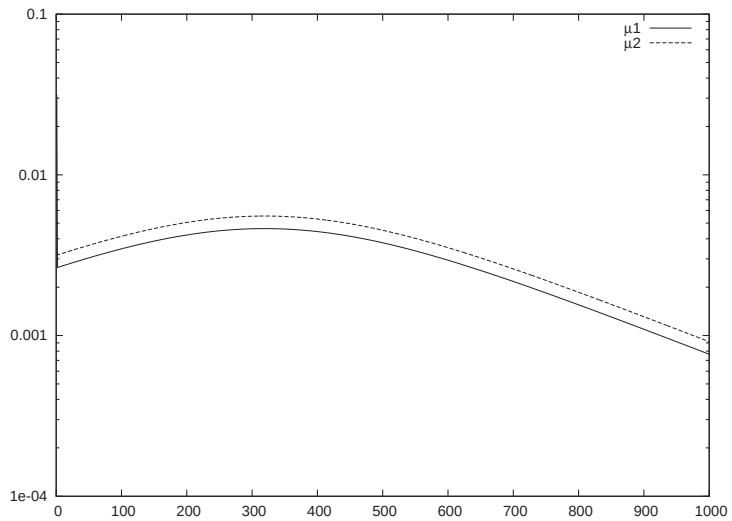


FIG. 2.2 – Convergence of the multipliers, random 100 by 100 diagonal matrices $\mathbf{H}_1$ and $\mathbf{H}_2$, $\rho = 1$.

restriction on the stepsize in (2.6) is an artifact of the analysis presented in this paper.

In (2.51) we show that the multipliers associated with the local steps in the decomposition algorithm almost decay monotonically. In Remark 4, we show that the decay is monotone when $\mathbf{H}_1 = \mathbf{H}_2$. Numerically, we found that when $\mathbf{H}_1 \neq \mathbf{H}_2$, the convergence of the multipliers may not be monotone. An illustration is given in Figure 2.2 where we randomly generate two 100 by 100 diagonal matrices $\mathbf{H}_1$ and $\mathbf{H}_2$ with entries between -1 and $+1$, and we plot the multipliers as a function of the iteration number. Due to the initial growth in the multipliers during the first 300 iterations, the convergence is not monotone and the inequalities $\succeq$ in (2.51) can not be replaced by $\geq$ in general.

In our experiments, the convergence speed when $\mathbf{H}_1 = \mathbf{H}_2$ was closely related to the distribution of the smallest 3 eigenvalues of the matrix, independent of the matrix dimension. To illustrate the typical convergence, we consider the 3 by 3 diagonal matrix

$$\mathbf{H}_1 = \mathbf{H}_2 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 2+\alpha \end{bmatrix}, \tag{2.55}$$

where $\alpha > 0$ is a parameter which we vary to explore the convergence. The starting guess is

$$\mathbf{x}_{02}^{\mathsf{T}} = \frac{1}{\sqrt{3}}[1 \;\; 1 \;\; 1]^{\mathsf{T}}. \tag{2.56}$$

Since $\mathbf{H}_1 = \mathbf{H}_2$, the reverse step can be skipped, and decomposition algorithm operates in "forward mode" without a global step. The components of the iterates

are always nonzero, and the iterates given by (2.26) can be expressed in the form

$$x_{k1i} = \frac{\mu_{k1} x_{k-1,2i}}{\epsilon_i - \lambda_{k1}}, \quad x_{k2i} = \frac{\mu_{k2} x_{k,1i}}{\epsilon_i - \lambda_{k2}}. \tag{2.57}$$

The iterates converge to the solution

$$\mathbf{x}_1^\mathsf{T} = [1 \;\; 0 \;\; 0]^\mathsf{T} \quad \text{and} \quad \mathbf{x}_2^\mathsf{T} = [0 \;\; 1 \;\; 0]^\mathsf{T}.$$

Since the third component of the solution vanishes, we will study how quickly the third component approaches 0. By (2.57), the ratio between the second and third components can be expressed as

$$\frac{x_{k13}}{x_{k12}} = \frac{x_{k-1,23}}{x_{k-1,22}} \left(\frac{\epsilon_2 - \lambda_{1k}}{\epsilon_3 - \lambda_{1k}} \right) = \frac{x_{k-1,13}}{x_{k-1,12}} \left(\frac{\epsilon_2 - \lambda_{2,k-1}}{\epsilon_3 - \lambda_{2,k-1}} \right) \left(\frac{\epsilon_2 - \lambda_{1k}}{\epsilon_3 - \lambda_{1k}} \right).$$

Since $\epsilon_2 > \epsilon_3$ for the matrix (2.55), the rational function $(\epsilon_2 - \lambda)/(\epsilon_3 - \lambda)$, $\lambda \in [1, 2]$, attains its maximum value $1/(1+\alpha)$ at $\lambda = 1$. Hence, by induction, we have

$$\frac{x_{k13}}{x_{k12}} \leq \left(\frac{1}{1+\alpha} \right)^{2k}. \tag{2.58}$$

Thus as α approaches 0, the bound on the rate at which the third component approaches 0, relative to the second component, grows, and the convergence could be much slower. And as α becomes large, the bound decreases and the convergence rate increases. In summary, the convergence speed seems to depend on the ratio of the gap between ϵ_2 and ϵ_3 relative to the gap between ϵ_1 and ϵ_2. As the ratio approaches 0, the convergence could be slower, as seen in (2.58).

In Figure 2.3 we show the convergence of $\mathbf{x}_{k13}$ as a function of the iteration number k for various choices of α. Notice that as α approaches 0, the optimization problem (2.1) becomes more poorly conditioned since the points

$$\mathbf{x}_1^\mathsf{T} = [1 \;\; 0 \;\; 0] \quad \text{and} \quad \mathbf{x}_2^\mathsf{T} = [0 \;\; 0 \;\; 1]$$

are feasible with objective function value $3 + \alpha \approx 3$, when $\alpha \approx 0$. Thus there are feasible points which are separated from the optimal solution, but with nearly the same cost as the optimal solution.

We now give an example where the decomposition algorithm does not converge to the global minimum when the starting guess is sufficiently poor. In Remark 1, we point out that the decomposition algorithm is convergent for any starting guess when $\mathbf{H}_1 = \mathbf{H}_2$ and $\epsilon_3 - \epsilon_2 \geq \epsilon_2 - \epsilon_1$. Suppose that $\epsilon_3 - \epsilon_2 < \epsilon_2 - \epsilon_1$, and the starting guess is

$$\mathbf{x}_{02}^\mathsf{T} = \frac{1}{\sqrt{2}} [1 \;\; 0 \;\; 1 \;\; 0 \;\; 0 \;\; \ldots \;]^\mathsf{T}.$$

According to case 3c of Section 2.3,

$$\mathbf{x}_{11}^\mathsf{T} = \frac{1}{\sqrt{2}} [-1 \;\; 0 \;\; 1 \;\; 0 \;\; 0 \;\; \ldots \;]^\mathsf{T} \quad \text{and} \quad \mathbf{x}_{12} = \mathbf{x}_{02}.$$

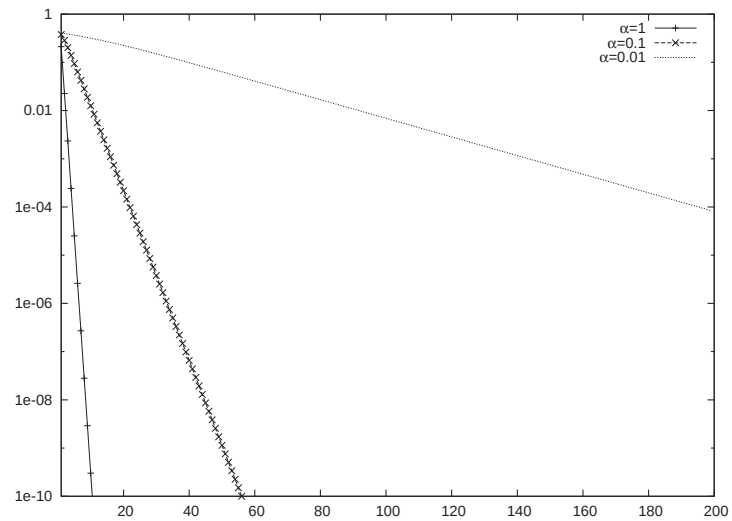


FIG. 2.3 – Convergence of x_{k13} for matrices (2.55) and starting point (2.56).

Hence, the algorithm converges after one iteration to the stationary point

$$\mathbf{x}_1^{\mathsf{T}} = \frac{1}{\sqrt{2}}[-1 \;\; 0 \;\; 1 \;\; 0 \;\; 0 \;\; \ldots \;]^{\mathsf{T}} \quad \text{and} \quad \mathbf{x}_2 = \frac{1}{\sqrt{2}}[1 \;\; 0 \;\; 1 \;\; 0 \;\; 0 \;\; \ldots \;]^{\mathsf{T}}.$$

This starting guess, however, is exceptional. If the second component of $\mathbf{x}_{02}$ is change to any nonzero value α, then the iterates quickly converge to the global minimum. For example, if $\alpha = 10^{-14}$ and

$$\mathbf{H}_1 = \mathbf{H}_2 = \text{diag}\,[-0.9, -0.5, -0.4, -0.3, -0.2, -0.1,\; 0.0, +0.1, +0.2, +0.3],$$

then the error is reduced to 10^{-10} within 44 iterations.

For the case $\mathbf{H}_1 \neq \mathbf{H}_2$, the decomposition algorithm may converge to a stationary point which is not the global minimum when the starting guess is degenerate and the degenerate iterates are chosen in a very special way. As an example, suppose that $\mathbf{H}_1$ and $\mathbf{H}_2$ are diagonal matrices, with diagonal elements arranged in increasing order, and that all the components of the starting point $\mathbf{x}_{02}$ are nonzero except for component 2 which is 0. The nonzero components of $\mathbf{x}_{02}$ are chosen to make it degenerate for $\mathbf{H}_1$. The initial iterate $\mathbf{y}_{11}$ is described by case 3d of Section 2.3. There are an infinite number of solutions to the local subproblem. We choose the solution for which the second component is zero (the remaining components are nonzero). Take ϵ_{11} close enough to ϵ_{12} to ensure that y_{111} is near 1 in magnitude. In this case, $g(\epsilon_{22}) < 0$ in the second local step. By case 3b of Section 2.3, all the components of the iterate $\mathbf{y}_{12}$ are zero except for the second component which is 1. Since $F_1'(0) = 0$, the global step has no effect ; we have $\mathbf{x}_{21} = \mathbf{y}_{11}$ and $\mathbf{x}_{22} = \mathbf{y}_{12}$. Thereafter, $\mathbf{x}_{k1}$ is the first column of the identity and $\mathbf{x}_{k2}$ is the second column of

the identity. If $\mathbf{H}_1$ and $\mathbf{H}_2$ are chosen so that

$$\epsilon_{11} + \epsilon_{22} > \epsilon_{21} + \epsilon_{12},$$

then the iteration has reached a stationary point which is not the global minimum. In contrast, with any perturbation in the second component of $\mathbf{x}_{02}$, we obtain convergence to the global minimum.

If the eigenvectors corresponding to the smallest eigenvalues of $\mathbf{H}_1$ and $\mathbf{H}_2$ are orthogonal, then these orthogonal eigenvectors are the solution of (2.1). By randomly choosing the remaining orthogonal eigenvectors of $\mathbf{H}_1$ and $\mathbf{H}_2$, we obtain noncommuting matrices for which the solution of (2.1) is known. As a specific example, we took $\mathbf{H}_i = \mathbf{Q}_i\mathbf{D}_i\mathbf{Q}_i^\mathsf{T}$ where $\mathbf{D}_i$ is a diagonal matrix with diagonal elements chosen randomly on $[-1, 1]$, and $\mathbf{Q}_i$, $i = 1$ or 2, is an orthogonal matrix of the form

$$\mathbf{Q}_1 = \begin{bmatrix} 1 & \mathbf{0}^\mathsf{T} \\ \mathbf{0} & \mathbf{U}_1 \end{bmatrix} \quad \text{and} \quad \mathbf{Q}_2 = \begin{bmatrix} 0 & \mathbf{0}^\mathsf{T} \\ \mathbf{e}_1 & \mathbf{U}_2 \end{bmatrix}.$$

Here $\mathbf{e}_1$ is the first column of the identity and $\mathbf{U}_i$ are random orthogonal matrices. For all starting points, we observed convergence to the global minimum. Convergence to local, non global, minima has also been observed in the case where the matrices $\mathbf{H}_1$ and $\mathbf{H}_2$ do not commute, but have the same ground state eigenvector.

2.6 Conclusions

A decomposition algorithm is developed for a quadratic programming problem with sphere and orthogonality constraints. The algorithm consists of local steps, both forward and reverse, and a global step where we minimize over a subspace. Without the global step, any limit $(\mathbf{y}_1, \mathbf{y}_2)$ of the local step satisfies

$$\begin{bmatrix} \mathbf{H}_1 & \mathbf{0} \\ \mathbf{0} & \mathbf{H}_2 \end{bmatrix} \begin{bmatrix} \mathbf{y}_1 \\ \mathbf{y}_2 \end{bmatrix} = \begin{bmatrix} \lambda_1\mathbf{I} & \mu_1\mathbf{I} \\ \mu_2\mathbf{I} & \lambda_2\mathbf{I} \end{bmatrix} \begin{bmatrix} \mathbf{y}_1 \\ \mathbf{y}_2 \end{bmatrix}.$$

This differs from the first-order optimality conditions (2.8) associated with the original optimization problem (2.1) because μ_1 may not equal μ_2. If the local step is followed by the global step, then according to the analysis of Section 2.4, $F_k'(0)$ tends to zero (see (2.41)), which implies that (see (2.45)) $\mathbf{y}_1^\mathsf{T}\mathbf{H}_1\mathbf{y}_2 = \mathbf{y}_2^\mathsf{T}\mathbf{H}_2\mathbf{y}_1$. Since $\mu_1 = \mathbf{y}_1^\mathsf{T}\mathbf{H}_1\mathbf{y}_2$ and $\mu_2 = \mathbf{y}_2^\mathsf{T}\mathbf{H}_2\mathbf{y}_1$, the global step ensures that $\mu_1 = \mu_2$. Consequently, the first-order optimality condition for (2.1) is satisfied.

The complexity of the analysis is connected with the proof of convergence. To show that the iterate $\mathbf{x}_{k2}$ converges to the same limit as $\mathbf{y}_{k2}$, we studied the properties of multipliers for the subproblem (2.9). We showed that the multiplier λ for the sphere constraint lies between ϵ_1 and ϵ_2, the two smallest eigenvalues of $\mathbf{H}$; moreover, λ depends Lipschitz continuously on $\mathbf{a}$. In contrast, the multiplier μ associated with the orthogonality constraint is continuous at *nondegenerate* choices for $\mathbf{a}$.

A less technical explanation for the performance of the decomposition algorithm is that the local steps steer the iterates into a low dimensional subspace associated with the eigenvectors of the smallest eigenvalues of $\mathbf{H}_1$ or $\mathbf{H}_2$, while the global step finds the best point in this subspace.

Chapitre 3

Domaines alignés : implémentation parallèle

Ce chapitre reproduit intégralement l'article [A1] rédigé en collaboration avec M. Barrault, E. Cancès, W. W. Hager et C. Le Bris.

On y introduit dans un premier temps (section 3.2) la méthode MDD, dans sa version "1D", ce qui signifie qu'on est soumis aux restrictions introduites dans [6] et [11] :

- les domaines sont alignés,
- les domaines ne se recouvrent qu'entre premiers voisins.

Concrètement, il n'est pas nécessaire que les atomes soient géométriquement alignés. Seuls les domaines doivent vérifier cette contrainte. Cette version "1D" peut donc être utilisée pour simuler des édifices tridimensionnels comme des nanotubes ou des molécules d'ADN.

Les éléments nouveaux par rapport à [11] sont :

- L'accélération de l'étape globale (section 3.3), grâce à une reformulation de la matrice hessienne de la fonctionnelle minimisée dans l'étape globale. Le détail de ces calculs est donné en Annexe, chapitre B.
- L'implémentation parallèle de MDD (section 3.4), avec des premiers résultats numériques obtenus sur un cluster de 16 processeurs (section 3.5.3).

Ce chapitre est prolongé par le chapitre suivant dans lequel :

- on complète les résultats de l'implémentation parallèle par ceux obtenus sur une machine *BlueGene/L* en utilisant jusqu'à 1024 processeurs,
- on revient sur la question de la vérification approchée de la contrainte d'orthogonalité, qui est simplement mentionnée dans la section 3.3, p. 50.

Enfin, les résultats de ce chapitre et du suivant ont fait l'objet des présentations orales [I1] à [I4].

Domain decomposition and electronic structure computations : a promising approach

G. Bencteux[1,4], M. Barrault[1], E. Cancès[2,4], W. W. Hager[3] and C. Le Bris[2,4]

[1]EDF R&D, 1 avenue du Général de Gaulle, 92141 Clamart Cedex, France
{guy.bencteux,maxime.barrault}@edf.fr

[2]CERMICS , École Nationale des Ponts et Chaussées, 6 & 8, avenue Blaise Pascal, Cité Descartes, 77455 Marne-La-Vallée Cedex 2, France,
{cances,lebris}@cermics.enpc.fr

[3]Department of Mathematics, University of Florida, Gainesville FL 32611-8105, USA, hager@math.ufl.edu

[4]INRIA Rocquencourt, MICMAC project, Domaine de Voluceau, B.P. 105, 78153 Le Chesnay Cedex, FRANCE

Abstract :

We describe a domain decomposition approach applied to the specific context of electronic structure calculations. The approach has been introduced in [11]. We survey here the computational context, and explain the peculiarities of the approach as compared to other problems of seemingly the same type of other engineering sciences. Improvements of the original approach presented in [11], including algorithmic refinements and effective parallel implementation, are included here. Test cases supporting the interest of the method are also reported.

It is our pleasure and an honor to dedicate this contribution to Olivier Pironneau, on the occasion of his sixtieth birthday. With admiration, respect and friendship.

3.1 Introduction and motivation

General context Numerical simulation is nowadays an ubiquituous tool in materials science, chemistry and biology. Design of new materials, irradiation induced damage, drug design, protein folding are instances of applications of numerical simulation. For convenience we now briefly present the context of the specific computational problem under consideration in the present article. A more detailed, mathematically-oriented, presentation is the purpose of the monograph [1] or of the review article [107].

For many problems of major interest, empirical models where atoms are represented as point particles interacting with a parameterized force-field are adequate models. On the other hand, when electronic structure plays a role in the phenomenon under consideration, an explicit quantum modelling of the electronic wavefunctions is required. For this purpose, two levels of approximation are possible.

The first category is the category of *ab initio* models, which are general purpose models that aim at solving sophisticated approximations of the Schrödinger equation. Such models only require the knowledge of universal constants and require a, ideally null but practically limited, number of adjustable parameters. The most commonly used models in this category are Density Functional Theory (DFT) based models and Hartree-Fock type models, respectively. Although these two families of models have different theoretical grounding, they share the same mathematical nature. They are *constrained minimization problems*, of the form

$$\inf \left\{ E(\psi_1, \ldots, \psi_N),\ \psi_i \in H^1(\mathbb{R}^3), \int_{\mathbf{R}^3} \psi_i \psi_j = \delta_{ij},\ \forall\, 1 \leq i, j \leq N \right\} \tag{3.1}$$

The functions ψ_i are called the *molecular orbitals* of the system. The energy functional E, which of course depends on the model employed, is parametrized by the charges and positions of the nuclei of the system under consideration. With such models, systems with up to 10^4 electrons can be simulated.

Minimization problems of the type (3.1) are not approached by minimization algorithms, mainly because they are high-dimensional in nature. In contrast, the numerical scheme consists in solving their Euler-Lagrange equations, which are nonlinear eigenvalue problems. The current practice is to iterate on the nonlinearity using fixed-point type algorithms, called in this framework *Self Consistent Field* iterations, with reference to the mean-field nature of DFT and HF type models.

The second category of models is that of semi-empirical models, such as Extended Hückel Theory based and tight-binding models, which contain additional approximations of the above DFT or HF type models. They consist in solving linear eigenvalue problems. State-of-the-art simulations using such models address systems with up to $10^5 - 10^6$ electrons.

Finite-difference schemes may be used to discretize the above problems. They have proved successful in some very specific niches, most of them related to solid-state science. However, in an overwhelming number of contexts, the discretization of

the nonlinear or linear eigenvalue problems introduced above is performed using a Galerkin basis $\{\chi_i\}_{1\leq i\leq N_b}$, with size $N_b > N$, the number of electrons in the system. Basis functions may be plane waves . This is often the case for solid state science applications and then N_b is very large as compared to N, typically one hundred times as large or more. They may also be *localized* functions, namely compactly supported functions or exponentially decreasing functions. Such basis sets correspond to the so-called *Linear Combination of Atomic Orbitals* (LCAO) approach. Then the dimension of the basis set needed to reach the extremely demanding accuracy required for electronic calculation problems is surprisingly small. Such basis sets, typically in the spirit of spectral methods, or modal synthesis, are, indeed, remarkably efficient. The domain decomposition method described in the present article is restricted to the LCAO approach. Indeed, it strongly exploits the locality of the basis functions.

In both categories of models, linear or nonlinear, the elementary brick is the solution to a (generalized) linear eigenvalue problem of the following form :

$$\begin{cases} Hc_i = \epsilon_i S c_i, & \epsilon_1 \leq \ldots \leq \epsilon_N \leq \epsilon_{N+1} \leq \ldots \leq \epsilon_{N_b}, \\ c_i^t S c_j = \delta_{ij}, \\ D_\star = \displaystyle\sum_{i=1}^{N} c_i c_i^t. \end{cases} \tag{3.2}$$

The matrix H is a $N_b \times N_b$ symmetric matrix, called the *Fock matrix*. When the linear system above is one iteration of a nonlinear cycle, this matrix is computed from the result of the previous iteration. The matrix S is a $N_b \times N_b$ symmetric positive definite matrix, called the *overlap* matrix, which depends only on the basis set used (it corresponds to the mass matrix in the language of finite element methods).

One searches for the solution of (3.2), that is the matrix $D_\star$ called the *density matrix*. This formally requires the knowledge of the first N (generalized) eigenelements of the matrix H (in fact we shall see below this statement is not exactly true).

The system of equations (3.2) is generally viewed as a generalized eigenvalue problem, and most of the computational approaches consist in solving the system via the computation of each individual vector c_i (discretizing the wavefunction ψ_i of (3.1)), using a direct diagonalization procedure.

Specificities of the approaches for large systems The procedure mentioned above may be conveniently implemented for systems of limited size. For large systems however, the solution procedure for the linear problem suffers from two computational bottlenecks. The first one is the need for assembling the Fock matrix. It *a priori* involves $O(N_b^3)$ operations in DFT models and $O(N_b^4)$ in HF models. Adequate approaches, which lower the complexity of this step, have been proposed. Fast multipole methods (see [68]) are one instance of such approaches. The second

practical bottleneck is the diagonalization step itself. This is the focus of the present contribution. Because of the possibly prohibitive $O(N_b^3)$ cost of direct diagonalization procedures, the so-called *alternatives to diagonalization* have been introduced. The method introduced in the present contribution aims at competing with such methods, and eventually outperforming them. With a view to understanding the problem under consideration, let us briefly review some peculiarities of electronic structure calculation problems.

The situation critically depends on the type of basis set employed. With plane wave basis sets, the number N of eigenelements to determine can be considered as small, compared to the size N_b of the matrix H ($N_b \sim 100N$). Then, iterative diagonalization methods, based on the inverse power paradigm, are a natural choice. In contrast, in the case of localized basis sets we deal with in this article, N_b varies from 2 to 10 times N. In any case it remains strictly proportional to N. Hence, problem (3.2) can be rephrased as follows : identify say one half of the eigenelements of a given matrix. This makes the problem very specific as compared to other linear eigenvalue problems encountered in other fields of the engineering sciences (see [100, 106] for instance). The sparsity of the matrices in the present context is another peculiarity of the problem. Although the matrices H and S are sparse for large molecular systems, they are not as sparse as the stiffness and mass matrices usually encountered when using finite difference or finite element methods. For example, the bandwith of H and S is of the order of 10^2 in the numerical examples reported in section 3.5.

Alternative methods towards linear scaling In addition to the above mentioned peculiarities, a crucial specificity of problem (3.2) is that the eigenelements indeed do not need to be explicitly identified. As expressed by the last line of (3.2), only the knowledge of the density matrix $D_\star$ is required, both for the evaluation of the Fock operator associated to the next iteration, in a nonlinear context, and for the evaluation of relevant output quantities, in the linear context or at the last step of the iteration loop.

From a geometrical viewpoint, $D_\star$ is the S-orthogonal projector (in the sense that $D_\star S D_\star = D_\star$ and $D_\star^t = D_\star$) on the vector subspace generated by the eigenvectors associated with the lowest N eigenvalues of the generalized eigenvalue problem $Hc = \epsilon Sc$.

The above elementary remark is the bottom line for the development of the alternative to diagonalization methods, also often called *linear scaling* methods because their claimed purpose is to reach a linear complexity of the solution procedure (either in terms of N the number of electrons, or N_b the dimension of the basis set). For practical reasons, which will not be further developed here, such methods assume that :

- (H1). The matrices H and S are sparse, in the sense that, for large systems, the number of non-zero coefficients scales as N. This assumption is not restrictive.

In particular, it is automatically satisfied for DFT and HF models as soon as the basis functions are localized ;

– (H2). The matrix $D_\star$ built from the solution to (3.2) is also sparse. This condition seems to be fulfilled as soon as the relative gap

$$\gamma = \frac{\epsilon_{N+1} - \epsilon_N}{\epsilon_{N_b} - \epsilon_1}. \tag{3.3}$$

deduced from the solution of (3.2) is large enough. This observation can be supported by qualitative physical arguments [45], but has seemingly no mathematical grounding to date (see, however, [44]).

State-of-the-art surveys on such methods are [15,36]. One of the most commonly used linear scaling method is the Density Matrix Minimisation method [47].

3.2 A new domain decomposition approach

Our purpose is now to expose a method, based on the *domain decomposition* paradigm, which we have recently introduced in [11], and for which we also consider a setting where the above two assumptions are valid. Although still in its development, we have good hope that this approach will outperform existing ones in a near future. Preliminary test cases support this hope.

The approach described below is not the first occurrence of a method based on a "geographical" decomposition of the matrix H in the context of quantum chemistry (see *e.g.* [80]). A significant methodological improvement is, however, fulfilled with the present method. To the best of our knowledge, existing methods in the context of electronic calculations that may be recast as domain decomposition methods only consist of local solvers complemented by a crude global step. Our method seems to be the first one really exhibiting the local/global paradigm in the spirit of methods used in other fields of the engineering sciences.

In the following, we expose and make use of the method on one-dimensional systems, typically nanotube or linear hydrocarbons. Generalizations to three-dimensional systems do not really bring up new methodological issues. They are, however, much more difficult in terms of implementation.

For simplicity, we now present our method assuming that $S = I_{N_b}$, i.e. that the Galerkin basis $\{\chi_i\}_{1\le i\le N_b}$ is orthonormal. The extension of the method to the case when $S \neq I_{N_b}$ is straightforward. The space $\mathcal{M}^{k,l}$ denotes the vector space of the $k \times l$ real matrices.

Let us first notice that a solution $D_\star$ of (3.2) reads

$$D_\star = C_\star C_\star^t \tag{3.4}$$

where $C_\star$ is a solution to the minimization problem

$$\inf\Big\{\mathrm{Tr}\Big(HCC^t\Big), \quad C \in \mathcal{M}^{N_b,N}(\mathbb{R}),\ C^tSC = I_N\Big\}. \tag{3.5}$$

Our approach consists in solving an *approximation* of problem (3.5). The latter is obtained by minimizing the exact energy $\mathrm{Tr}\left(HCC^t\right)$ on the set of the matrices C that have the block structure displayed on Figure 3.1 and satisfy the constraint $C^tC = I_N$.

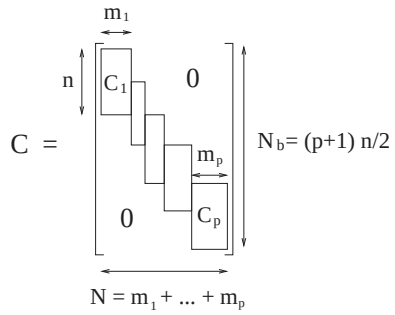


FIG. 3.1 – Block structure of the matrices C.

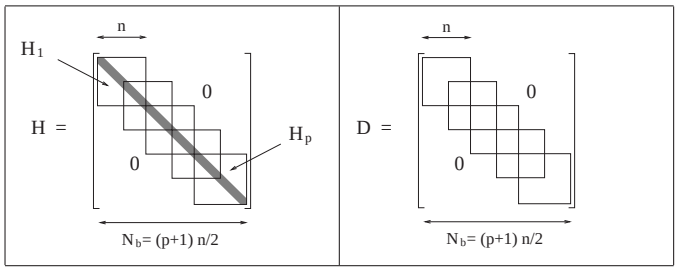


FIG. 3.2 – Block structure of the matrices H and D

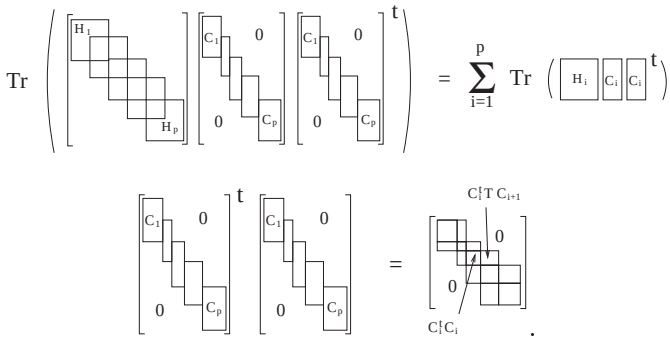
A detailed justification of the choice of this structure is given in [11]. Let us only mention here that the decomposition is suggested from the localization of electrons and the use of a localized basis set. Note that each block overlaps only with its first neighbors. Again for simplicity, we expose the method in the case where overlapping is exactly $n/2$, but it could be any integer smaller than $n/2$.

The resulting minimization problem can be recast as

$$\inf\left\{ \sum_{i=1}^{p} \mathrm{Tr}\left(H_iC_iC_i^t\right), \quad C_i \in \mathcal{M}^{n,m_i}(\mathbb{R}), \quad m_i \in \mathbb{N}, \quad C_i^tC_i = I_{m_i} \quad \forall\, 1 \le i \le p, \right.$$
$$\left. C_i^tTC_{i+1} = 0 \quad \forall\, 1 \le i \le p-1, \quad \sum_{i=1}^{p} m_i = N \right\}. \tag{3.6}$$

In the above formula, $T \in \mathcal{M}^{n,n}(\mathbb{R})$ is the matrix defined by

$$T_{kl} = \begin{cases} 1 & \text{if } k - l = \frac{n}{2} \\ 0 & \text{otherwise} \end{cases} \tag{3.7}$$



and $H_i \in \mathcal{M}^{n,n}(\mathbb{R})$ is a symmetric submatrix of H (see Figure 3.2), and

In this way, we replace the $\frac{N(N+1)}{2}$ *global* scalar constraints $C^tC = I_N$ involving vectors of size N_b, by the $\sum_{i=1}^{p} \frac{m_i(m_i+1)}{2}$ *local* scalar constraints $C_i^tC_i = I_{m_i}$ and the $\sum_{i=1}^{p-1} m_i m_{i+1}$ *local* scalar constraints $C_i^tTC_{i+1} = 0$, involving vectors of size n. We would like to emphasize that we can only obtain in this way a basis of the vector space generated by the lowest N eigenvectors of H. This is the very nature of the method, which consequently cannot be applied for the search for the eigenvectors themselves.

Before we describe in details the procedure employed to solve the Euler-Lagrange equations of (3.6) in a greater generality, let us consider, for pedagogic purpose, the following oversimplified problem :

$$\inf\left\{\langle H_1Z_1, Z_1\rangle + \langle H_2Z_2, Z_2\rangle, \quad Z_i \in \mathbb{R}^{N_b},\ \langle Z_i, Z_i\rangle = 1,\ \langle Z_1, Z_2\rangle = 0\right\}. \tag{3.8}$$

We have denoted by $\langle\cdot,\cdot\rangle$ the standard Euclidean scalar product on $\mathbb{R}^{N_b}$.

Problem (3.8) is not strictly speaking a particular occurence of (3.6), but it shows the same characteristics and technical difficulties : a separable functional is minimized, there are constraints on variables of each term and there is a cross constraint between the two terms.

The bottom line for our decomposition algorithm is to attack (3.8) as follows. Choose (Z_1^0, Z_2^0) satisfying the constraints and construct the sequence $(Z_1^k, Z_2^k)_{k\in\mathbb{N}}$ by the following iteration procedure. Assume (Z_1^k, Z_2^k) is known, then

– Local step. Solve

$$\begin{cases} \widetilde{Z}_1^k = \operatorname{arginf}\left\{\langle H_1Z_1, Z_1\rangle,\ Z_1 \in \mathbb{R}^{N_b},\ \langle Z_1, Z_1\rangle = 1\ \langle Z_1, Z_2^k\rangle = 0\right\}, \\ \widetilde{Z}_2^k = \operatorname{arginf}\left\{\langle H_2Z_2, Z_2\rangle,\ Z_2 \in \mathbb{R}^{N_b},\ \langle Z_2, Z_2\rangle = 1\ \langle \widetilde{Z}_1^k, Z_2\rangle = 0\right\}; \end{cases} \tag{3.9}$$

– Global step. Solve

$$\alpha^* = \operatorname{arginf}\left\{\langle H_1Z_1, Z_1\rangle + \langle H_2Z_2, Z_2\rangle,\ \alpha \in \mathbb{R}\right\} \tag{3.10}$$

where

$$Z_1 = \frac{\widetilde{Z}_1^k + \alpha\widetilde{Z}_2^k}{\sqrt{1+\alpha^2}},\ Z_2 = \frac{-\alpha\widetilde{Z}_1^k + \widetilde{Z}_2^k}{\sqrt{1+\alpha^2}}, \tag{3.11}$$

and set

$$Z_1^{k+1} = \frac{\widetilde{Z}_1^k + \alpha^* \widetilde{Z}_2^k}{\sqrt{1+(\alpha^*)^2}}, \quad Z_2^{k+1} = \frac{-\alpha^* \widetilde{Z}_1^k + \widetilde{Z}_2^k}{\sqrt{1+(\alpha^*)^2}}. \tag{3.12}$$

This algorithm operates at two levels : a fine level where two problems of dimension N_b are solved (rather than one problem of dimension $2N_b$) ; a coarse level where a problem of dimension 2 is solved.

The local step monotonically reduces the objective function; however, it may not converge to the global optimum. The technical problem is that the Lagrange multipliers associated with the constraint $\langle Z_1, Z_2 \rangle = 0$ may converge to different values in the two subproblems associated with the local step. The global step again reduces the value of the objective function since $\tilde{Z}_1^k$ and $\tilde{Z}_2^k$ are feasible in the global step. The combined algorithm (local step + global step), therefore, makes the objective function monotonically decrease. The simple case $H_1 = H_2$ is interesting to consider. First, if the algorithm is initialized with $Z_2^0 = 0$ in the first line of (3.9), it is easily seen that the local step is sufficient to converge to the global minimizer, in one single step. Second, it has been proved in [6] that for a more general initial guess and under some assumption on the eigenvalues of the matrix H_1, this algorithm globally converges to an optimal solution of (3.8). Ongoing work aims at generalizing the above proof when the additional assumption on eigenvalues is omitted. The analysis of the convergence in the case $H_1 \neq H_2$ is a longer term goal.

3.3 The Multilevel Domain Decomposition (MDD) algorithm

We define, for all p-tuple $(C_i)_{1 \leq i \leq p}$,

$$\mathcal{E}\Big((C_i)_{1 \leq i \leq p}\Big) = \sum_{i=1}^{p} \mathrm{Tr}\Big(H_i C_i C_i^t\Big), \tag{3.13}$$

and set by convention

$$U_0 = U_p = 0. \tag{3.14}$$

It has been shown in [11] that updating the block sizes m_i along the iterations is crucial to make the domain decomposition algorithm converge toward a good approximation of the solution to (3.5). It is, however, observed in practice that after a few iterations, the block sizes have converged (they do not vary in the course of the following iterations). This is why, for the sake of clarity, we have chosen to present here a simplified version of the algorithm where block sizes are held constant along the iterations. For a description of the complete algorithm with variable block sizes, we refer to [11].

At iteration k, we have at hand a set of matrices $(C_i^k)_{1 \leq i \leq p}$ such that $C_i^k \in \mathcal{M}^{n,m_i}(\mathbb{R})$, $[C_i^k]^t C_i^k = I_{m_i}$, $[C_i^k]^t T C_{i+1}^k = 0$. We now explain how to compute the new iterate $(C_i^{k+1})_{1 \leq i \leq p}$.

- **Step 1 : Local fine solver**.

(a) For each i, find :

$$\inf\Big\{ \mathrm{Tr}\left(H_i C_i C_i^t\right), \qquad C_i \in \mathcal{M}^{n,m_i}(\mathbb{R}), \quad C_i^t C_i = I_{m_i}, \\ [C_{i-1}^k]^t T C_i = 0, \quad C_i^t T C_{i+1}^k = 0 \Big\}. \tag{3.15}$$

This is done via diagonalization of the matrix H_i in the subspace

$$V_i^k = \left\{ x \in \mathbb{R}^n, \quad \left[C_{i-1}^k\right]^t T x = 0, \quad x^t T C_{i+1}^k = 0 \right\},$$

i.e. diagonalize $P_i^k H_i P_i^k$ where P_i^k is the orthogonal projector on V_i^k. This provides (at least) $n - m_{i-1} - m_{i-1}$ real eigenvalues and associated orthonormal vectors $x_{i,j}^k$. The latter are T-orthogonal to the column vectors of C_{i-1}^k and C_{i+1}^k.

(b) Collect the lowest m_i vectors $x_{i,j}^k$ in the $n \times m_i$ matrix $\widetilde{C}_i^k$.

- **Step 2 : global coarse solver**. Solve

$$\mathcal{U}^* = \mathrm{arginf}\Big\{ f(\mathcal{U}),\ \mathcal{U} = (U_i)_i,\ \forall 1 \le i \le p-1 \ \ U_i \in \mathcal{M}^{m_{i+1},m_i}(\mathbb{R}) \Big\}, \tag{3.16}$$

where

$$f(\mathcal{U}) = \mathcal{E}\left(\left(C_i(\mathcal{U}) \big(C_i(\mathcal{U})^t C_i(\mathcal{U}) \big)^{-\frac{1}{2}} \right)_i \right), \tag{3.17}$$

and

$$C_i(\mathcal{U}) = \widetilde{C}_i^k + T \widetilde{C}_{i+1}^k U_i \Big([\widetilde{C}_i^k]^t T T^t \widetilde{C}_i^k \Big) - T^t \widetilde{C}_{i-1}^k U_{i-1}^t \Big([\widetilde{C}_i^k]^t T^t T \widetilde{C}_i^k \Big). \tag{3.18}$$

Next set, for all $1 \le i \le p$,

$$C_i^{k+1} = C_i(\mathcal{U}^*) \ \left(C_i(\mathcal{U}^*)^t \, C_i(\mathcal{U}^*) \right)^{-1/2} . \tag{3.19}$$

Notice that in Step 1, the computations of each odd block is independant from the other odd blocks, and obviously the same for even blocks. Thus, we use here a red/black strategy.

In the global step, we perturb each variable by a linear combination of the adjacent variables. The matrices $\mathcal{U} = (U_i)_i$ in (3.16) play the same role as the real parameter α in the toy example, equation (3.10). The perturbation is designed so that the constraints are satisfied. However, our numerical experiments show that this is not exactly the case, in the sense that, for some i, $[C_i^{k+1}]^t [C_i^{k+1}]$ may present coefficients as large as about 10^{-3}. All linear scaling algorithms have difficulties in

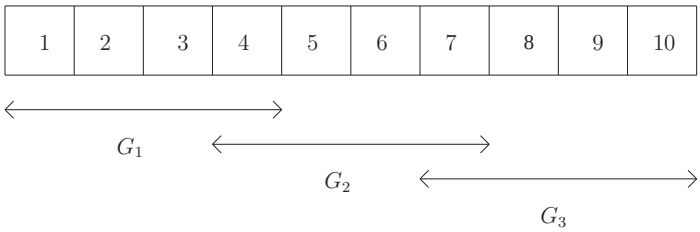


FIG. 3.3 – Collection of $p = 10$ blocks into $r = 3$ groups.

Repeat until convergence :

1a) Local step on blocks : 1, 3, ..., $(2i+1)$, ...
1b) Local step on blocks : 2, 4, ..., $(2i)$, ...
2a) Global step on groups : $\{1,2\}$, $\{3,4\}$, ..., $\{2i-1,2i\}$, ...
2b) Global step on groups : $\{2,3\}$, $\{4,5\}$, ..., $\{2i,2i+1\}$, ...

FIG. 3.4 – Schematic view of the algorithm in the case of 2-block groups ($r = 2$) : tasks appearing on the same line are independent from one another. Order between steps 1a and 1b is reversed from on iteration to the other. The same holds for steps 2a and 2b.

ensuring this constraint. We should mention here that in our case, the resulting deviation of C^tC from identity is small, C^tC being in any case block tridiagonal.

In practice, we reduce the computational cost of the global step, by again using a domain decomposition method. The blocks $(C_i)_{1\leq i\leq p}$ are collected in r overlapping groups $(G_l)_{1\leq l\leq r}$ as shown in Figure 3.3. As each group only overlaps with its first neighbors, the problem (3.16) can be solved first for the groups (G_{2l+1}), next for the groups (G_{2l}). We have observed that the number of iterations of the outer loop (local step + global step) does not significantly increases when the 'exact' global step (3.16) is replaced by the approximate global step consisting in optimizing first the odd groups, then the even groups. The numerical results performed so far (see section 3.5) tend to show that the resulting algorithm scales linearly with the system size.

A schematic view of the algorithm is provided in figure 3.4.

One important point (not taken into account in [11]) is that the Hessian of f enjoys a very specific structure. It is a sum of tensor products of square matrices of size m_i. For example, with two-block groups ($r = 2$), we have

$$HU = \sum_{i=1}^{4} A^{(i)} U B^{(i)} \tag{3.20}$$

with $A^{(i)} \in \mathcal{M}^{m_2,m_2}(\mathbb{R})$ and $B^{(i)} \in \mathcal{M}^{m_1,m_1}(\mathbb{R})$. Consequently, it is possible to com-

pute Hessian-vector products, without assembling the Hessian, in $\mathcal{O}(m_1\ m_2\ \max(m_1, m_2))$ elementary operations, instead of $\mathcal{O}(m_1^2\ m_2^2)$ with a naive implementation. An additional source of acceleration is the fact that this formulation uses only matrix-matrix products, taking advantage of higher numbers of floating point operations per memory access, are available in the BLAS 3 library (see for instance [118]). This makes Newton-like methods affordable : a good estimation of the Newton direction can be easily computed using an iterative method.

In the current version of our domain decomposition algorithm, the global step is solved approximatively by a single iteration of the Newton algorithm with initial guess $U_i = 0$, the Newton iteration being computed iteratively by means of the SYMMLQ algorithm [108]. In a next future, we plan to test the efficiency of advanced first order methods such as the one described in [105]. No definite conclusions about the comparative efficiencies of the various numerical methods for performing the global step can be drawn yet.

3.4 Parallel implementation

For parallel implementation, the single-program, multi-data (SPMD) model is used, with message passing techniques using the MPI library, which allows to maintain only one version of the code.

Each processor executes a single instance of the algorithm presented in section (3.3) applied to a contiguous subset of blocks. Compared to the sequential version, additional data structures are introduced : each processor needs to access the matrices C_i and F_i corresponding to the last block of the processor located on its left and to the first block of the processor located on its right, as shown in figure 3.5. These frontier blocks play the role of ghost nodes in classical domain decomposition without overlapping. For this reason, we will sometimes call them the ghost blocks.

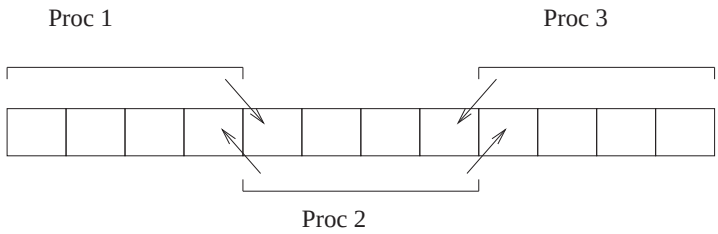


FIG. 3.5 – Distribution of blocks over 3 processors. Arrows indicate the supplementary blocks a processor needs to access

The major part of the communications is performed between neighboring processors at the end of each step of the algorithm (i.e. of each line in the scheme displayed on figure (3.4)), in order to update the ghost blocks. This occurs only four times per iteration and, as we will see in the next section, the size of the exchanged messages are moderate.

Collective communications are needed to compute the current value of the function f appearing in the formula (3.17) and to check that the maximum deviation from orthogonality remains acceptable. They are also needed to sort the eigenvalues of the different blocks in the local step, in the complete version of the algorithm, allowing variable block sizes (see [11]). The important point is that the amount of data involved in the collective communications if small as well.

With this implementation we can use up to $nbloc/2$ processors. In order to efficiently use a larger number of processors, sublevels of parallelism should be introduced. For instance, each subproblem (3.15) (for a given i) can itself be parallelized.

Apart from the very small part of collective communications, the communication volume associated with each single processor remains constant irrespective of the number of blocks per processor and the number of processors. We can thus expect a very good scalability, except for the situations when load balancing is strongly heterogeneous.

This implementation of the MDD algorithm can be easily extended to cover the case of $2D$ and $3D$ molecular systems.

3.5 Numerical tests

This section is devoted to the presentation of the performance of the Multilevel Domain Decomposition (MDD) algorithm on matrices actually arising in real-world applications of electronic structure calculations. The benchmark matrices are of the same type of those used in the reference paper [11].

In a first subsection, we briefly recall how these matrices are generated and we provide some practical details on our implementation of the MDD algorithm. The computational performances obtained on sequential and parallel architectures, including comparisons with the density matrix minimization (DMM) method and with direct diagonalization using LAPACK, are discussed in the second and third subsections, respectively.

3.5.1 General presentation

Three families of matrices corresponding to the Hartree-Fock ground state of some polymeric molecules are considered :

- Matrices of type $\mathcal{P}_1$ and $\mathcal{P}_2$ are related to COH-$(CO)_{n_m}$-COH polymeric chains with interatomic Carbon-Carbon distances equal to 5 atomic units and 4 atomic units (a.u.), respectively ;
- Matrices of type $\mathcal{P}_1$ are obtained with polyethylen molecules (CH_3-$(CH_2)_{n_m}$-CH_3) with physically relevant Carbon-Carbon distances.

The geometry of the very long molecules is guessed from the optimal distances obtained by geometry optimization (with constraints for $\mathcal{P}_1$ and $\mathcal{P}_2$) on moderate size system (about 60 carbon atoms) and minimal basis sets. All these off-line calcula-

	$\mathcal{P}_1$	$\mathcal{P}_2$	$\mathcal{P}_3$
Bandwith of S	59	79	111
Bandwith of H	99	159	255
n	130	200	308
q	50	80	126
Asymptotic gap (a.u.)	1.04×10^{-3}	3.57×10^{-3}	2.81×10^{-2}

TAB. 3.1 – Localization parameters, block sizes and asymptotic gaps for the test cases.

tions are performed using the GAUSSIAN package [29]. It is then observed that the overlap matrix and the Fock matrix obtained exhibit a periodic structure in their bulk. Overlap and Fock matrices for large size molecules can then be constructed using this periodicity property. For n_m sufficiently large, bulk periodicity is also observed in the density matrix. This property is used to generate reference solution for large molecules.

Table 3.1 gives a synthetic view of the different structure properties of the three families of matrices under examination. The integer q stands for the overlap between two adjacent blocks (note that one could have taken $n = 2q$ if the overlap matrix was equal to identity, but that one has to take $n > 2q$ in our cas since $S \neq I$).

Initial guess generation is of crucial importance for any linear scaling method. Procedure in use here is in the spirit of the domain decomposition method :

- A first guess of the block sizes is obtained by locating Z electrons around each nucleus of charge Z ;
- A set of blocks C_i is built from the lowest m_i (generalized) eigenvectors associated with the block matrices H_i and S_i (the block matrices H_i are introduced in Section 3.2 ; the block matrices S_i are defined accordingly) ;
- These blocks are eventually optimized with the local fine solver of the MDD algorithm, including block size update (electron transfer).

Criteria for comparing the results The quality of the results produced by the MDD and DMM methods is evaluated by computing two criteria. The first criterion is the relative energy difference $e_E = \dfrac{|E - E_0|}{|E_0|}$ between the energy E of the current iterate D and the energy E_0 of the reference density matrix $D_\star$. The second criterion is the semi-norm

$$e_\infty = \sup_{(i,j)\,\text{s.t.}\,|H_{ij}|\geq\varepsilon} \left| D_{ij} - [D_\star]_{ij} \right|, \tag{3.21}$$

with $\varepsilon = 10^{-10}$. The introduction of the semi-norm (3.21) is consistent with the cut-off on the entries of H (thus the value chosen for ε). Indeed, in most cases, the matrix D is only used for the calculations of various observables (in particular the electronic energy and the Hellman-Feynman forces), all of them of the form $\mathrm{Tr}(AD)$,

where the symmetric matrix A shares the same pattern as H (see [1] for details). The final result of the calculation is, therefore, insensitive to entries $D_{i,j}$ with indices (i, j) such that $|H_{ij}|$ is below some cut-off value.

In all the calculations presented below, global step is performed with groups consisting of two blocks ($r = 2$), and the algorithm is, therefore, exactly that displayed in figure 3.4.

3.5.2 Sequential computations

The numerical results presented in that section have been obtained with a single 2.8 GHz Xeon processor.

Density matrices have been computed for a series of matrices H of types $\mathcal{P}_1$, $\mathcal{P}_2$, $\mathcal{P}_3$, using (1) the MDD algorithm, (2) a diagonalization procedure (the *dsbgv.f* routine from the LAPACK library), and (3) the DMM method [47]. The latter method belongs to the class of linear scaling algorithms. An important feature of the DMM method is that linear scaling is achieved through cut-offs on the matrix entries. We have chose here a cut-off strategy based on *a priori* defined patterns, that may be suboptimal. Our implementation of DMM converges to a fairly good approximation of the exact density matrix and scales linearly, but the prefactor might possibly be improved by more refined cut-off strategies.

A detailed presentation of the comparison between the three methods is provided in [11]. Our new approach for computing the Newton direction in the global step (see section 3.3) further improves the efficiency of MDD : with the new implementation of MDD, and with respect to the former implementation reported on in [11], CPU time is divided by 2 for $\mathcal{P}_1$ type molecules, by 5 for $\mathcal{P}_2$, by 10 for $\mathcal{P}_3$, and the memory required is now lower for MDD than for DMM. These results are shown for $\mathcal{P}_2$ in figure 3.6 and 3.7. They clearly demonstrate that the MDD algorithm scales linearly with respect to the parameter n_m (in both CPU time and memory occupancy).

Let us notice that for ($\mathcal{P}_2$), the cross over point between diagonalization and MDD (as far as CPU time is concerned) is now shifted to less than 2,000 basis functions.

3.5.3 Parallel computations

We conclude with some tests of our parallel implementation of the MDD algorithm described in section 3.4. These tests have been performed on a 8 node Linux cluster in dedicated mode, consisting of 8 biprocessors DELL Precision 450 (Intel(R) Xeon(TM) CPU 2.40GHz), with Gigabit Ethernet connections. They concern the polyethylen family $\mathcal{P}_3$, for which the size of each ghost block is about 150 Ko.

We only test here the highest level of parallelism of the MDD algorithm, consistently with the relatively low number of processors that have been used in this first study. We plan to test multilevel parallelism in a near future. In particular, the local step in each block, as well as the global step in each group, will be parallelized.

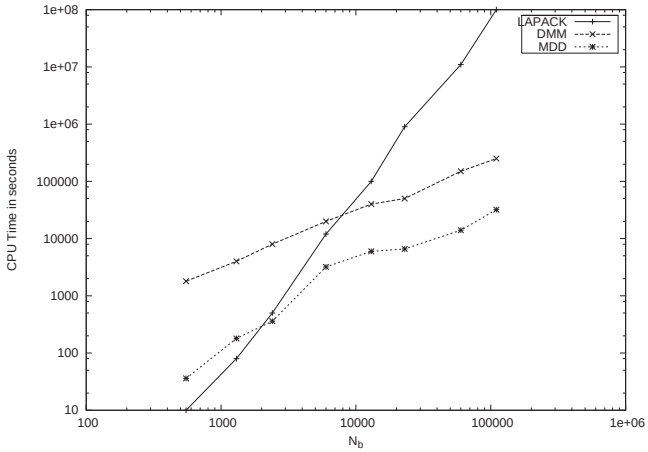


FIG. 3.6 – Requested CPU time for computing the density matrix of a molecule of type $\mathcal{P}_2$ as a function of the number of basis functions.

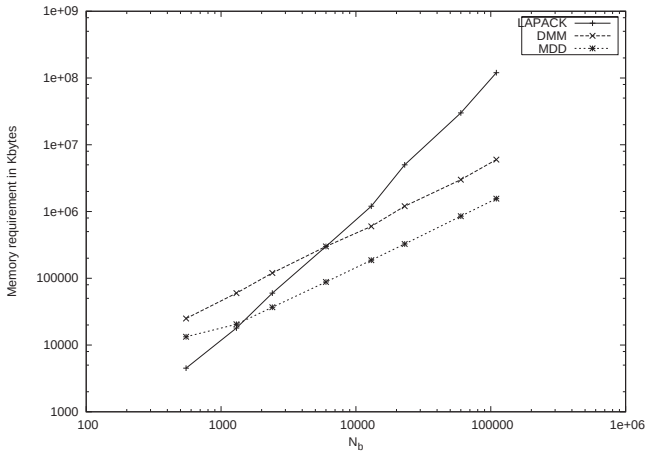


FIG. 3.7 – Requested memory for computing the density matrix of a molecule of type $\mathcal{P}_2$ as a function of the number of basis functions.

Tables 3.2 and 3.3 report on the speedup (ratio between the wall clock time with one processor and the wall clock time for several processors) and efficiency (ratio between the speedup and the number of processors) of our parallel MDD algorithm.

The scalability, namely the variation of the wall clock time when the number of processors and the size of the matrix proportionally grow, is reported in table 3.4, for a molecule of type $\mathcal{P}_3$.

Note that the calculations reported in this article have been performed with minimal basis sets. It is the subject of ongoing works to test the efficiency of the MDD algorithm for larger basis sets.

Let us finally mention that our parallel implementation of the MDD algorithm allows to solve (3.2) for a polyethylene molecule with 106530 atoms (372862 basis functions) on 16 processors, in 90 minutes.

Number of processors	1	2	4	8	16
CPU(s)	4300	2400	1200	580	360
Speedup		1.8	3.6	7.4	12
Efficiency		0.9	0.9	0.9	0.75

TAB. 3.2 – Wall-clock time as a function of the number of processors for a molecule of type $\mathcal{P}_3$ with $n_m = 3300$ (128 blocks). 8 MDD iterations are necessary to achieve convergence up to 5×10^{-8} in energy and 3×10^{-3} in the density matrix (for the semi-norm (3.21)).

Number of processors	1	4	8	16
CPU(s)	18460	4820	2520	1275
Speedup		3.8	7.3	14.5
Efficiency		0.96	0.92	0.91

TAB. 3.3 – Wall-clock time as a function of the number of processors for a molecule of type $\mathcal{P}_3$ with $n_m = 13300$ (512 blocks). 7 MDD iterations are necessary to achieve convergence up to 5×10^{-8} in energy and 3×10^{-3} in the density matrix (for the semi-norm (3.21)).

Number of processors	1	4	8	16
CPU(s) with 200 atoms per processors	167	206	222	253
CPU(s) with 800 atoms per processors	1249	1237	1257	1250

TAB. 3.4 – Scalability of the MDD algorithm for a molecule of type $\mathcal{P}_3$. The convergence thresholds are 2.5×10^{-7} in energy and $4\ 10^{-3}$ in density matrix (for the semi-norm 3.21)).

3.6 Conclusions and perspectives

In its current implementation, the MDD algorithm allows to solve efficiently the linear subproblem for linear molecules (polymers or nanotubes). The following issues will be addressed in a near future :

- Still in the case of 1D systems, we will allow blocks to have more than two neighbors. This should increase the flexibility and efficiency of the MDD algorithm. For instance, this should render calculations with large basis sets including diffuse atomic orbitals affordable.
- We plan to implement the MDD algorithm in the framework of 2D and 3D molecular systems. Note that even with minimal overlap a given block has typically 8 neighbors in 2D and 26 neighbors in 3D.
- The MDD algorithm will be extended to the cases of the nonlinear Hartree-Fock and Kohn-Sham problems.
- The present version of the MDD algorithm is retricted to insulators (i.e. to matrices H with a sufficiently large gap). The possibility of extending the MDD methodology to cover the case of metallic systems is a challenging issue that will be studied.

Chapitre 4

Domaines alignés : comportement numérique

Ce chapitre est la suite naturelle du précédent car on reste dans le cadre de la version "1D" de l'algorithme. Il est divisé en trois parties relativement indépendantes les unes des autres.

La première partie décrit les résultats de l'implémentation parallèle sur une machine *BlueGene/L*, en utilisant jusqu'à 1024 processeurs. Ces résultats ont été publiés dans [A2].

Dans la suivante, section 4.2, on décrit l'importance de la relaxation de la contrainte d'orthogonalité entre les blocs, en relation avec la pertinence de la localisation choisie d'une part, et avec la convergence de MDD d'autre part.

Enfin, on conclut ce chapitre par une section présentant l'effet cumulé des améliorations obtenues par rapport à [11], ainsi qu'une comparaison des performances de la version "1D" de MDD et des méthodes directes de diagonalisation les plus courantes, pour plusieurs types de molécules.

4.1 Calcul distribué sur un grand nombre de processeurs

Cette section complète le paragraphe 3.5.3 par des résultats obtenus sur beaucoup plus de processeurs. Dans un premier temps, on présente le calculateur qui a servi pour ces simulations, puis on analyse les résultats obtenus.

Le calculateur utilisé, *Frontier*, est un modèle *BlueGene/L* (BG/L) construit par IBM et acquis par EDF en 2006. On en donne ici les grandes caractéristiques. Pour une description plus détaillée des systèmes BG/L (réseaux, processeurs), on pourra consulter [113] et les références y figurant. Les machines BG/L sont composées de nœuds de calcul[1] interconnectés par plusieurs réseaux de communication très performants :

- un réseau torique pour les communications point à point,
- un réseau en arbre pour les communications globales.

Les bandes passantes annoncées par le constructeur pour ces réseaux sont respectivement de 175 MB/s et de 350 MB/s. La latence (temps nécessaire pour initier chaque communication) est très faible (3.3μs).

Chaque nœud est muni de deux processeurs de puissance modeste (700 MHz) et dispose de seulement 512 Ko de mémoire. Par défaut, un noeud ne gère qu'un seul processus de calcul (mode *Coprocessor*), qui est traité par l'un des deux processeurs, tandis que l'autre sert à accélérer les communications. On peut également utiliser les deux processeurs pour le calcul (mode *Virtual Node*) et avoir ainsi deux processus de calcul par noeud. En plus d'induire un possible ralentissement des communications, il faut dans ce cas avoir des processus utilisant peu de mémoire vive, puisque les deux processus doivent se partager les 512 Ko du nœud.

Pour résumer, cette machine dispose d'une grande puissance de calcul[2] obtenue par le biais d'un très grand nombre de processeurs relativement lents, reliés par un réseau très performant. Seuls les codes ayant une bonne scalabilité pour un grand nombre de processeurs sont en mesure d'exploiter pleinement cette puissance de calcul.

On présente maintenant les performances de MDD en se basant, comme dans le paragraphe 3.5.3, sur des simulations numériques de molécules de polyéthylène (famille $\mathcal{P}_3$, molécules de type CH_3-$(CH_2)_{n_m}$-CH_3) dont on fait varier la taille, n_m, par périodisation. La localisation est inchangée pour toutes les simulations :

- le nombre de fonctions de bases par domaines, n, est égal à 308,
- le nombre de fonctions communes entre deux domaines voisins, q, vaut 126.

On étudie donc ici simplement la scalabilité de l'implémentation informatique de MDD, pas celle de la méthode de décomposition de domaine.

La scalabilité est mise en évidence par deux tests complémentaires. Dans un premier temps, on cherche à résoudre un problème de taille donnée en un minimum

[1] 4096 sur le modèle acquis par EDF.

[2] Les machines de ce type sont parmi les plus puissantes du monde depuis plusieurs années [121].

de temps par l'utilisation d'un nombre croissant de processeurs. On appelle *speed-up* l'accélération obtenue par rapport à l'algorithme le plus rapide pour résoudre le même problème sur un unique processeur. Dans notre cas, on considère que MDD est l'algorithme le plus rapide sur un unique processeur pour résoudre le problème approché (3.6). De plus, on ne prend pas l'exécution séquentielle comme temps de référence, car un unique processeur de *Frontier* ne dispose pas d'une quantité de mémoire suffisante. Les tableaux 4.1, 4.2 et la figure 4.2 montrent les résultats obtenus pour une série de problèmes. Pour ces tests, la bibliothèque mathématique ESSL a été utilisée pour toutes les opérations algébriques intervenant dans MDD.

Dans la version "1D" de MDD, le nombre maximal de processeurs utilisables est égal au nombre de domaines[3]. Le nombre de processeurs a été augmenté jusqu'à atteindre 8 domaines par processeurs et on observe des accélérations quasiment idéales.

Nombre de processeurs	16	32	64	128
Temps de calcul (s)	1892	968	516	260
Speed-up	1	2.0	3.7	7.3

TAB. 4.1 – Evolution du temps de calcul pour $n_m \approx 1.3 \times 10^4$ en fonction du nombre de processeurs. Toutes les simulations ont été faites en mode *Virtual node*, donc en occupant deux fois moins de nœuds que de processeurs. La simulation à 128 processeurs correspond à 8 domaines par processeurs.

Nombre de processeurs	16	32	64	128	256
Temps de calcul (s)	5613	2855	1506	735	400
Speed-up	1	2.0	3.7	7.6	14

TAB. 4.2 – Evolution du temps de calcul pour $n_m = 4 \times 10^4$ en fonction du nombre de processeurs. Seule la simulation à 16 processeurs a été faite en mode *Coprocessor*. Toutes les autres simulations ont été faites en mode *Virtual node*, donc en occupant deux fois moins de nœuds que de processeurs. La simulation à 128 processeurs correspond à 12 domaines par processeurs.

Une autre approche pour l'étude de la scalabilité vise à répondre à la question : pour un type de processeur et un réseau de communication fixés, pour un temps de calcul donné, quelle est la taille maximale du problème que l'on peut résoudre ? Le test consiste alors à augmenter simultanément la taille du problème à résoudre et le nombre de processeurs. Le code est scalable[4] si le temps de calcul ne croît

[3] Le temps de résolution de chaque domaine étant relativement faible, il aurait été inutile d'introduire un parallélisme de deuxième niveau.

[4] On parle alors de *scalabilité faible*.

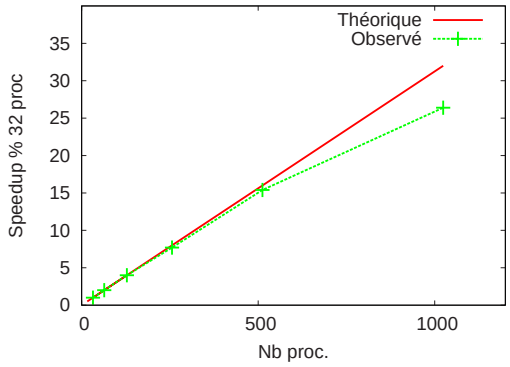


FIG. 4.1 – Evolution du *speed-up* entre 32 et 1024 processeurs pour un calcul à $n_m \approx 10^5$.

pas fortement avec la taille du problème. On observe, figure 4.2, que le temps de calcul ne varie quasiment pas quand on résout un problème de plus en plus grand pour une charge de calcul constante par processeur. Des résultats analogues ont été obtenus pour des matrices correspondant à d'autres types de molécules. L'implémentation "1D" de MDD présente donc une excellente scalabilité relativement au nombre d'atomes représenté par la matrice F, c'est-à-dire dans le cadre d'un calcul de type semi-empirique. Dans le cadre des calculs *ab initio*, on aurait pu également s'intéresser à la scalabilité du code en termes de précision du calcul, c'est-à dire relativement à l'augmentation de la taille de la base de discrétisation. Ce type de tests numériques sera réalisé après la thèse.

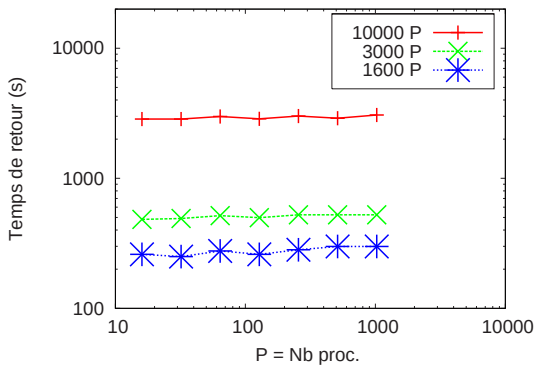


FIG. 4.2 – Evolution du temps de calcul quand on augmente simultanément la taille du monomère n_m et le nombre de processeurs utilisés P. La légende indique la taille des matrices H, S et D en fonction du nombre de processeurs. Le plus gros calcul met en jeu des matrices de taille 10^7 environ.

Les tailles des simulations réalisées ici ne sont pas du même ordre que celles présentées au paragraphe 3.5.3 à cause des différences entre les architectures : on a peu de processeurs ayant beaucoup de mémoire sur l'une, on a beaucoup de processeurs ayant peu de mémoire sur l'autre. On peut néanmoins comparer les temps

de calcul pour un même nombre d'atomes par processeur : le test à 200 atomes par processeurs dans le tableau 3.4 correspond à 1400 fonctions de base par processeur. Les temps obtenus (autour de 200 s) sont légèrement inférieurs à ceux des tests à 1600 fonctions de base par processeurs sur *Frontier*. On peut considérer que, pour cette taille, MDD a la même vitesse d'exécution sur les deux machines : l'écart de puissance entre les processeurs est compensé par la vitesse des communications.

4.2 Relaxation de la contrainte d'orthogonalité

Cette section détaille un peu plus le comportement numérique de l'algorithme.

Dans la première partie, on montre d'abord que, si l'on veut éviter de traiter des domaines de taille trop importante, le problème approché (3.6) ne peut fournir une solution précise du problème complet, à moins de relaxer la contrainte d'orthogonalité. Cela vient du fait que, en général pour des molécules réelles, les orbitales moléculaires les plus localisées ne sont pas réellement à support compact mais ont une décroissance exponentielle. C'est un problème largement discuté dans la littérature de la chimie computationnelle, référencé par l'expression "orthogonalization tails" [28, 32].

Ne pas imposer la stricte orthogonalité des blocs entre eux est également déterminant pour la précision de la solution obtenue par l'algorithme MDD : c'est l'objet de la deuxième partie.

La dernière partie porte sur l'importance du choix de l'algorithme utilisé pour rendre chaque bloc orthogonal à ses voisins.

Pour rester cohérent avec la présentation de l'algorithme faite aux paragraphes précédents, on continue d'écrire les formules en supposant $S = I$. Tous les tests numériques ont été effectués en tenant compte de la véritable matrice S.

Enfin, on introduit les notations :

- $\mathcal{G}_C(n, q)$ pour représenter l'ensemble des matrices de $\mathcal{M}^{N_b,N}$ qui ont la localisation illustrée par la figure 3.1 : chaque bloc a n lignes non nulles et deux blocs consécutifs ont q lignes non nulles en commun,
- $\mathcal{G}_D(n, q)$ est l'ensemble des matrices de $\mathcal{M}_S^{N_b}$ qui ont la localisation illustrée par la figure 3.2 : les blocs diagonaux sont des matrices de $\mathcal{M}_S^n$ et deux blocs diagonaux consécutifs définissent une matrice de $\mathcal{M}_S^q$.

4.2.1 Localisation des orbitales moléculaires et orthogonalité

Remplacer le problème de la détermination des orbitales moléculaires (3.5) par le *problème approché* (3.6), où on détermine des orbitales moléculaires localisées *a priori*, revient à rechercher dans $\mathcal{G}_C(n, q)$ une base orthonormée du sous-espace des vecteurs propres associés aux N plus petites valeurs propres de H. Rappelons que l'existence d'une base ayant ce type de localisation n'est pas prouvée mathématique-

ment et qu'on s'oriente vers cette décomposition à partir de considérations physiques qualitatives basées sur le principe de localité.

Il n'y a pas à notre connaissance de résultats permettant d'estimer l'extension minimale des orbitales moléculaires[5] à partir de l'extension effective de la densité En pratique, dans MDDla largeur de bande de la matrice densité, cf. figure 3.2). Il faut donc passer à des tests numériques pour s'en faire une idée un peu plus précise.

L'objet de cette section est d'étudier, sur un cas concret et pour des localisations raisonnables, l'existence d'orbitales moléculaires ayant une localisation fixée *a priori* et de montrer que l'impact de la localisation dans le problème approché s'effectuera via la contrainte d'orthonormalité des orbitales (ou idempotence de la matrice densité).

Dans un premier temps on va essayer de construire une matrice $C \in \mathcal{G}_C(n,q)$ à partir de combinaisons des vecteurs propres de H. Pour simplifier l'exposé, on présente ce test numérique comme si on avait $S = I$. En réalité, la non-orthogonalité des fonctions de base a été prise en compte dans les calculs. Fixons nous comme matrice H la matrice obtenue pour la molécule de la famille $\mathcal{P}_3$ avec 188 atomes de Carbone (1316 fonctions de base, pour 737 paires d'électrons) et notons V la matrice des vecteurs propres de H associés aux 737 plus petites valeurs propres :

$$V \in \mathcal{M}(N_b, N)\ , \quad V^t V = I_N.$$

Après diagonalisation de H, cherchons donc à construire numériquement des combinaisons linéaires de vecteurs propres de H dont les coefficients sont tous nuls sauf pour le bloc i.

On résout ce problème tour à tour pour chaque bloc i, indépendamment des blocs voisins, ce qui revient à construire des blocs C_i en abandonnant la contrainte d'orthogonalité entre blocs différents.

Trouver un bloc localisé pour les fonctions du domaine i revient à calculer le noyau de la sous-matrice de V obtenue en supprimant toutes les lignes correspondant au bloc i. Pour le premier bloc par exemple, cela revient à écrire V sous la forme :

$$V = \begin{pmatrix} V_1 \\ V_2 \end{pmatrix}$$

avec $V_1 \in \mathcal{M}^{n,N}$ et $V_2 \in \mathcal{M}^{N_b-n,N}$, n étant la taille d'un bloc. Trouver des vecteurs localisés sur le premier bloc revient à trouver une matrice $R \in \mathcal{M}^{N,m}$ qui vérifie :

$$VR = \begin{pmatrix} V_1' \\ 0 \end{pmatrix}$$

Il suffit pour cela que les colonnes de R soient dans le noyau de V_2. On aura un m maximal en prenant pour R une base du noyau de V_2. Dans ce cas, les colonnes

[5] En pratique dans MDD, le paramètre q, introduit au paragraphe 3.5.1 comme le nombre de fonctions de base à prendre dans le recouvrement entre 2 blocs voisins, représente cette extension minimale.

de VR fournissent bien une combinaison linéaire de vecteurs propres strictement localisés dans le premier domaine.

Le tableau 4.3 donne la dimension du noyau de V_2 (obtenue par une décomposition en valeurs singulières, avec un seuil à 10^{-13}) pour chacun des 6 blocs correspondant à une localisation très proche de celle qui a été utilisée pour les tests numériques de la section 3.5. On a ainsi le nombre maximal de vecteurs indépendants s'écrivant exactement comme une combinaison linéaire de vecteurs propres de H et strictement localisés (à la 10^{-13} près) dans le bloc. La somme de tous ces nombres est 740, soit légèrement plus que la valeur de N. Les vecteurs obtenus à l'intérieur d'un même bloc sont orthogonaux, mais c'est loin d'être le cas pour les vecteurs entre blocs différents. En effet, le cosinus directeur le plus élevé entre le sous-espace engendré par les vecteurs du bloc i et le sous-espace engendré par les vecteurs du bloc $i+1$ est toujours supérieur à 0.8.

Ce test montre que, pour cette localisation, il est impossible de respecter à la fois la contrainte de localisation et d'orthogonalité en prenant des combinaisons linéaires exactes des orbitales canoniques de H, et suggère l'introduction d'une tolérance sur l'orthogonalité entre les blocs. La localisation n'est pourtant pas particulièrement contraignante puisque la demi-largeur de bande de la matrice densité localisée est égale à 125 et correspond à tous les termes supérieurs à 10^{-6} dans la matrice densité exacte.

Numéro du bloc	1	2	3	4	5	6
Nombre de vecteurs	146	118	111	111	111	143

TAB. 4.3 – Nombre maximal de vecteurs indépendants s'écrivant comme une combinaison linéaire de vecteurs propres de H et strictement localisés (à 10^{-13} près) dans chaque bloc. Ces 6 blocs correspondent aux paramètres de localisation : $n = 322$ et $q = 126$.

On peut arriver à la même conclusion en étudiant l'approximation induite par la localisation du point de vue de la matrice densité.

Notons $D_t \in \mathcal{G}_D(n,q)$ la matrice densité tronquée obtenue à partir de la densité exacte $D_\star$ (introduite par les formules (3.2)) en remplaçant par 0 tous les coefficients situés hors de la zone de localisation (voir figure 3.2).

Rappelons que la solution $D_\star$ du système (3.2) peut être caractérisée par :

$$D_\star = \inf\left\{ \mathrm{Tr}\,(H\ D)\,, D \in \mathcal{M}_S^{N_b}, D^2 = D,\ \mathrm{Tr}\,(D) = N \right\}. \tag{4.1}$$

Remplacer le problème complet (3.5) par le problème approché (3.6) revient à chercher la matrice densité dans l'ensemble :

$$\left\{ D \in \mathcal{G}_D(n,q),\ D^2 = D,\ \mathrm{Tr}\,(D) = N \right\}. \tag{4.2}$$

L'algorithme MDD ne minimise pas réellement $\mathrm{Tr}\,(H\ D)$ sur cet ensemble car il n'y a pas équivalence entre la localisation sur les orbitales moléculaires et la localisation sur la matrice densité. Toute matrice $C \in \mathcal{G}_C(n,q)$ fournit bien $D \in \mathcal{G}_D(n,q)$ mais la réciproque est fausse. Cependant, on souhaite que la densité construite à partir de la solution calculée par MDD approche le mieux possible la matrice D_t, et donc que D_t appartienne à l'ensemble (4.2).

Malheureusement, à moins d'avoir des relations très particulières dans la matrice $D_\star$, l'écart à l'idempotence de la matrice D_t est du même ordre de grandeur que l'erreur de troncature $||D_\star - D_t||$:

$$||D_t^2 - D_t|| \approx ||D_\star - D_t||.$$

Ceci vient du fait que $D_\star$ est une matrice bande idempotente, dont la diagonale et les premières sur-diagonales contiennent des termes de l'ordre de 1. On retrouve ce résultat sur nos tests numériques (voir figure 4.3), signe qu'il n'y a pas de relations particulières qui viennent compenser les écarts à l'idempotence.

On en déduit que, même si on savait résoudre le problème approché (3.6) de manière exacte, cela ne nous fournirait pas D_t puisque cette matrice ne respecte pas les contraintes. Ceci prouve qu'il est nécessaire de relaxer la contrainte d'idempotence dans (4.2), et donc d'orthogonalité dans (3.6), pour que la matrice D_t appartienne à l'espace de minimisation, et ce d'autant plus que l'erreur de troncature induite par la localisation choisie est grande. Dès que D_t est dans (4.2), on est assuré que D_t est le minimum du problème avec localisation, si toutefois $H \in \mathcal{G}_D(n,q)$. En effet, $\mathrm{Tr}\,(H\ D)$ ne fait intervenir que les termes de D correspondant à des termes non nuls de H.

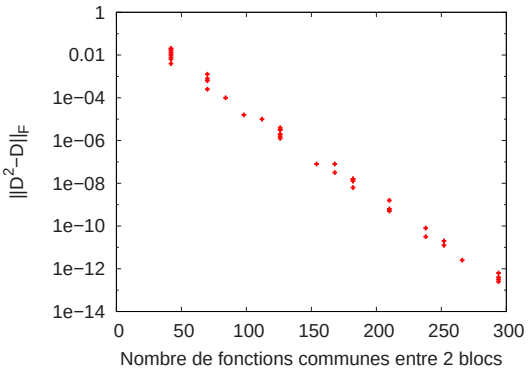


FIG. 4.3 – Norme de Frobenius de la matrice $D_t^2 - D_t$ pour différentes valeurs des paramètres de localisation n et q (l'abscisse représente le paramètre q), dans le cas d'une matrice H correspondant à un alcane dans une base STO-3G, famille $\mathcal{P}_3$. De plus, on constate sur ce cas que l'erreur de commutation (non indiqué sur le graphe), $||HD - DH||$ est, elle aussi, du même ordre de grandeur que l'erreur de troncature.

4.2.2 Orthogonalité et convergence de l'algorithme

Dégrader l'orthogonalité entre les blocs est également déterminant pour la précision de la solution obtenue par l'algorithme, indépendamment de l'erreur introduite par la localisation. On introduit la tolérance au niveau de l'étape locale et au cours de l'étape globale l'ordre de grandeur des produits scalaires est maintenu (voir section 5.1.1).

La présente section décrit dans un premier temps la méthode de résolution de l'étape locale et en particulier comment on introduit un écart à l'orthogonalité, écart représenté par la suite par ϵ. On montre dans un second temps l'effet de cette tolérance sur la convergence de la méthode MDD. L'orthogonalité obtenue effectivement dans le calcul est bien meilleure que la tolérance introduite. On tente d'expliquer ce point dans un troisième temps. Dans cette section, les illustrations numériques correspondent toutes à la simulation d'une molécule de la famille $\mathcal{P}_3$ avec 93 monomères, avec une décomposition correspondant à $q = 210$, soit une erreur de troncature sur D de 6×10^{-10}.

Algorithme pour l'étape locale. Au cours de l'étape locale à l'itération $k+1$, on résout, pour chaque bloc i :

$$\inf\Big\{ \mathrm{Tr}\left(H_i C_i C_i^t\right), \qquad C_i \in \mathcal{M}^{n,\tilde{m}_i}(\mathbb{R}), \quad C_i^t C_i = I_{\tilde{m}_i}, \\ [C_{i-1}^k]^t T C_i = 0, \quad C_i^t T C_{i+1}^k = 0 \Big\}, \tag{4.3}$$

où $\tilde{m}_i$ est pris strictement supérieur à m_i pour permettre d'ajuster, une fois tous les domaines calculés, le nombre d'orbitales de chaque domaine[6] (le nombre de colonnes dans chaque bloc C_i).

Comme on l'a déjà présenté dans le chapitre 3, paragraphe 3.3, l'idée est de calculer V une base orthonormée du sous-espace :

$$\mathcal{V} = \left\{ x \in \mathbb{R}^n, \quad \left[C_{i-1}^k\right]^t T x = 0, \quad x^t T C_{i+1}^k = 0 \right\}, \tag{4.4}$$

et de diagonaliser la matrice : $V^t\, H_i\, V$.

$\mathcal{V}$ est le noyau de la transposée de la matrice :

$$A = [T^t \left[C_{i-1}^k\right] \ , \quad T C_{i+1}^k]. \tag{4.5}$$

On introduit une tolérance sur l'orthogonalité entre les blocs en remplaçant $\mathcal{V}$ par le sous-espace vectoriel V_ϵ de dimension maximale tel que :

$$\forall x \in V_\epsilon, \ \frac{||A^t\, x||_2}{||x||_2} < \epsilon \tag{4.6}$$

[6] C'est l'étape dite d'échange, décrite dans [11]. Elle consiste à choisir le N-uplet (m_i) donnant l'énergie $\mathrm{Tr}\,(HCC^t)$ minimale sous la contrainte $\sum_i m_i = N$.

Les valeurs singulières d'une matrice ont une caractérisation minimax (analogue au théorème de Courant-Fischer pour les valeurs propres) qui montre que V_ϵ peut être obtenu à partir de la décomposition en valeurs singulières de la matrice A. V_ϵ est constitué des vecteurs singuliers à gauche de la matrice A correspondant aux valeurs singulières inférieures à ϵ. On est ainsi assuré d'avoir *un* sous-espace de $\mathbb{R}^n$ constitué de vecteurs dont le produit scalaire avec tout vecteur des blocs voisins est inférieur à ϵ. Le sous-espace ainsi calculé est maximal dans la mesure où il n'existe pas de sous-espace de dimension strictement plus grande où tous les vecteurs sont orthogonaux aux blocs voisins à ϵ près. Mais il n'est pas unique puisque chaque vecteur de base de ce sous-espace pourrait être légèrement perturbé et conserver les mêmes propriétés d'orthogonalité approché vis-à-vis des blocs voisins. Cette remarque sera sera utilisée plus loin dans la section 4.3.2.
Dans les tests présentés dans ce document, on a toujours pris $\tilde{m}_i$ maximal, c'est-à-dire égal à la dimension de V_ϵ.
Notant $X \in \mathcal{M}^{\tilde{m}_i,\tilde{m}_i}$ la matrice des vecteurs propres de $V^t\ H_i\ V$, les colonnes de VX fournissent $\tilde{m}_i$ vecteurs colonnes parmi lesquels seront choisis, au cours de l'étape d'échange, ceux de $\widetilde{C}_i^{k+1}$.

L'effet de ϵ sur la convergence. Pour les décompositions où le recouvrement entre les domaines est grand, l'erreur de troncature sur la densité est très faible et le problème localisé est une très bonne approximation du problème complet. Les arguments développés dans la partie 4.2.1 ne sont plus valides. Relaxer la contrainte d'orthogonalité dans le problème approché (3.6) ne se justifie plus à ce niveau.

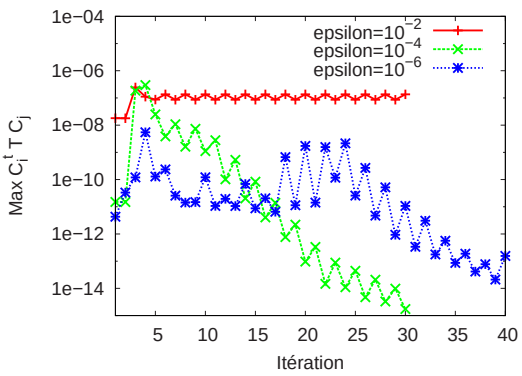


FIG. 4.4 – Orthogonalité entre les blocs pour différentes valeurs de ϵ.

Il est cependant crucial d'utiliser un paramètre ϵ suffisamment grand dans le traitement de l'étape locale (voir figures 4.4 et 4.5), quitte à récupérer une orthogonalité exacte à la précision machine près en fin de simulation (voir le cas $\epsilon = 10^{-4}$ sur les figures). L'effet de ϵ sur la solution de chaque problème local (4.3) est clair : la dimension du sous-espace $\mathcal{V}$ est d'autant plus grande que ϵ est grand, et donc le minimum obtenu pour le problème (4.3) est inférieur. Ceci explique la position relative des courbes sur les premières itérations de la figure (4.5).

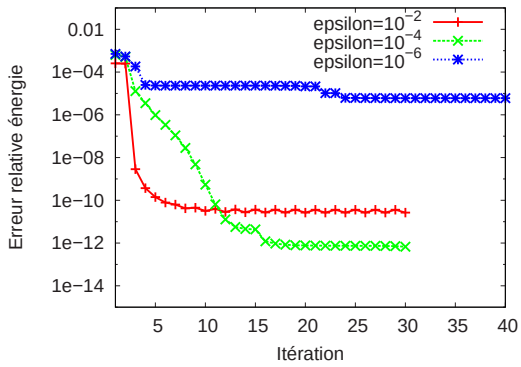


FIG. 4.5 – Précision sur l'énergie pour différentes valeurs de ϵ. L'erreur sur la densité suit la même évolution que l'erreur sur l'énergie. Les valeurs de ϵ : 10^{-2}, 10^{-4}, 10^{-6}, donnent respectivement en fin de simulation pour e_∞ (défini à la formule (3.21) p. 54) : 6×10^{-5} , 10^{-5} , 2×10^{-2}.

Il se trouve, en outre, qu'un choix inapproprié de ϵ ($\epsilon = 10^{-6}$ dans notre cas) conduit à un minimum local qui est loin de la solution du problème approché. D'autre part, prendre ϵ le plus grand possible (tout en gardant une orthogonalité acceptable) ne suffit pas toujours, voir à ce titre la courbe obtenue pour $\epsilon = 10^{-2}$. Ce comportement de MDD n'est pas encore élucidé.

L'orthogonalité effective est bien meilleure que l'écart toléré. Il est remarquable que l'orthogonalité entre les blocs de la solution calculée par MDD soit bien meilleure que ce qu'on impose réellement via ϵ (cf. figure 4.4). Dans les tests numériques réalisés jusqu'à présent, on n'a pas observé de *gap* significatif dans les valeurs singulières de la matrice A (4.5) : la plus grande valeur singulière strictement inférieure à ϵ est toujours du même ordre de grandeur que ϵ, donc fournit un vecteur dont le produit scalaire avec une colonne d'un bloc voisin peut s'élever jusqu'à ϵ.

On observe en fait, que les composantes des vecteurs propres de V^tHV sur les vecteurs ayant les valeurs singulières les plus élevées restent faibles. Pourtant, ces composantes ont une importance déterminante pour la convergence de MDD, puisque ne pas les prendre en compte (*i.e.* faire un calcul avec un ϵ trop faible) dégrade la convergence.

Terminons sur une dernière observation concernant l'évolution de l'orthogonalité dans les tests numériques réalisés : pour ϵ pas trop grand, l'orthogonalité continue de s'améliorer de manière très significative alors que l'énergie ne décroît plus (cf figures 4.4 et 4.5, pour $\epsilon = 10^{-4}$ et $\epsilon = 10^{-6}$). Autrement dit, en termes géométriques : alors que l'algorithme s'éloigne dans un premier temps du sous-ensemble défini par les contraintes du problème approché (3.6) pour trouver le minimum de l'énergie, il revient naturellement sur ce sous-ensemble dans un second temps en conservant une énergie constante.

4.2.3 Importance de l'algorithme d'orthogonalisation

Calculer une décomposition en valeurs singulières est une opération coûteuse qui n'avait pas été considérée dans un premier temps. On décrit dans cette section la procédure utilisée pour le calcul de $\mathcal{V}$ dans la première version de MDD[7], et en quoi elle est inadaptée.

Dans [11], une base orthonormée de $\mathcal{V}$ est construite comme base orthonormée du supplémentaire orthogonal de la matrice (4.5), dont on commence par calculer une base orthonormée par un algorithme de Gram-Schmidt itéré (voir [93], p.92-93). Pour tenir compte du fait que la matrice (4.5) n'est pas de rang maximal, au cours du processus d'orthonormalisation, on élimine tout vecteur lié aux précédents, en pratique tout vecteur dont la projection sur le supplémentaire orthogonal des premiers vecteurs de la base est inférieur à ϵ.

Cette manière de traiter la dégénerescence du rang de la matrice (4.5) est rigoureuse en arithmétique exacte mais ne l'est plus en présence d'erreurs d'arrondis. En effet, les vecteurs ainsi construits pour $\mathcal{V}$ sont bien orthogonaux à ceux des blocs voisins à ϵ près (voir la figure 4.6), mais ils sont en nombre systématiquement inférieur à la dimension de $\mathcal{V}$, telle que calculée par décomposition en valeurs singulières pour le même seuil ϵ[8]. Il s'ensuit qu'il manque des directions dans le sous-espace sur lequel on minimise dans l'étape locale. On résout donc mal les problèmes locaux et la solution calculée par l'algorithme reste éloignée du minimum de (3.6), quelle que soit la valeur de ϵ (cf figure 4.7).

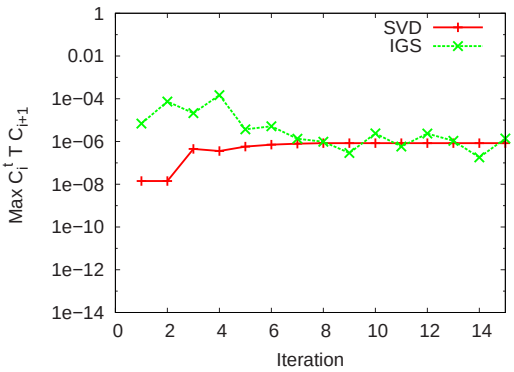


FIG. 4.6 – Evolution de l'orthogonalité entre les blocs en calculant le sous-espace $\mathcal{V}$ par décomposition en valeurs singulières (SVD) ou par une méthode de Gram-Schmidt (IGS). Dans les deux cas : $\epsilon = 10^{-2}$. Ces simulations ont été réalisées sur des molécules de types "alcane" (famille $\mathcal{P}_3$).

[7] Version correspondant aux articles [11], [A1] et [A2]

[8] Ceci a été d'abord observé sur des matrices construites aléatoirement, puis confirmé sur les matrices des cas test de ce travail. Pour la famille $\mathcal{P}_3$ par exemple, l'écart entre la dimension calculée par décomposition en valeurs singulières et par Gram-Schmidt est de l'ordre de 10 à 20, pour des sous-espaces $\mathcal{V}$ de dimension de l'ordre de 200.

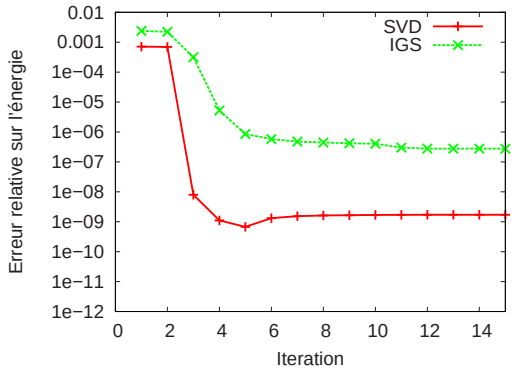


FIG. 4.7 – Evolution de l'erreur relative sur l'énergie, e_E, dans les mêmes conditions que dans la figure 4.6. Sur la courbe "SVD", on observe une augmentation de e_E à partir de la cinquième itération, signe que l'énergie calculée par MDD est devenue inférieure à l'énergie de référence.

4.3 Conclusion de la partie 1D

4.3.1 Synthèse des améliorations par rapport à [11]

Le travail effectué sur la version "1D" de l'algorithme MDD a conduit à des améliorations :

- en termes de précision de la solution calculée, par une modification de l'étape locale (cf section 4.2.3),
- en termes de temps de calcul et de mémoire nécessaire, par une modification de l'implémentation de l'étape globale (cf formule (3.20), et section 3.5.2)

La figure 4.8 montre l'effet cumulé de ces deux améliorations sur le calcul d'une molécule de la famille $\mathcal{P}_3$.

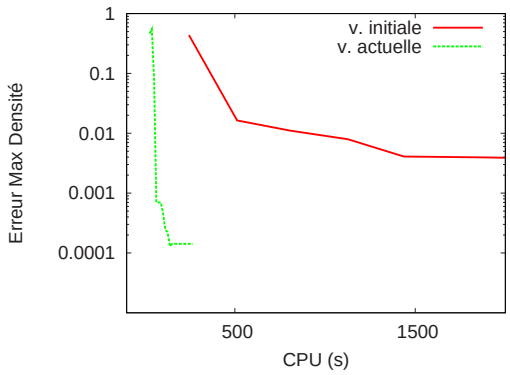


FIG. 4.8 – Simulation sur une matrice de la famille $\mathcal{P}_3$, avec 230 atomes de Carbone. Le temps CPU a été obtenu pour un processeur Intel Pentium M, 1.6GHz, avec une mémoire Cache de 1 Mo.

La figure 4.9 montre l'effet de la taille du recouvrement, sur le cas "polyéthène". Quand on augmente le recouvrement, la précision de la solution calculée par MDD augmente, comme attendu. Pour les recouvrements où l'erreur de troncature est comprise entre 10^{-2} et 10^{-4}, MDD se montre "optimal" dans le sens où la précision obtenue sur la matrice densité est du même ordre de grandeur que l'erreur de troncature. En revanche, quand le recouvrement est grand, il devient difficile d'augmenter la précision de la solution calculée par MDD et l'erreur est de plusieurs ordres de grandeur plus importante que l'erreur de troncature.

4.3.2 Améliorations envisageables pour l'étape locale

Choix de ϵ. Le paramètre ϵ introduit dans l'algorithme de l'étape locale a une grande influence sur la qualité du résultat fourni par MDD. Au vu des tests numériques réalisés jusqu'à présent et présentés dans la section 4.2.2, il semble qu'il existe un choix optimal de ϵ : trop faible, la solution reste bloquée dans un minimum local

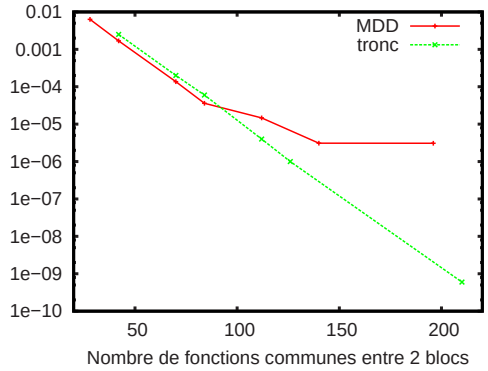


FIG. 4.9 – Simulations portant sur une molécule de la famille $\mathcal{P}_3$ avec 93 monomères. La courbe "MDD" donne e_∞ (cf formule 3.21), pour différentes décompositions spécifiées par la valeur de q (en abscisse). La courbe "tronc" rappelle l'erreur de troncature, *i.e.* e_∞ pour la matrice densité exacte. Le résultat de MDD est plus précis quand le recouvrement entre deux blocs augmente.

de l'énergie, trop grand, l'énergie converge relativement bien mais l'orthogonalité n'est pas acceptable pour la matrice densité calculée. Il est souhaitable d'automatiser le choix de ϵ par rapport à un problème donné (type de molécule, paramètres de localisation adoptés). On pourrait même penser à le rendre adaptatif au cours des itérations. Notre analyse du comportement de MDD en fonction de ce paramètre n'est pas encore assez avancée, et notre expérience sur les tests numériques pas encore assez riche, pour qu'on puisse proposer un choix *a priori* ou une stratégie pour faire évoluer ϵ : c'est une voie qu'on propose d'explorer pour la suite du développement de MDD.

Algorithme pour le calcul d'une base de $\mathcal{V}$. La décomposition en valeurs singulières est la méthode la plus fiable pour calculer le noyau d'une matrice. Néanmoins, la quantité d'informations fournies dépasse largement les besoins de l'étape locale : outre une base orthonormée du noyau, on obtient une base orthonormée de l'image et toutes les meilleures approximations de rang 1, 2, 3 etc. de l'opérateur (en norme euclidienne). On peut se demander si des méthodes moins coûteuses et calculant uniquement le noyau ne pourraient pas convenir.

Parmi elles, les méthodes calculant une décomposition QR adaptée aux matrices singulières (*rank revealing QR decomposition*, [94, 102]) présentent une alternative qu'il serait intéressant de tester. La robustesse de ces méthodes est parfois dénigrée [109], mais il semble que des progrès ont été récemment réalisés sur cette question [104], en particulier à partir de la version 3.1.0 de la bibliothèque LAPACK.

Algorithme pour l'étape locale. Dans l'étape locale, il faut résoudre, pour chaque bloc de C pris tour à tour, un problème de minimisation sous contraintes.

La méthode choisie consiste à fixer un sous-espace respectant les contraintes venant des blocs voisins à ϵ près, puis à minimiser $\mathrm{Tr}\left(H_i C_i C_i^t\right)$ sur ce sous-espace en traitant les contraintes internes au bloc, et sans tenir compte de la solution calculée à la fin de l'itération précédente. Notons deux inconvénients liés à cette approche.

D'une part, on pourrait gagner en rapidité en passant à un procédé itératif, qui permettrait d'utiliser l'information contenue dans C_i^k pour le calcul de $\widetilde{C}_i^{k+1}$, du moins quand les tailles de bloc m_i sont stabilisées et qu'il n'est plus nécessaire d'avoir $\tilde{m}_i > m_i$, ce qui arrive en pratique après quelques itérations de MDD.

D'autre part, il arrive que l'énergie obtenue pour $\widetilde{C}_i^{k+1}$ soit légèrement plus élevée que celle de C_i^k, ce qui est anormal puisque C_i^k devrait faire partie des solutions admissibles pour le problème de minimisation (4.3). En fait, même si C_i^k vérifie l'orthogonalité à ϵ près, il n'est pas forcément dans le sous-espace V_ϵ. Le minimum de l'énergie sur V_ϵ peut-être plus élevé que l'énergie de C_i^k. Ce défaut de monotonie de l'étape locale est d'autant plus fréquent que ϵ est élevé. On en voit une illustration à la figure 4.5 : pour le cas $\epsilon = 10^{-2}$, l'énergie n'est plus décroissante après la huitième iteration.

Une manière de régler ces deux difficultés serait de résoudre de manière itérative le problème suivant :

$$\inf\Big\{ \mathrm{Tr}\left(H_i C_i C_i^t\right), \qquad C_i \in \mathcal{M}^{n,\tilde{m}_i}(\mathbb{R}), \quad ||C_i^t C_i - I_{\tilde{m}_i}|| < \epsilon, \\ ||[C_{i-1}^k]^t T C_i|| < \epsilon, \quad ||C_i^t T C_{i+1}^k|| < \epsilon \Big\}. \tag{4.7}$$

4.3.3 Comparaison de deux algorithmes de calcul de la densité : MDD version 1D et diagonalisation de la matrice de Fock

Pour terminer ce chapitre, on présente de nouveaux résultats de comparaison de MDD à la manière standard de calculer la matrice densité. On liste ci-dessous les éléments supplémentaires apportés par cette comparaison par rapport au paragraphe 3.5.2.

- **Le choix de la méthode de diagonalisation :** on a privilégié la vitesse d'éxecution, au prix d'un surcoût en mémoire. En effet, "Divide-and-Conquer" s'avère être la méthode la plus rapide pour le type de matrices que l'on traite ici [103]. Les calculs ont été réalisés avec un stockage plein pour les matrices H et S. Dans la perspective du calcul de la matrice densité, cela ne provoque pas de surcroît d'utilisation de mémoire par rapport à un stockage symétrique, et cela améliore nettement les performances [117]. Pour ces choix, avec 8 Go de mémoire RAM, la taille maximale est : $N_b = 14322$.
  Un autre choix aurait pu être : méthode QR et stockage bande des matrices H et S. Le gain en mémoire est significatif, mais la pénalisation en vitesse de calcul est très importante. On peut effectivement traiter des matrices plus

grosses mais le temps de calcul est tellement long[9] que cela n'a pas de sens dans le cadre de la comparaison à MDD.

- **Les processeurs utilisés :** deux processeurs différents ont été utilisés, que l'on nomme ici *P1* et *P2*. *P1* est le modèle *Intel Xeon E5410*, 2.33 GHz. Il a deux unités de calcul comportant chacune quatre cœurs et 6 Mo de mémoire "cache", pour 8 Go de mémoire RAM. *P2* est le modèle *Dual AMD Opteron 280* à 2.39 GHz. Il a deux unités de calcul comportant chacune deux cœurs et 1 Mo de mémoire cache, l'ensemble partageant 16 Go de mémoire RAM.
- **La bibliothèque d'algèbre linéaire :** c'est un choix critique pour la performance des codes de diagonalisation mais aussi pour celle de MDD. Tous les tests présentés ici sont basés sur la bibliothèque INTEL MATH KERNEL LIBRARY version 10.0.
- **Les molécules testées :** en plus de molécules d'alcane de la famille $\mathcal{P}_3$, une comparaison a été réalisée sur des chaînes linéaires d'alcène CH_2-$(CH)_{n_m}$-CH_2. C'est un cas plus difficile pour MDD car les électrons sont plus délocalisés, en raison de la conjugaison des doubles liaisons.

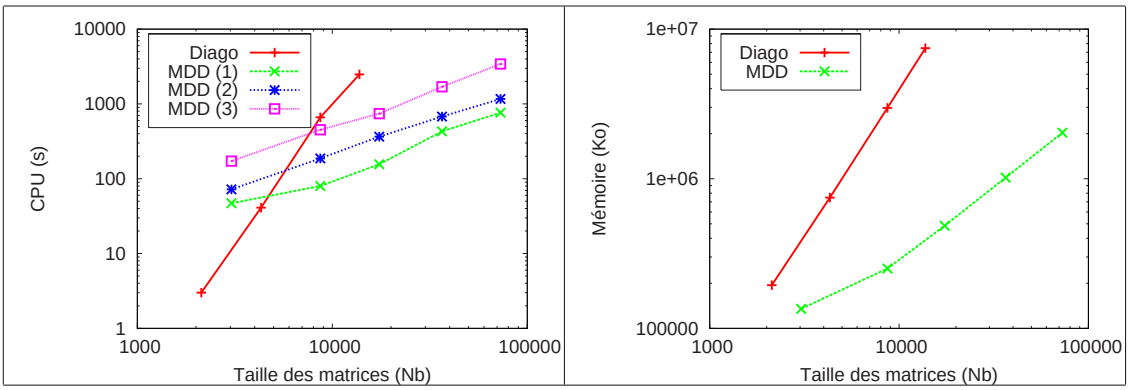


FIG. 4.10 – Calcul de la densité pour la série "alcane" sur le processeur *P1*. Comparaison du temps de calcul et de la mémoire nécessaire pour MDD et pour la diagonalisation. Voir le texte pour la signification de la légende.

Les figures 4.10 et 4.11 montrent les résultats de la comparaison. Pour MDD, on indique le temps de calcul correspondant à différents seuils de précision atteints à la fois sur l'énergie et sur les termes calculés dans la matrice densité (e_E et e_∞ sont définis à la page 54) :

Seuil	$e_E < ...$	$e_\infty < ...$
(1)	10^{-8}	10^{-3}
(2)	10^{-10}	10^{-4}
(3)	10^{-12}	10^{-5}

[9] Pour le processeur *P1* à 8 Go de mémoire, on peut aller jusqu'à $N_b = 22000$ mais la simulation dure plus de 36 h !

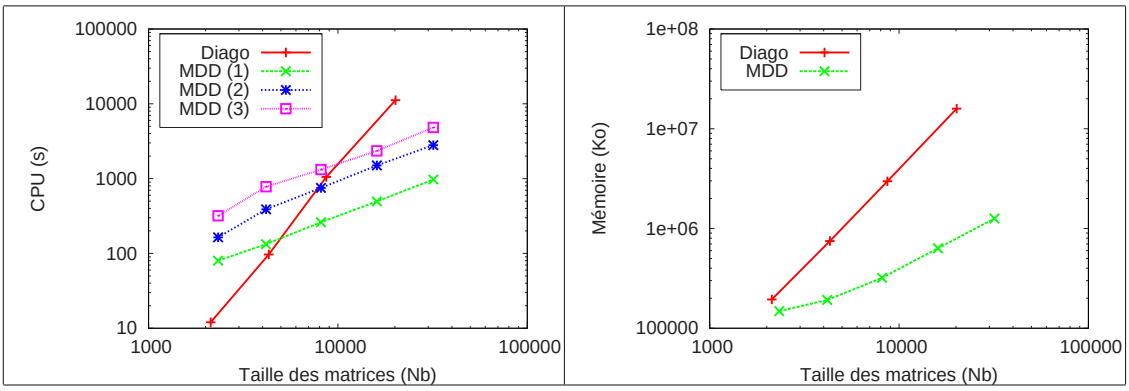


FIG. 4.11 – Calcul de la densité pour la série "alcène" sur le processeur *P2*. Comparaison du temps de calcul et de la mémoire nécessaire pour MDD et pour la diagonalisation. Voir le texte pour la signification de la légende.

Pour chacune des simulations MDD présentées ici, le temps de calcul de chaque itération de MDD se décompose en parts égales entre les étapes locales et les étapes globales. Pour les deux familles de molécules, on observe que :

- Le temps de calcul pour MDD évolue linéairement avec la taille de la matrice, pour les trois niveaux de précision. Ceci est le signe que le nombre d'itérations pour atteindre la précision indiquée n'augmente pas avec la taille de la molécule, puisque le nombre d'opérations pour chaque itération de MDD croît linéairement avec N_b. Le point de croisement du temps de calcul de MDD et de la diagonalisation est compris entre 5×10^3 et 10^4 fonctions de base (entre 1000 et 2000 atomes de carbone). La taille N_b à partir de laquelle MDD devient plus avantageuse est plus élevée que sur la courbe de la figure 3.6. Cela provient d'une part de l'utilisation d'un algorithme de diagonalisation plus performant, d'autre part du fait que la diagonalisation met en œuvre des matrices de taille plus importante, ce qui maximise le rapport du nombre d'opérations sur le nombre d'accès mémoire et optimise l'exploitation de tous les cœurs du processeur.
- Pour $N_b > 10^4$, la mémoire utilisée par MDD est inférieure d'un facteur 10 à la mémoire utilisée par la diagonalisation.

Dans la comparaison entre MDD et la diagonalisation, le point de croisement est obtenu pour des tailles de molécule analogues dans le cas "alcane" et dans le cas "alcène". Ces résultats ne reflètent pas le fait que la simulation des molécules d'alcène nécessite d'utiliser des blocs de taille plus importante (en raison de la plus grande délocalisation des orbitales). Cette anomalie provient probablement de la différence des processeurs.

Pour terminer ce chapitre, le tableau suivant synthétise les caractéristiques des plus grosses simulations réalisées pour les matrices venant de l'alcane (sur le processeur *P1* et sur 1000 processeurs de BG/L) et pour les matrices venant de l'alcène (processeur *P2*). Dans tous les cas, le temps de calcul est réparti de manière égale entre l'étape locale et l'étape globale.

Molécule	Machine	N_b	CPU (s)
Alcane	*P1*	$7.2\,10^4$	1100
Alcane	*BG/L*	$1.\,10^7$	3500
Alcène	*P2*	$5.\,10^4$	4650

Chapitre 5

Répartition quelconque des domaines : algorithme, implémentation, tests numériques

On a vu dans le chapitre 4 que la méthode de décomposition de domaine introduite dans [11] a de bonnes performances d'une part en termes de temps de calcul et de mémoire sur un unique processeur, d'autre part en termes de scalabilité dans un calcul distribué sur plusieurs processeurs. Pour que cette méthode puisse s'appliquer en pratique à des arrangements atomiques n'ayant pas forcément de direction particulière (voir figure 5.1), il faut lever la contrainte d'alignement des domaines que l'on énonce à nouveau :

chaque domaine partage des fonctions de base avec *au plus un* domaine à gauche et *au plus un* domaine à droite.

Dans tout ce chapitre on considère implicitement qu'on utilise une base de fonctions constituées d'Orbitales Atomiques et pour clarifier la présentation on représente les domaines en termes de répartition d'atomes. Il faut bien sûr garder à l'esprit qu'un domaine représente un ensemble de fonctions de base et qu'il y a recouvrement entre deux domaines différents dès que deux fonctions de base ont un produit scalaire non nul, ce qui est une condition moins restrictive que la seule localisation des atomes (voir chapitre 1, paragraphe 1.2.1).

L'adaptation de l'étape locale ne pose pas de problème. Chaque problème local de MDD garde la même structure : minimisation d'une fonctionnelle dépendant d'un opérateur local, avec contraintes d'orthonormalité et contraintes d'orthogonalité avec toutes les orbitales des domaines voisins. Rappelons que dans le schéma de résolution qu'on a adopté pour les problèmes locaux dans la version "1D" (cf section 4.2.2, p. 67), on travaille sur le sous-espace

$$\mathcal{V} = \left\{ x \in \mathbb{R}^n, \quad \left[C_{i-1}^k\right]^t Tx = 0, \quad x^t T C_{i+1}^k = 0 \right\}.$$

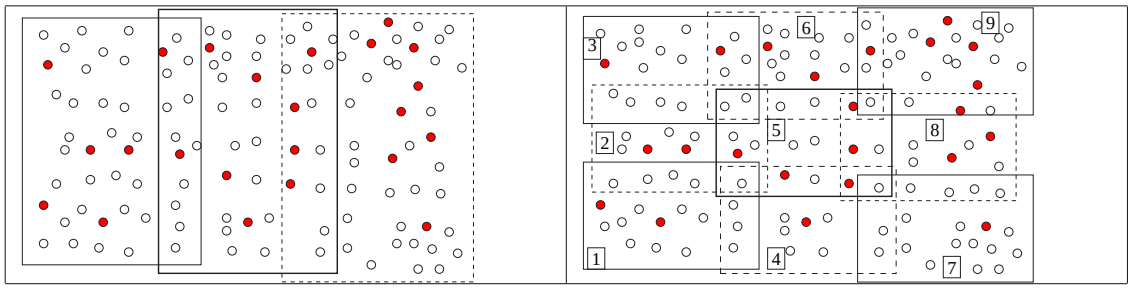


FIG. 5.1 – Répartition des domaines pour un système où la disposition des atomes est homogène en deux dimensions. A gauche, une répartition de domaines utilisable avec la version "1D" de l'algorithme : les domaines ont une grande taille. A droite, une répartition "cartésienne" où on découpe dans les deux directions du système : chaque domaine interagit avec au plus 8 voisins.

Lors du passage du schéma de gauche au schéma de droite dans la figure 5.1, le seul changement est que les contraintes d'orthogonalité viennent d'un plus grand nombre de voisins. Cela se traduit aussi par une augmentation du nombre de vecteurs définissant l'orthogonal de $\mathcal{V}$, et une diminution de n, la dimension de l'espace sur lequel $\mathcal{V}$ est défini. Il se peut que ce sous-espace soit de dimension nulle pour un domaine, i, ce qui signifierait qu'il n'existe aucune orbitale qui utilise les seules fonctions du domaine i et qui soit orthogonale à toutes les orbitales déjà définies sur les domaines voisins : cette situation serait révélatrice de la présence d'un trop grand nombre de domaines, ou du mauvais positionnement de certains d'entre eux.

En revanche, les modifications à apporter à l'étape globale sont bien plus profondes. Ceci vient du fait que certaines fonctions de base font partie de plus de deux domaines. La transformation définie par les équations (3.18) et (3.19) perd alors la propriété de conservation de l'orthogonalité entre les orbitales moléculaires (colonnes de C) de domaines différents. On pourrait imaginer se restreindre à des décompositions où chaque fonction de base n'appartient qu'à deux domaines au plus (pour $S = I$), mais cela restreindrait l'application de la méthode à des systèmes très particuliers (figure 5.2).

Ce chapitre comprend deux parties. La première est consacrée à l'adaptation de l'étape globale. La seconde décrit l'implémentation de l'algorithme sans contrainte géométrique de recouvrement entre les domaines. On conclut cette seconde partie par une estimation de la mémoire nécessaire et de la complexité de chaque étape.

Il est utile d'introduire dès maintenant une nouvelle notation pour représenter le produit scalaire entre orbitales de domaines voisins. Dans les conditions du chapitre 4, chaque domaine se recouvre de la même manière avec tous ses voisins : le recouvrement est entièrement déterminé par le nombre de fonctions de base commune entre voisins et on peut exprimer le produit scalaire inter-blocs avec la seule matrice

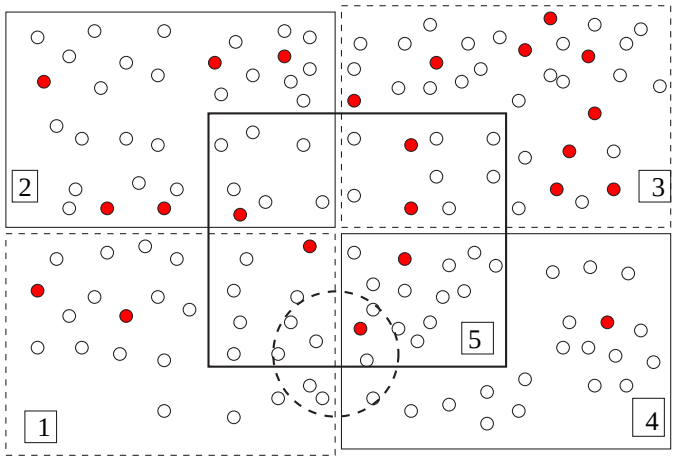


FIG. 5.2 – Répartition de domaines où chaque atome est dans deux domaines au plus. On suppose ici que $S = I$ de sorte qu'on puisse avoir effectivement deux domaines contigus qui ne se recouvrent pas. Avec cette décomposition on exclut *a priori* certaines combinaisons linéaires de fonctions de base "vivant" dans le même voisinage, par exemple, toute combinaison linéaire d'orbitales atomiques de la zone pointillée située sur les domaines 1, 4 et 5.

T. Avec une répartition "2D" ou "3D" des domaines, la position relative des voisins entre en compte et on introduit la notation suivante :
$S^{(ij)}$ représente la matrice de recouvrement (*overlap*, voir équations 1.6) entre les fonctions de base (Orbitales Atomiques) du domaine i et du domaine j. Concrètement, $S^{(ij)}$ la sous-matrice de S dont les lignes correspondent aux fonctions de base du domaine i et les colonnes à celles du domaine j.
Ainsi, le produit scalaire entre les orbitales du domaine i et celles du domaine j s'écrit :

$$C_i^t \, S^{(ij)} C_j. \tag{5.1}$$

On a bien sûr : $S^{(ji)} = [S^{(ij)}]^t$, relation qu'on utilisera désormais sans commentaire particulier.

5.1 Adaptation de l'étape globale au cas où plus de deux domaines se recouvrent

5.1.1 Retour sur la construction de l'étape globale dans le cas "1D"

Le but de cette section est de montrer comment ont été construites les formules (3.18)-(3.19) et comment on les met en œuvre numériquement dans la version "1D".

5.1.1.1 Du problème modèle au calcul d'orbitales moléculaires

Rappelons que l'étape globale est nécessaire pour minimiser correctement le problème approché (3.6), p. 47 : elle assure un traitement cohérent des contraintes et permet de minimiser la fonctionnelle complète.

La solution suggérée par l'étude d'un problème simplifié, dans lequel on a reproduit la même difficulté (chapitre 2), est d'ajouter une étape où on minimise la fonctionnelle complète $\mathrm{Tr}\,(HCC^t)$ sur l'ensemble des combinaisons linéaires des solutions obtenues à l'issue de l'étape locale, en imposant à ces combinaisons de garder l'orthogonalité entre les vecteurs.

Dans le cas du problème non simplifié (3.6), la formule (3.18), que l'on rappelle ci-dessous dans les notations adoptées dans ce chapitre, complétée par une orthonormalisation de chaque bloc $C_i(\mathcal{U})$, est une tentative pour générer ce même ensemble *en conservant la localisation* :

$$\begin{aligned} C_i(\mathcal{U}) \;&= \widetilde{C}_i^k \;+ S^{(i,i+1)}\widetilde{C}_{i+1}^k \quad U_i \quad \Big([\widetilde{C}_i^k]^t S^{(i,i+1)}\; [S^{(i,i+1)}]^t \widetilde{C}_i^k\Big) \\ &\qquad - [S^{(i-1,i)}]^t \widetilde{C}_{i-1}^k \quad U_{i-1}^t \quad \Big([\widetilde{C}_i^k]^t [S^{(i-1,i)}]^t\; S^{(i-1,i)}\widetilde{C}_i^k\Big). \end{aligned} \tag{5.2}$$

On *maintient* le produit scalaire maximal entre vecteurs de blocs voisins au niveau où il était en fin d'étape locale :

$$C_i(\mathcal{U})^t\; S^{(i,i+1)}\; C_{i+1}(\mathcal{U}) = \mathcal{O}\Big([\widetilde{C}_i^k]^t\; S^{(i,i+1)}\; \widetilde{C}_{i+1}^k\Big),$$

grâce à deux propriétés de la matrice de recouvrement :

1. $S^{(i,i+1)}\; S^{(i+1,i+2)} = 0$, ce qui est le cas dès que le recouvrement est limité au premier domaine voisin,
2. $S^{(i,i+1)}\; [S^{(i,i+1)}]^t S^{(i,i+1)} = S^{(i,i+1)}$.

Cette seconde propriété est clairement vérifiée dans le cas où S est la matrice identité. Dans le cas $S \neq I$, on a introduit une réorthogonalisation locale de S, domaine par domaine. On constate en pratique dans les tests numériques réalisés jusqu'à présent en "1D" que les matrices obtenues vérifient la propriété, sans avoir de preuve générale. On revient sur cette question dans la section 5.2.2.1.

En pratique, comme déjà signalé dans le chapitre 3, p. 51, on n'applique pas les transformations (5.2) simultanément à tous les blocs, mais à des sous-ensembles de 2 blocs adjacents. L'étape globale consiste alors à traiter tour à tour toutes les paires de blocs voisins.

5.1.1.2 Diminution du nombre de degrés de liberté

Dans les transformations (5.2), l'idée est de générer toutes les combinaisons linéaires possibles entre vecteurs colonnes de $\widetilde{C}_i^k$ et de $S^{(i,i+1)}\widetilde{C}_{i+1}^k$, tout en gardant l'orthogonalité. Les colonnes de $S^{(i,i+1)}\widetilde{C}_{i+1}^k$ sont liées, il est donc possible de diminuer le nombre de degrés de liberté contenus dans U_i en remplaçant la matrice

$S^{(i,i+1)}\widetilde{C}^k_{i+1}$ par une approximation de rang maximal, que l'on note B_{i+1}. Cela revient à retirer de U les degrés de liberté associés aux noyaux de $S^{(i,i+1)}\widetilde{C}^k_{i+1}$ et de $\left[S^{(i,i+1)}\right]^t \widetilde{C}^k_i$ qui n'ont aucun effet sur $C_i(U)$ et $C_{i+1}(U)$. La transformation (5.2) s'écrit maintenant[1] :

$$C_i(U) = \quad \widetilde{C}^k_i \quad + B_{i+1}\, U\, B^t_i \left(\left[S^{(i,i+1)}\right]^t \widetilde{C}^k_i \right) \tag{5.3}$$

$$C_{i+1}(U) = \quad \widetilde{C}^k_{i+1} \quad - B_i\, U^t\, B^t_{i+1} \left(S^{(i,i+1)}\widetilde{C}^k_{i+1} \right) \tag{5.4}$$

En pratique, B_i et B_{i+1}, sont calculés par décomposition en valeurs singulières des matrices $S^{(i,i+1)}\widetilde{C}^k_{i+1}$ et $\left[S^{(i,i+1)}\right]^t \widetilde{C}^k_i$. La matrice $S^{(i,i+1)}\widetilde{C}^k_{i+1}$ est remplacée par la matrice composée de ses vecteurs singuliers à gauche dont la valeur singulière dépasse un seuil, que l'on note ϵ_g. La valeur de ϵ_g doit être choisie avec soin :

- si elle est trop élevée, il manque des directions dans les colonnes de B_i, qui ne représentent pas complètement $S^{(i,i+1)}\widetilde{C}^k_{i+1}$, et cela perturbe la convergence de MDD,
- si elle est trop faible, on ne contrôle plus l'orthogonalité :
  la matrice $C_i(U)^t\ S^{(i,i+1)}\ C_{i+1}(U)$ comporte des termes qui ne sont plus négligeables.

La fin de cette section consiste en une analyse de ce comportement numérique.

La décomposition en valeurs singulières de la matrice $S^{(i,i+1)}\widetilde{C}^k_{i+1}$ s'écrit :

$$S^{(i,i+1)}\widetilde{C}^k_{i+1}\ Y = X\ \Sigma$$

avec $Y \in \mathcal{O}(m)$, $X \in \mathcal{O}(n)$ et $\Sigma \in \mathcal{M}^{(n,m)}$ matrice diagonale.
On décompose Σ sous la forme

$$\begin{pmatrix} \Sigma_a & 0 \\ 0 & \Sigma_b \end{pmatrix},$$

la sous-matrice Σ_a étant carrée et exclusivement composée des valeurs singulières supérieures à ϵ_g. On décompose de la même manière (en colonnes) les matrices X et Y :

$$X = [X_a\ X_b] \text{ et } Y = [Y_a\ Y_b]$$

On a ainsi :

$$B_{i+1} = X_a = S^{(i,i+1)}\widetilde{C}^k_{i+1}\ Y_a\ \Sigma_a^{-1} \tag{5.5}$$

De même :

$$B_i = [X']_a = [S^{(i,i+1)}]^t \widetilde{C}^k_i\ [Y']_a\ [\Sigma']_a^{-1} \tag{5.6}$$

Le produit scalaire : $C_i(U)^t\ S^{(i,i+1)}\ C_{i+1}(U)$ est la somme de deux termes :

- $\widetilde{C}^t_i\ S^{(i,i+1)}\ \widetilde{C}_{i+1}$,

[1] Pour deux blocs notés i et $i+1$, mais on peut aussi faire cette réduction dans le cas où on traite plus de deux blocs simultanément comme dans la formule (3.18).

– et un produit de 11 matrices faisant apparaître le facteur : $B_{i+1}^t S^{(i,i+1)} B_i$, qui devient, en utilisant (5.6) et (5.5) :

$$[\Sigma']_a^{-t}\ [Y']_a^t \left(\widetilde{C}_i^t\ S^{(i,i+1)}\ \widetilde{C}_{i+1}\right) Y_a\ \Sigma_a^{-1}.$$

La parenthèse centrale $\left(\widetilde{C}_i^t\ \ S^{(i,i+1)}\ \ \widetilde{C}_{i+1}\right)$ représente les produits scalaires obtenus en fin d'étape locale entre vecteurs du bloc i et vecteurs du bloc $i+1$. Le plus grand produit scalaire est donc potentiellement multiplié par ϵ_g^{-2}, ce qui explique la dégradation de l'orthogonalité entre blocs lorsque ϵ_g est trop faible.

Suite à cette analyse, on pourrait bien sûr adapter la valeur de ϵ_g à la plus grande valeur obtenue en fin d'étape locale dans les matrices : $\widetilde{C}_i^t\ S^{(i,i+1)}\ \widetilde{C}_{i+1}$, mais cela n'a pas encore été testé. Dans les résultats numériques présentés dans le chapitre 4, on a utilisé $\epsilon_g = \epsilon$ et cela suffit à ce que le produit scalaire reste stable au cours de l'étape globale.

5.1.2 Adaptation au cas "2D/3D"

Dans le cas où trois domaines se recouvrent, les formules présentées dans la section précédente ne permettent plus de maintenir l'orthogonalité entre les blocs. La recherche d'une formule généralisant (5.2)[2] a été infructueuse.

En revanche, on peut adapter la transformation à deux blocs (formules (5.3) et (5.4)) pour que chacun des deux reste orthogonal au troisième, qui reste fixe. On montre dans cette section comment faire cette adaptation d'un point de vue formel puis algorithmique.

Reprenons les formules (5.3)-(5.4) pour les domaines i et $i+1$ à l'itération k, et supposons qu'un troisième domaine, noté j, interagisse avec les deux premiers.

$$[\widetilde{C}_j^k]^t S^{(j,i)} C_i(U) = [\widetilde{C}_j^k]^t S^{(j,i)} B_{i+1}\ U\ B_i^t \left([S^{(i,i+1)}]^t \widetilde{C}_i^k\right)$$

Ce produit scalaire n'est pas nul, à moins d'imposer aux colonnes de B_{i+1} d'être orthogonales à celles de $S^{(i,j)}\widetilde{C}_j^k$. On parvient donc à la transformation annoncée en prenant pour colonnes de B_{i+1} une base de l'intersection :

– d'une part, du sous-espace engendré par les colonnes de $S^{(i,i+1)}\ \widetilde{C}_{i+1}^k$, pour avoir l'orthogonalité des blocs i et $i+1$, comme en "1D",
– d'autre part, du supplémentaire orthogonal du sous-espace engendré par les colonnes de toutes les matrices $S^{(i,j)}\widetilde{C}_j^k$, où j parcourt l'ensemble des domaines ayant un recouvrement à la fois avec le domaine i et le domaine $i+1$.

[2] Combinaisons des orbitales moléculaires des trois domaines, avec conservation de l'orthogonalité et de la localisation de ces orbitales.

De manière symétrique pour B_i, on doit calculer l'intersection de $S^{(i+1,i)}\ \widetilde{C}_i^k$ et de l'orthogonal des colonnes des matrices $S^{(i+1,j)}\widetilde{C}_j^k$ pour tous les domaines j interagissant avec les deux domaines i et $i+1$.

Notons que pour une décomposition "2D" comme celle de la figure 5.1 schéma de droite, il y a au maximum 4 domaines j. En généralisant cette décomposition en "3D", on aurait au maximum 16 domaines j.

Calcul de l'intersection de deux sous-espaces vectoriels. Nous nous sommes basés sur l'algorithme décrit dans [96], p. 604, qui fournit une base orthonormée de l'intersection de deux sous-espaces $\mathcal{A}$ et $\mathcal{B}$ à partir de la séquence suivante :

1. calculer Q_A et Q_B bases orthonormées de $\mathcal{A}$ et $\mathcal{B}$,
2. déterminer la décomposition en valeurs singulières de $Q_A^t Q_B$: $Y\ \Sigma X^t$,
3. sélectionner les valeurs singulières égales à 1 et les vecteurs singuliers associés : Y_A pour les vecteurs à gauche, X_B pour les vecteurs à droite. On a : $Q_A Y_A = Q_B X_B$ et les colonnes de ces matrices forment une base orthonormée de l'intersection recherchée.

L'algorithme pour le calcul des matrices B_i. Il est constitué des opérations suivantes :

1. Calcul d'une base orthonormée de $S^{(i,i+1)}\ \widetilde{C}_{i+1}^k$, comme décrit dans le paragraphe 5.1.1.2 : $Q_A = X'_a \in \mathcal{M}^{(n,l_A)}$, où on a introduit la notation l_A pour désigner le nombre de colonnes de Q_A.
2. Calcul d'une base orthonormée des colonnes de la matrice
$$\left[(S^{(i,j)}\ \widetilde{C}_j^k)_{j\in\{\text{domaines recouvrant } i \text{ et } i+1\}}\right].$$
Cette base, représentée par la matrice $Q_B \in \mathcal{M}^{(n,l_B)}$, est obtenue par une décomposition en valeurs singulières, comme on l'a fait pour Q_A au point précédent. Il est crucial d'utiliser le seuil ϵ défini dans l'étape locale (paragraphe 4.2.2) pour bien maintenir l'orthogonalité.
3. Calcul de la décomposition en valeurs singulières de $Q_A^t Q_B$. On garde les notations utilisées pour la présentation de l'algorithme de calcul de l'intersection de deux sous-espaces. Les vecteurs colonnes de Y_A doivent correspondre aux valeurs singulières de $Q_A^t Q_B$ strictement égales à 1. Dans des tests numériques préliminaires portant sur une molécule d'alcane, on a observé une grande sensibilité par rapport au seuil pris sur ces valeurs singulières, avec une perte de l'orthogonalité entre les blocs si on descendait par exemple à $1 - 10^{-8}$. En attendant des tests plus approfondis, on fixe prudemment ce seuil à $1 - 10^{-14}$.

On peut aussi reprendre l'ensemble de ce développement pour des groupes comprenant plus de deux domaines, pourvu qu'ils respectent les contraintes "1D" : au

sein du groupe, chaque domaine recouvre au plus deux domaines qui ne se recouvrent pas mutuellement. Par exemple, sur le schéma de droite de la figure 5.1, on pourrait traiter en un seul groupe les domaines 1, 2 et 3.

5.2 Implémentation

Nous décrivons, dans cette section, les choix qui ont été faits au cours de l'implémentation informatique de la version "2D/3D" de MDD. Les deux premières parties donnent le détail de la structuration des données en machine et de l'implémentation des calculs. Les parties 5.2.3 et 5.2.4 fournissent des informations plus synthétiques sur le code : estimations des besoins en mémoire et de la complexité pour une itération de MDD en fonction des paramètres de l'algorithme.

Une idée directrice dans les choix réalisés au cours du développement a été de minimiser la mémoire utilisée pour les calculs, dans le but de pouvoir maximiser le nombre de domaines sur un seul processeur et ainsi diminuer le rapport du temps de communication par rapport au temps de calcul. .

Pour les calculs "3D", la question est même de savoir si on pourra traiter *un seul* domaine sur un unique processeur. On peut en effet s'attendre à ce que la mémoire nécessaire pour un calcul "3D" soit importante, car le nombre de fonctions dans chaque domaine sera élevé. Estimons ce nombre par une simple extrapolation des résultats obtenus avec la version "1D". On a constaté lors des tests numériques portant sur les matrices de la famille $\mathcal{P}_3$ (polyéthylène) qu'on pouvait obtenir une approximation raisonnable de la densité (erreur absolue maximale de l'ordre de 10^{-4}) en limitant le recouvrement entre deux domaines à 5 monomères, soit 10 atomes de carbone (cf. figure 4.9). On peut donc s'attendre à ce que le recouvrement minimal à utiliser entre deux domaines pour un calcul "3D" (une structure diamant par exemple) corresponde à un cube de 10^3 atomes de carbone et pour un domaine complet à environ 2 à 5 fois ce nombre. Une base minimale comme la base de gaussiennes STO-3G utilise 5 fonctions par atome de carbone et conduirait alors à 10^4 fonctions de base par domaine. Le stockage de chaque matrice H_i (dense) représenterait de l'ordre de 10^8 valeurs ce qui occupe déjà presque 1 GB en mémoire. Ce nombre est une estimation basse, pour une base minimale et une hypothèse de localisation assez stricte au regard de résultats publiés par ailleurs : dans [49], pour la même molécule, les auteurs estiment qu'une orbitale localisée devrait se développer sur un nombre d'atomes de carbone allant de 14 à 58. Cette question sera reprise à la section 5.2.3.

Comme la version "1D", ce code a été écrit en *FORTRAN77*, avec un programme principal écrit en *langage C* pour pouvoir allouer automatiquement les différents tableaux en mémoire en fonction des caractéristiques du calcul. Le choix de ce langage a été principalement motivé par la possibilité de réutiliser des routines de calcul écrites pour la version "1D".

Pour toutes les opérations algébriques, on fait appel à la bibliothèque LAPACK, version 3.1.1.

5.2.1 Structure des données

Dans cette section, on décrit :
- la manière de définir les domaines dans le code, le recouvrement entre les domaines,
- le montage et le stockage en mémoire des matrices locales,
- le stockage des coefficients des orbitales moléculaires.

5.2.1.1 Définition des domaines

Rappelons que dans la méthode MDD, un domaine représente un ensemble de fonctions de base. On doit identifier les fonctions composant chaque domaine à chaque fois que l'on veut passer de la description globale du problème à une description locale, et inversement : d'abord pour construire les matrices locales, H_i et S_i (analogue de H_i pour la matrice S), à partir des matrices H et S du problème complet (cf section 5.2.1.4), puis en fin de calcul pour reconstruire la matrice densité à partir des coefficients des orbitales moléculaires construites sur chaque domaine. L'identification des domaines qui se recouvrent et la construction des matrices $S^{(ij)}$ sont des opérations qui utilisent également cette information (cf. section 5.2.1.3).

On parlera de numérotation locale pour représenter la numérotation des fonctions de base à l'intérieur d'un même domaine et de numérotation globale en référence au calcul complet.

Pour chaque domaine, une liste de fonctions (identifiées en numérotation globale) doit donc être conservée en mémoire pendant tout le calcul.

L'utilisateur du code ne doit pas donner explicitement la liste des fonctions composant chaque domaine. Pour la construire automatiquement, on a imposé *a priori* que la base soit composée d'orbitales atomiques et que les domaines soit parallélépipédiques. Trois fichiers doivent être renseignés :
- un fichier décrivant les coordonnées cartésiennes de chaque atome,
- un fichier de correspondance entre chaque fonction de base et l'atome qui lui est associé,
- un fichier donnant les bornes de chaque domaine.

Tout autre choix de type de base ou de forme de domaine peut-être considéré : l'impact sur le code est limité à la constitution de cette liste de fonctions composant chaque domaine.

La taille des domaines n'est pas nécessairement uniforme. On note n_i le nombre de fonctions du domaine i.

5.2.1.2 Matrices du problème

Les deux matrices H et S qui spécifient complètement le problème résolu par MDD sont lues dans des fichiers, puis stockées suivant le format creux le moins sophistiqué : on stocke pour tous les termes non nuls, l'indice de ligne, l'indice de colonne et la valeur. Ce format de stockage a été choisi car il est adapté à la structure des fichiers contenant les matrices pour nos tests numériques. Modifier ce format n'impacte que la construction des matrices locales (voir section 5.2.1.4) et la détermination du recouvrement entre domaines. Ces données ne sont plus conservées en mémoire dès que les matrices locales ont été construites une fois (voir 5.2.1.4).

5.2.1.3 Recouvrement entre les domaines

Rappelons que dans notre terminologie (voir chapitre 1, paragraphe 1.2.1), on dit que deux domaines *se recouvrent* dès qu'une fonction de base de l'un n'est pas orthogonale à une fonction de base de l'autre. Dans le cas où $S = I$, il suffit de regarder si deux domaines ont des fonctions en commun en numérotation globale. Dans le cas contraire, on identifie, à partir de la matrice S et de la liste des fonctions constituant chaque domaine, les couples de domaines (i, j) pour lesquels la matrice $S^{(ij)}$ (cf. formule (5.1)) contient des termes non nuls.

Pour chaque domaine i, on conserve en mémoire jusqu'à la fin du calcul la liste des domaines qui le recouvre. Cette liste est utilisée pour l'implémentation de l'étape locale et de l'étape globale.

5.2.1.4 Matrices des opérateurs locaux

Il s'agit d'une part des matrices H_i, S_i pour tous les domaines i et d'autre part des matrices $S^{(ij)}$ (cf. formule (5.1)), pour les couples de domaines (i, j) qui se recouvrent.

On adopte un stockage dense pour toutes ces matrices. Les matrices H_i sont effectivement pleines tant que les domaines ne sont pas très étendus. Les matrices S_i contiennent bien plus de termes nuls que H_i. Enfin, pour les matrices $S^{(ij)}$, on pourrait considérer un stockage creux (surtout quand on a une décomposition cartésienne, et que les domaines se recouvrent sur un coin) mais cette éventualité a été écartée dans un premier temps parce qu'il faudrait que ce stockage permette d'implémenter le produit de matrices $S^{(ij)}C_j$, fréquemment calculé, de manière performante, ce qui demande un développement conséquent.

Le principe de construction de ces matrices est très simple : extraction dans H et S (cf section 5.2.1.2) des lignes et des colonnes correspondant aux fonctions des domaines concernés.

Le plus simple est de conserver toutes les matrices locales en mémoire jusqu'à la fin du calcul.

Dans le cas où $S = I$, cela introduit une redondance dans le stockage : tous les termes des H_i correspondant à des zones de recouvrement entre domaines sont

stockés plusieurs fois, ce qui est compensé par le fait qu'on ne doit pas stocker les S_i et les $S^{(ij)}$. Par contre, dans le cas $S \neq I$, chacune des matrices locales subit une transformation spécifique sur chaque domaine, pour se ramener localement au cas $S = I$ (voir section 5.2.2.2). Il n'y a donc pas de redondance de stockage venant du recouvrement entre les domaines.

Stocker toutes les matrices locales en mémoire pendant tout le calcul peut représenter une part significative de la mémoire totale allouable sur chaque processeur, notamment pour un calcul "3D", si l'on considère l'estimation faite en introduction de la partie 5.2. Une solution est d'écrire sur disque (*out of core*) les matrices locales, puis de recharger les matrices locales nécessaires au moment de leur utilisation. Suivant le calculateur utilisé (notamment le temps de chargement en mémoire des données écrites sur disque), cette solution peut être plus avantageuse que la transformation globale pour ramener le problème au cas $S = I$.

On pourrait également envisager de recalculer les termes des matrices H_i, S_i et $S^{(ij)}$ à chaque fois que l'on passe au calcul du domaine i. Cette solution dépasse le cadre de ce travail mais pourra s'envisager quand on utilisera MDD au sein de la boucle SCF.

Signalons pour terminer deux simplifications apparaissant dans le cas $S = I$, si l'on conserve en mémoire la liste des fonctions de base communes entre deux domaines . Cela permet d'éviter le stockage des matrices $S^{(ij)}$ et de remplacer le calcul des produits matriciels de la forme $S^{(ij)}C_j$ par une simple extraction d'éléments dans le tableau où sont stockés les C_j.

5.2.1.5 Coefficients des orbitales moléculaires

Les n_i fonctions de base du domaine i sont utilisées pour m_i orbitales moléculaires. A l'itération k, le nombre exact de coefficient à stocker est donc :

$$\sum_{i=1}^{p} n_i m_i^k \ ,$$

où p est le nombre de domaines dans le calcul.

On réserve *a priori* en début de calcul une zone de mémoire pour le stockage des C_i, composée d'un nombre de valeurs égal à : $N \ \max_i\{n_i\}$. Ce choix est contestable dans le cas où les tailles de domaine n_i sont très hétérogènes : une partie de la mémoire réservée pour les C_i n'est jamais utilisée. Pour éviter cette perte, on pourrait implémenter une gestion dynamique du nombre total de coefficients, au niveau de l'étape d'échange :

- soit en redimensionnant l'espace mémoire alloué aux matrices C_i,
- soit en fixant *a priori* un nombre maximal d'éléments non nuls dans la matrice C, noté Nc_{max}, plus faible que : $N \ \max_i\{n_i\}$, et en ajoutant la contrainte suivante dans l'algorithme de calcul des m_i^k (étape locale) :

$$\sum_{i=1}^{p} n_i m_i \leq Nc_{max}.$$

5.2.2 Implémentation

5.2.2.1 Traitement de $S \neq I$

On connaît deux manières de compenser le fait que les fonctions de base ne sont pas orthogonales. La première possibilité consiste à transformer le problème complet pour se ramener au cas $S = I$ en calculant l'inverse des facteurs de Cholesky de la matrice S. Des algorithmes ont été récemment proposés pour effectuer cette transformation en $O(N)$ opérations dans le cadre des calculs de chimie computationnelle [43, 63, 69]. Dans la deuxième possibilité, on se ramène localement au cas $S = I$. Dans ce cas, le maintien de l'orthogonalité entre les blocs (orbitales de domaines différents) au cours de l'étape globale n'est pas garanti.

Les deux solutions sont implémentées dans le code.

Orthogonalisation globale de S. La matrice S étant définie positive, la manière usuelle de transformer le problème aux valeurs propres généralisé (3.2) en un problème aux valeurs propres standard consiste à remplacer H par $L^{-1}HL^{-t}$, L étant le facteur de Cholesky triangulaire inférieur de S [114]. Evidemment, cette opération n'est acceptable dans la perspective d'un calcul d'ordre N que si on utilise l'un des algorithmes cités plus haut. Le code informatique de l'un d'eux [69] est fournie par les auteurs et utilisable dans cette version de MDD.

Orthogonalisation locale de S. Dans MDD, on peut aussi faire cette transformation localement sur chaque bloc au début du calcul :

$$S_i = L_i L_i^t \tag{5.7}$$

$$H_i \text{ est remplacée par : } \quad L_i^{-1} H_i L_i^{-t} \tag{5.8}$$

$$S^{(ij)} \text{ est remplacée par : } \quad L_i^{-1} S^{(ij)} L_j^{-t}. \tag{5.9}$$

Cela revient à faire le changement de variable : $C_i \rightarrow L_i^t C_i$, de manière à avoir $S_i = I$. Une fois ces transformations faites, le code calcule en pratique les blocs : $L_i^t C_i$.

On peut ainsi traiter chaque problème local comme si on avait $S = I$, donc comme présenté à la section 4.2.2, p. 67, en remplaçant toutefois T par $S^{(ij)}$ dans les équations (4.3)-(4.5).

Rappelons que la propriété :

$$S^{(i,j)} \, [S^{(i,j)}]^t S^{(i,j)} = S^{(i,j)} \tag{5.10}$$

doit être vérifiée pour que le produit scalaire $C_i(U)^t \, S^{(i,j)} \, C_j(U)$ reste nul quand on fait l'étape globale (5.2) sur les blocs i et j. Cette propriété est en général fausse si $S^{(ij)}$ est une matrice extraite d'une matrice S symétrique définie positive quelconque. Elle est par exemple fausse pour les matrices S correspondant à la base gaussienne STO3G, dans le cas des molécules de la famille $\mathcal{P}_3$. En revanche, on observe que

cette propriété est vérifiée dans nos tests numériques quand on utilise les matrices transformées par (5.9).

D'une manière générale, on ne sait pas encore caractériser les cas dans lesquels (5.10) est vérifiée, même après les transformations (5.9) et c'est une question qui sera approfondie après la thèse. Des premiers tests numériques nous ont donné les résultats qualitatifs suivants :

- pour les molécules des familles $\mathcal{P}_i$, (5.10) est vérifiée *pour les matrices obtenues après la transformation (5.9)*, à condition que le recouvrement entre les domaines i et j soit assez grand,
- dans certains cas où L_i^{-1} est pleine, (5.10) n'est pas vérifiée.

5.2.2.2 Initialisation des orbitales

On suit exactement la procédure décrite dans le paragraphe 3.5.1.

5.2.2.3 Etape locale

Les opérations qui doivent être effectuées pour la résolution de l'étape locale à l'itération $k+1$ sont :

- pour chaque domaine i, résoudre le problème local (4.3) et stocker la solution $C_i \in \mathcal{M}^{(n_i,\tilde{m}_i)}$, $\tilde{m}_i > m_i^k$,
- classer les énergies de tous les vecteurs colonnes des C_i, choisir les N plus petites et en déduire le nombre d'orbitales par domaine, m_i^{k+1} (étape d'échange).

Adopter un découpage aussi strict entre la résolution des problèmes locaux et l'étape d'échange dans l'implémentation serait très pénalisant car il faudrait stocker les $\sum_i n_i\tilde{m}_i$ valeurs des vecteurs colonnes en attente de classement. Il faut mêler les deux opérations.

Par ailleurs, la grande indépendance entre les problèmes locaux peut être exploitée en distribuant le calcul sur plusieurs processeurs. Cela revient à partitionner l'ensemble des domaines : un élément de cette partition (qu'on nomme une *couleur*) regroupe des domaines dont les problèmes locaux sont indépendants. On peut mettre deux domaines dans la même couleur dès qu'ils ne se recouvrent pas. Cette condition étant également nécessaire, elle définit le nombre minimal de couleurs. Par exemple, pour une décomposition comme celle du schéma de droite de la figure 5.1, on doit définir 4 couleurs, contre seulement 2 sur le schéma de gauche (voir figure 5.3 pour une représentation des couleurs sur l'exemple de la figure 5.1).

Ce partitionnement de l'étape locale est bénéfique indépendamment du fait que le calcul est mono- ou multi-processeurs, car cela permet :

- d'utiliser les solutions calculées à la fin d'une couleur au cours du calcul des couleurs suivantes, ce qui conduit à une procédure de type Gauss-Seidel sur les blocs, avec une accélération de la convergence de MDD,
- de réaliser l'échange dès la fin du traitement de chaque couleur, donc de limiter le stockage des solutions C_i en attente de classement. Le classement porte alors sur tous les domaines de la couleur qui vient d'être traitée et ceux des

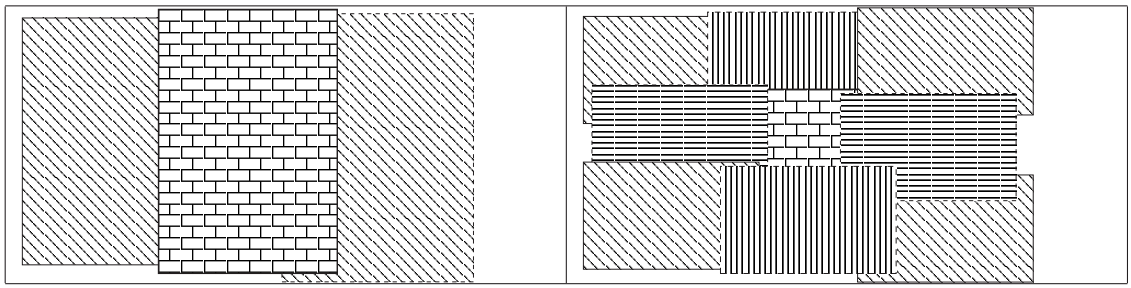


FIG. 5.3 – Les couleurs dans l'étape locale pour la décomposition de la figure 5.1. Pour clarifier la présentation, les atomes ont été omis.

couleurs qui ont déjà été traitées. Pour ces derniers, le nombre m_i ne peut que diminuer puisqu'on n'a gardé en mémoire que la partie "utile" de la solution C_i du problème local.

Dans la version séquentielle du code, on a donc introduit un regroupement des blocs par couleurs. Aux avantages listés ci-dessus s'ajoute la possibilité de tester, dans un calcul monoprocesseur, l'effet de l'ordre de traitement des blocs qui serait adopté au cours d'un calcul multiprocesseur.

Dans cette version séquentielle, on pourrait même envisager de diminuer encore plus le poids du stockage des solutions C_i en attente de classement en faisant l'étape d'échange à la fin de la résolution de chaque domaine. Il n'y aurait aucun effet sur la solution calculée. Ceci est encore vrai dans un calcul parallèle où chaque processeur calcule plusieurs blocs de la même couleur. Mais cela induirait un retard dans le calcul, puisqu'on devrait synchroniser les processeurs et communiquer les énergies à classer à la fin du traitement de chaque bloc. Il y a dans ce cas un compromis à trouver qui dépend de la mémoire disponible sur chaque processeur et de l'efficacité du réseau. Pour une machine comme BG/L où il y a peu de mémoire sur chaque processeur et un réseau de communication performant, il sera sans doute avantageux de faire l'étape d'échange à la fin de chaque problème local et non pas à la fin de chaque couleur.

On revient sur l'implémentation de l'étape d'échange dans la section C.1.2.

Avant de terminer cette section par quelques détails d'implémentation, signalons qu'il peut arriver que le nombre $\tilde{m}_i$ de solutions du problème local sur le domaine i soit inférieur à m_i^k, le nombre de vecteurs dans le bloc C_i^k avant le début de l'étape locale. C'est le cas notamment au cours des premières itérations de MDD et si le recouvrement entre les domaines est important ce qui a pour effet de conduire à un sous-espace $\mathcal{V}$ (introduit à l'équation (4.4), p. 67) de petite taille. Ce déficit d'orbitales sur un domaine peut être rattrapé par les autres domaines de la même couleur au cours de l'échange, puis par les domaines des couleurs suivantes. La persistance d'un déficit d'orbitales à la fin de la dernière couleur traduit une répartition inadaptée des domaines. Le calcul s'arrête dans ce cas.

Résolution d'un problème local. La décomposition en valeurs singulières pour le calcul du sous-espace $\mathcal{V}$ est réalisée par une méthode directe. En fonction de la mémoire disponible, on utilise la méthode QR (*dgesvd*) ou la méthode Divide-and-Conquer (*dgesdd*). Cette dernière est moins précise : l'erreur sur chaque valeur singulière peut atteindre $O(\epsilon_{mach})\sigma_1$, σ_1 étant la plus grande valeur singulière et ϵ_{mach} la précision machine [95]. Cela ne nous affecte pas ici, car la plus grande valeur singulière est toujours de l'ordre de 1 car le recouvrement entre deux blocs voisins est assez grand pour contenir tous les coefficients d'au moins une orbitale.

Une méthode directe est également utilisée pour la diagonalisation de la matrice $V^t\ H_i\ V$. On exclut les méthodes itératives car on veut tous les vecteurs propres et cette matrice est dense. Le choix est limité là encore aux méthodes QR (*dsyev*) et Divide-and-Conquer (*dsyevd*), car les matrices H_i peuvent avoir des *clusters* de valeurs propres très proches. On se permet un stockage dense plein pour la matrice $V^tH_iV \in \mathcal{M}^{(\tilde{m}_i,\tilde{m}_i)}$ alors que cette dernière est symétrique pour les raisons suivantes :

- cela n'induit aucun surcoût en mémoire, puisqu'on doit récupèrer tous les vecteurs propres,
- la version "stockage dense" de la routine de diagonalisation de LAPACK est plus rapide que la version "stockage symétrique" car elle fait une plus grande utilisation des routines BLAS de niveau 3 (voir [117], chapitre 6, "Coding Techniques").

5.2.2.4 Etape globale

L'étape globale consiste en une boucle sur les couples de domaines qui se recouvrent. Pour chaque couple de domaines (i, j), on doit :

1. calculer les matrices B_i et B_j apparaissant dans les formules (5.3)-(5.4),
2. minimiser :

$$\begin{aligned} f(U) &= \mathrm{Tr}\left(H_iC_i(U)\left[[C_i(U)]^t\,C_i(U)\right]^{-1}[C_i(U)]^t\right) \\ &\quad +\mathrm{Tr}\left(H_jC_j(U)\left[[C_j(U)]^t\,C_j(U)\right]^{-1}[C_j(U)]^t\right) \end{aligned}$$

Comme dans l'étape locale, on regroupe les couples de domaines pour lesquels les formules (5.3) et (5.4) sont indépendantes.

On peut mettre deux couples dans la même couleur si aucun des domaines les composant ne se recouvre. Ce n'est pas une condition nécessaire : par exemple, sur le schéma de décomposition de droite de la figure 5.1, on peut traiter simultanément les couples $(1, 4)$ et $(2, 3)$. Il est difficile de donner une condition suffisante générale pour que deux couples soient dans la même couleur, car cela dépend de la position relative des domaines entre eux. Dans cette première version du code, le regroupement des couples de domaines en couleurs est réalisé par l'utilisateur.

Le calcul des matrices B_i est réalisé comme décrit à la section 5.1.2. Ici encore, on a utilisé une méthode directe pour les différentes décompositions en valeurs singulières.

<table>
<tr><td colspan="5">Calcul MDD</td></tr>
<tr><td colspan="5">Une itération de MDD</td></tr>
<tr><td colspan="2">Etape locale</td><td></td><td colspan="2">Etape globale</td></tr>
<tr><td colspan="2">Problèmes locaux sur une couleur</td><td></td><td colspan="2">Problèmes globaux sur une couleur</td></tr>
<tr><td colspan="2">Problème local sur un domaine</td><td></td><td colspan="2">Problème global sur deux domaines voisins</td></tr>
<tr><td>Calcul du sous-espace V_ϵ</td><td></td><td>Diagonalisation de $V^t H_i V$</td><td>Calcul des matrices B_i</td><td>Minimisation de la fonctionnelle</td></tr>
</table>

TAB. 5.1 – Représentation partielle de la hiérarchie des opérations dans l'implémentation de MDD.

Pour la minimisation, on suit exactement l'algorithme de la version "1D", décrit dans la partie 3.3. Les formules du hessien et du gradient de f sont détaillées dans l'annexe B, ainsi que la manière dont elles sont implémentées dans le code.

5.2.3 Estimation de la mémoire totale

Le but de cette section est de donner une expression des termes dominants de la quantité de mémoire nécessaire pour un calcul MDD.

On a cherché à minimiser l'espace mémoire nécessaire en introduisant une gestion hiérarchique de la mémoire[3]. On peut représenter le calcul par une hiérarchie d'opérations de complexité décroissante (voir tableau 5.1) :

- au niveau le plus haut, le calcul de la densité par MDD,
- au niveau immédiatement inférieur, une opération plus simple : une itération de MDD,
- puis, au même niveau, l'étape locale et l'étape globale, chacune d'elles se décomposant en sous-opérations, etc.

Considérons une opération réalisée au niveau l de cette hiérarchie. Cette opération est représentée par une case dans le tableau 5.1, notée $\boxed{\text{op}}$, ou par un sous-programme bien identifié, *spgm*, dans le code FORTRAN. Dans ce sous-programme, on ne stocke explicitement dans des tableaux nommés que les données qui sont utilisées :

- soit dans le sous-programme lui-même,
- soit dans *toutes* les opérations du niveau $l+1$ (toutes les cases liées à la case $\boxed{\text{op}}$ dans le tableau, tous les sous-programmes appelés par *spgm* dans le code FORTRAN)

De plus, on réserve un tableau de travail qui servira successivement pour les stockages internes à chacune des opérations du niveau $l+1$. C'est l'opération de niveau $l+1$ la plus consommatrice de mémoire qui déterminera la taille du tableau de travail au niveau l.

[3]Le langage FORTRAN77 ne permet pas de faire simplement une réelle gestion dynamique de la mémoire.

Commençons au premier niveau. Seuls les blocs C_i et les matrices locales sont utilisés pendant toutes les itérations de MDD, et, à l'intérieur de chaque itération, à la fois par l'étape locale et l'étape globale. La mémoire totale utilisée par le code peut donc être divisée en trois parties : le stockage des données globales du problème (matrices locales, blocs C_i solutions) et les stockages temporaires pour la résolution d'une étape locale et d'une étape globale. Seule la plus grande de ces deux dernières parties sera déterminante et s'ajoutera à la première.

Le détail des calculs est donné dans l'annexe C, et on n'en donne ici que les conclusions.

Pour simplifier les expressions, on se place dans le cas où la taille des domaines est uniforme ($n_i = n$), ainsi que le nombre d'orbitales par domaine ($m_i = m$). On note :

- N le nombre total d'orbitales à calculer,
- N_b le nombre total de fonctions de base dans le calcul,
- p le nombre de domaines, $N = pm$
- $\overline{p}$ le nombre de couples de domaines qui se recouvrent. Pour une décomposition cartésienne comme présentée à la figure 5.1, on a : $\overline{p} \approx \alpha p$ avec $\alpha = 1$ en 1D, $\alpha = 4$ en 2D et $\alpha = 13$ en 3D.
- M_1 la somme des orbitales calculées sur tous les domaines voisins d'un domaine donné. Dans les cas où les m_i sont tous égaux à m, M_1 est égal à m fois le nombre de domaines voisins. Dans le cas d'une décomposition cartésienne, on a : $M_1 = 2\alpha m$.

Dans ces paramètres, N représente le *grand* terme par excellence : c'est le terme qu'on veut prendre aussi grand que possible. Avec N, p et $\overline{p}$ doivent aussi être considérés comme des grands termes. De plus, le poids de $\overline{p}$ augmente avec la dimensionnalité du système.

En bases localisées, bien que le nombre m_i d'orbitales calculées par domaine puisse varier de manière importante même quand n est constant, n et m sont intrinsèquement reliés par le nombre de fonctions de bases par électron. Dans l'esprit de la décomposition de domaine, m et n sont des petits termes devant N, puisqu'ils représentent des quantités locales. Il faut cependant nuancer. Tout d'abord, en pratique, on n'aura effectivement cette relation de domination que si le nombre de domaines p est grand et d'autant plus que le rapport n/m l'est aussi (influence de la qualité de la base utilisée). De plus, n et m augmentent tous les deux fortement avec la dimensionnalité du système. On a par exemple estimé en introduction de la section 5.2.2 que n pourrait atteindre facilement 10^4 pour des simulations "3D". Enfin, n augmente également avec la délocalisation des électrons. Dans les développements qui vont suivre, on aura donc tendance à considérer que n et m peuvent également être grands.

Si on garde toutes les matrices locales en mémoire, le terme dominant est Nn, la mémoire totale est bien d'ordre N. Le préfacteur est potentiellement important, d'où l'intérêt de l'option "stockage des matrices locales sur disque". En utilisant les écritures sur disques, on fait disparaître ce terme et la mémoire est alors, dans le

cas d'une décomposition cartésienne :

$$n^2 (p + \alpha p + 8 + 2\alpha) + (n+1)\, p_{coul} (m_{max} - m_{min})$$

5.2.4 Complexité des étapes

Le détail de la complexité de chaque étape est présenté en annexe C. On reprend ici les conclusions de ce calcul.

Chacune des deux étapes est en $O(n^3)$. La phase d'initialisation complète (*i.e.* la phase d'initialisation décrite en 5.2.2.2) représente un peu plus qu'une itération de MDD.

Les estimations de l'annexe C montrent que la résolution d'un problème local coûte à peu près le même nombre d'opérations que la résolution d'un problème global[4], mais l'étape globale complète représente sensiblement plus d'opérations puisqu'il faut résoudre un problème global pour tous les couples de domaines voisins. Si on se réfère encore une fois à une répartition cartésienne des domaines, la relation $\overline{p} \approx \alpha p$ avec $\alpha = 1$ en 1D, $\alpha = 4$ en 2D et $\alpha = 13$ en 3D, donne l'évolution du poids relatif des étapes locales et globales dans le nombre d'opérations. Dans un calcul "3D", l'étape globale représenterait ainsi plus de 90% du nombre d'opérations.

[4] C'est ce qu'on observe effectivement sur la répartition du temps CPU dans les tests numériques "1D".

Chapitre 6

Conclusion générale

Les modèles Hartree-Fock et DFT permettent de calculer la structure électronique de systèmes pouvant comporter jusqu'à 10^4 atomes avec une précision suffisante pour décrire l'état fondamental. La taille des simulations est limitée par une étape de complexité $O(N^3)$, où N est le nombre d'électrons du système. La méthode MDD, introduite dans [11], a pour objectif de remplacer cette étape par un algorithme de complexité linéaire basé sur une approche de décomposition de domaine.

Dans ce travail, le développement de la méthode MDD (*Multilevel Decomposition Domain*) a été poursuivi et des améliorations ont été obtenues grâce à un meilleur choix des algorithmes dans chaque étape :

- la convergence vers la matrice densité de référence est plus précise,
- le temps de calcul consommé par l'étape globale a été fortement réduit.

Le choix des algorithmes dans chaque étape n'est pas encore définitif et on a indiqué en conclusion du chapitre 4 quelques pistes pour améliorer le traitement de l'étape locale.

L'algorithme et son implémentation informatique ont été généralisés aux cas où les atomes sont répartis de manière homogène dans les trois directions de l'espace. Des tests numériques sont en cours pour cette version.

Les tests numériques ne sont présentés que pour la version "1D", avec une matrice de Fock fixe. Cela correspond au problème résolu au cours d'une étape de la boucle SCF d'un modèle Hartree-Fock/DFT ou au problème à résoudre dans un modèle semi-empirique (modèle de Hückel, de liaisons fortes). La scalabilité de la méthode a été étudiée par rapport au nombre d'atomes, l'étude de la scalabilité par rapport à la finesse de la base de discrétisation étant remise à des travaux ultérieurs.

Les tests sur machines parallèles ont montré que la version "1D" MDD permet la simulation de cas où le nombre d'atomes est très important.

Les paragraphes suivants développent un certain nombre de questions liées à la généralisation de l'utilisation de la méthode MDD au sein d'un calcul complet.

Interaction de MDD avec une boucle SCF. La précision obtenue sur la matrice densité varie en fonction du nombre de degrés de liberté pris en compte. Sur

les cas test correspondant au polyéthylène on a montré à la fin de chapitre 4 que l'erreur sur les termes de la matrice densité peut aller de 10^{-2} à 10^{-6} en fonction de la taille du recouvrement.

Augmenter la taille du recouvrement ou le nombre d'itérations pour atteindre une précision importante sur la matrice densité a un coût important en temps de calcul (voir figures 4.11 et 4.10). Cependant, si l'on considère l'algorithme MDD comme une étape élémentaire dans la boucle SCF, il n'est probablement pas nécessaire d'atteindre une grande précision sur la matrice densité à chaque itération. Dans les premières itérations, il suffit d'assurer la convergence de la boucle de plus haut niveau[1], la boucle SCF. Il serait donc intéressant d'étudier la convergence de la boucle SCF en fonction de perturbations introduites par un calcul approximatif de la matrice densité.

De plus, quand on passe à une nouvelle itération de la boucle SCF, la perturbation de la matrice H est faible (surtout dans les dernières itérations) et l'initialisation de MDD par la solution obtenue à l'étape précédente devrait permettre une convergence beaucoup plus rapide.

Ces questions sont communes à toutes les méthodes variationnelles de calcul de la densité (voir annexe A). De manière un peu plus spécifique à MDD (mais ce qui suit peu également s'appliquer à la méthode *Orbital Minimisation*), on pourrait aussi considérer les deux boucles imbriquées (boucle SCF et boucle de convergence dans MDD) de manière moins cloisonnée. On pourrait en effet limiter le calcul de la nouvelle matrice densité à certaines zones du système en fonction de l'avancement de la convergence de la boucle SCF. Dans un deuxième temps, on pourrait même songer à ne pas recalculer l'opérateur complet mais seulement pour les domaines pour lesquels la convergence n'est pas obtenue et faire la boucle SCF par domaines. Cela reviendrait à permuter la boucle sur les domaines et la boucle sur l'opérateur (boucle SCF).

Indicateur d'erreur. Dans les tests numériques présentés dans ce travail, la précision de la solution calculée par MDD est mesurée par comparaison à la solution de référence calculée par diagonalisation de H. Pour réaliser des calculs réels, il faudra bien sûr pouvoir estimer la précision de la solution sans avoir recours à cette solution de référence.

Une première piste est d'utiliser les relations d'idempotence de D et de commutation de D avec H. Cependant, ces deux relations ne caractérisent pas à elles seules la matrice densité associée à la matrice H. Lorsque la relation de commutation est vérifiée, les sous-espaces invariants du projecteur D sont aussi des sous-espaces invariants de H, mais on n'est pas assuré que le sous-espace image de D corresponde effectivement au sous-espace associé aux N plus petites valeurs propres de H.

[1]Dans la dernière itération, la précision requise sur la matrice densité dépend du type de propriétés calculées.

Annexe A

Présentation succincte des méthodes d'ordre N

Le contenu de ce chapitre s'inspire fortement de [1] et de la thèse de M. Barrault, avec des compléments issus de la lecture de sources plus récentes.

Ce chapitre est consacré à la description de méthodes concurrentes de MDD, qui ont donc pour objet de calculer, au cours d'une itération de la boucle SCF, la matrice densité à partir d'une matrice de Fock F donnée, avec un nombre d'opérations croissant linéairement avec la taille de la molécule simulée. On exclut donc de cette présentation les méthodes qui, dans le cadre d'une minimisation directe, font une sous-étape linéaire en $O(N)$ opérations. Dans cette catégorie, citons simplement [64] et [78].

Les méthodes qui nous intéressent ici sont toutes basées sur une caractérisation de la matrice densité, D, qui n'utilise pas de manière explicite les vecteurs propres de la matrice de Fock. On peut par exemple définir D comme la solution d'un problème de minimisation sous contrainte posé sur les orbitales moléculaires :

$$D = C\,C^t \text{ avec : } C = \text{arginf}\left\{\text{Tr}\left(F\,\tilde{C}\,\tilde{C}^t\right), \tilde{C} \in \mathcal{M}^{N_b,N}, \tilde{C}^t S \tilde{C} = I_N\right\}, \tag{A.1}$$

ou, de manière équivalente sur la matrice densité elle-même :

$$D = \text{arginf}\left\{\text{Tr}\left(F\,\tilde{D}\right), \tilde{D} \in \mathcal{M}_S^{N_b}, \tilde{D} S \tilde{D} = \tilde{D}, \text{Tr}\left(S\tilde{D}\right) = N\right\}, \tag{A.2}$$

ou encore comme une fonction de la matrice de Fock :

$$D = \mathcal{H}(F) \;\; \forall\, q \text{ tel que } \left| \begin{array}{ll} \mathcal{H}(\epsilon_i) = 1 & \forall\, 1 \leq i \leq N \\ \mathcal{H}(\epsilon_i) = 0 & \forall\, N+1 \leq i \leq N_b \end{array} \right. \tag{A.3}$$

Les méthodes décrites dans ce chapitre exploitent le principe de localité [45], qui est vérifié pour les isolants et qui se traduit numériquement par les deux propriétés suivantes :

- la matrice densité est creuse,
- il existe des orbitales moléculaires localisées.

Ces méthodes sont donc conçues pour être utilisées avec des bases de discrétisation localisées. Pour simplifier la présentation, on suppose de plus que la base de discrétisation utilisée est orthogonale : $S = I$. Ce n'est généralement pas le cas en pratique. Pour chaque méthode, des adaptations plus ou moins efficaces au cas $S \neq I$ ont été développées et elles seront décrites en fin de chapitre après l'exposé des méthodes.

On ne traite pas de manière exhaustive toutes les méthodes publiées, mais uniquement les plus utilisées d'entre elles, que l'on peut répartir en trois grandes classes :

- *les méthodes variationnelles*, qui sont basées sur (A.1) ou (A.2),
- *les méthodes de projection*, qui calculent la densité par la formule (A.3) en choisissant pour $\mathcal{H}$ des fonctions polynômiales ou rationnelles.
- les méthodes de type *décomposition de domaine*, qui consistent à calculer la densité en diagonalisant des représentations locales de F.

La méthode MDD est à la fois une méthode variationnelle et une méthode de décomposition de domaine.

A.1 Les méthodes variationnelles

Dans ces méthodes on utilise une caractérisation variationnelle de la densité, par le problème de minimisation sous contraintes (A.1) ou (A.2). Dans les deux cas, le nombre de contraintes est en N^2 et imposer les contraintes est une opération en N^α avec $\alpha = 2$ ou 3. Ces méthodes sont construites en 2 temps :

1. Pénalisation exacte des contraintes dans (A.1) ou (A.2) puisque l'imposition explicite des contraintes est trop coûteuse. L'élimination de la contrainte d'orthonormalité des orbitales (ou idempotence de la matrice densité) est assez complexe ; elle est détaillée dans les paragraphes suivants. Dans le cas de (A.2), la contrainte sur le nombre d'orbitales est éliminée de manière assez simple par l'introduction du multiplicateur de Lagrange associé μ et le remplacement de F par $F - \mu I$.

2. Formulation d'un algorithme de résolution du problème sans contrainte ainsi obtenu, de complexité $O(N^3)$ pour F et D quelconque, mais dans lequel on pourra exploiter facilement les conséquences du principe de localité pour se ramener à un algorithme d'ordre N. On impose *a priori* une structure creuse aux matrices D ou C, ce qui introduit une erreur. L'approche étant variationnelle, d'une part on est sûr d'être toujours au dessus du minimum recherché, et d'autre part l'erreur introduite peut être rendue petite en jouant sur le paramètre de localisation.

A.1.1 Minimisation sur les orbitales

Les développements basés sur (A.1) conduisent à la méthode *Orbital Minimisation* (OM). Cette méthode est implémentée dans le code SIESTA [91]. Dans ce paragraphe, on décrit d'abord le principe de la méthode, puis ses limitations.

On peut orthonormaliser un ensemble de N vecteurs quelconques (non liés) $\overline{C}$ à partir de sa matrice de Gram $\Sigma = \overline{C}^t\overline{C}$: les vecteurs colonnes de $\overline{C}\,\Sigma^{-1/2}$ sont orthogonaux.

On peut donc éliminer la contrainte d'orthonormalité de (A.1) en remplaçant la fonctionnelle par

$$\mathrm{Tr}\left(F\,\tilde{C}\,\Sigma^{-1}\tilde{C}^t\right) \text{ avec : } \Sigma = \tilde{C}^t\,\tilde{C}. \tag{A.4}$$

Pour obtenir un algorithme d'ordre N, deux approximations sont alors introduites :

- pour effectuer un calcul rapide de la matrice Σ^{-1}, cette dernière est approchée par : $2\,I - \Sigma$, ce qui revient à tronquer à l'ordre 1 le développement :

$$\Sigma^{-1} = (I - (I - \Sigma))^{-1} = \sum_{k=0}^{+\infty}(I - \Sigma)^k$$

  Si on remplace F par $F - \mu I$ avec μ supérieur à la plus grande valeur propre de F, la fonctionnelle obtenue après cette première approximation n'admet qu'un unique minimum, qui coïncide avec la solution de (A.1).
- les orbitales moléculaires sont localisées *a priori* dans des domaines de l'espace réel, ce qui revient à imposer une structure creuse à C. Si la localisation est trop stricte, la solution approchée ne vérifie pas la contrainte $\mathrm{Tr}(CC^t) = N$. La charge électronique totale ρ n'est donc pas conservée, ce qui peut être source de problème à l'intérieur de la boucle SCF [30].

Les problèmes suivants, liés à l'introduction de la localisation, sont beaucoup plus gênants en pratique :

- la fonctionnelle minimisée admet plusieurs minima locaux qui bloquent les itérés loin de la solution si l'algorithme n'est pas soigneusement initialisé [54],
- on observe des problèmes de conditionnement du problème de minimisation [36].

Récemment, une généralisation de la méthode OM a été proposée par Tsuchida [73, 74] et pourrait remédier aux difficultés mentionnées ci-dessus. La méthode *Augmented Orbital Minimisation* (AOM) consiste à renforcer l'orthonormalité des orbitales moléculaires sans faire d'orthonormalisation explicite et tout en conservant la localisation. Pour ce faire, le calcul tient compte d'un ensemble d'orbitales additionnelles dans chaque région de localisation (typiquement des orbitales de Wannier fortement localisées). Le support de ces fonctions additionnelles est beaucoup

plus petit que celui des orbitales moléculaires. Les orbitales moléculaires d'une région de localisation sont forcées à être orthogonales aux fonctions additionnelles des autres régions. Comme ces dernières ont un support très petit, cette orthogonalisation représente un faible nombre d'opérations supplémentaires. Cette méthode a été mise en œuvre dans le cadre d'une discrétisation par éléments finis. Aucune analyse détaillée de cette approche n'a encore été présentée, mais des tests numériques sur différentes molécules montrent que cette approche peut apporter un gain numérique significatif.

A.1.2 Minimisation sur la densité

Mener sur le problème (A.2) une démarche analogue à celle qui a été décrite au paragraphe précédent conduit à la *Density Minimisation Method* (DMM). Pour supprimer la contrainte d'idempotence, la fonctionnelle $\mathrm{Tr}\left(H\ \tilde{D}\right)$ est remplacée par $\mathrm{Tr}\left(H\ (\tilde{D} + g(\tilde{D}))\right)$, où g est une fonction polynôme vérifiant les deux propriétés suivantes :

- $g(D) = 0$ si $D^2 = D$,
- $g'(D) = -I$ si $D^2 = D$.

Ces deux propriétés suffisent pour que le minimum de (A.2) soit également un minimum de la nouvelle fonctionnelle.

Plusieurs variantes de la méthode sont obtenues pour différents choix de g. Dans le cas où $g(D) = 3\,D^2 - 2\,D^3 - D$, on peut montrer [1] que la nouvelle fonctionnelle n'est pas bornée inférieurement mais admet un unique minimum local qui est la solution du problème de minimisation initial.

Numériquement, cette fonctionnelle est minimisée par une méthode de gradient conjugué, avec une recherche linéaire exacte. Le gradient de la fonctionnelle est un produit de matrices.

Un algorithme d'ordre N est obtenu pour la minimisation de cette nouvelle fonctionnelle en imposant une structure creuse à la matrice densité, ce qui permet d'effectuer les produits de matrices en $O(N)$ opérations. Comme pour la méthode OM, cette troncature peut introduire une erreur sur la charge électronique [55]. En pratique, cette erreur est corrigée par une projection sur la contrainte $\mathrm{Tr}D = N$ [50]. En revanche, la troncature n'introduit pas de minima locaux et cette méthode se montre donc en pratique bien plus efficace que la méthode OM. La méthode DMM est implémentée dans les codes CONQUEST [83] et ONETEP [89].

A.2 Les méthodes de projection

Ce sont les méthodes basées sur des formules du type (A.3).

A.2.1 Approximation de l'opérateur de Fermi

La matrice densité peut s'écrire formellement à partir de la matrice de Fock F : $D = f_\mu(F)$, f_μ étant une fonction définie de $\mathbb{R}$ dans $\mathbb{R}$:

$$f_\mu(x) = \Theta(\mu - x)$$

où Θ représente la fonction de Heaviside et μ un réel compris entre la N-ème et la $(N+1)$-ème valeur propre de F. $f_\mu(F)$ est appelé *opérateur de Fermi*.

Le principe des méthodes de type FOE (*Fermi Operator Expansion*) et FOP (*Fermi Operator Projection*) est de remplacer la fonction f_μ par une approximation polynomiale ou rationnelle. Les algorithmes se ramènent donc à des suites de produits de matrices de complexité $O(N^3)$. On se ramène à une complexité linéaire par troncature des produits de matrices comme cela sera détaillé dans le paragraphe suivant.

Les différentes variantes correspondent à différentes représentations analytiques de l'opérateur de Fermi [10] ou à différents types d'approximations [21, 22, 36]. Les performances de ces méthodes sont directement reliées à la localisation de la singularité de f_μ par rapport au spectre de la matrice de Fock. Une analyse mathématique de cette question a récemment été publiée [101].

A.2.2 Méthode de purification de la densité

Cette méthode a été introduite dans [57]. Considérons une matrice F dont on connaît quatre valeurs propres : la plus petite, la plus grande, la N-ième et la $(N+1)$-ième, de sorte que l'on peut construire, par une transformation linéaire, une matrice F' ayant les mêmes vecteurs propres que F et dont les valeurs propres sont comprises entre 0 et 1, les N plus petites étant inférieures à $1/2$.

On définit la fonction f de $\mathbb{R}$ dans $\mathbb{R}$ par : $f(x) = 3x^2 - 2x^3$. Si on applique itérativement la fonction f à une valeur propre de F', on obtient une suite qui converge vers 0 ou vers 1 avec une vitesse de convergence quadratique. Le principe de la méthode de purification de la densité est d'appliquer itérativement la fonction f à la matrice F'. C'est donc un algorithme de point fixe appliqué à la matrice F'. La matrice F' a été construite pour que la suite de matrices obtenue, D_n, converge bien vers la matrice densité. Le nombre d'opérations induit par cet algorithme est équivalent à quelques produits de matrices soit $O(N^3)$.

En pratique, les itérations sont adaptées pour éviter le calcul *a priori* des N-ième et $(N+1)$-ième valeurs propres de F. De plus, comme dans les méthodes précédentes, on utilise le principe de localité pour se ramener à une méthode d'ordre N. La matrice F' ayant $O(N)$ termes non nuls, chaque produit de matrices au cours d'une itération peut être effectué en $O(N)$ opérations. Le nombre de termes non nuls dans les matrices F'^2, F'^3 est plus élevé que dans F' et après un certain nombre d'itérations les matrices D_n n'ont plus aucun terme non nul. Pour atteindre la complexité linéaire

de l'algorithme, il faut imposer à chaque itéré d'avoir la structure creuse que l'on attend pour la matrice densité.

On introduit donc une perturbation à chaque itération de l'algorithme de point fixe. L'accumulation de ces perturbations, si elle n'est pas contrôlée, finit par détruire les propriétés théoriques de convergence. Par exemple, si, au cours de l'itération n, la perturbation conduit la N-ième et la $(N+1)$-ième valeur propre de D_n à se croiser, la suite ne convergera pas vers le bon projecteur. L'algorithme peut même diverger si la perturbation produit une matrice D_n dont une valeur propre est hors de l'intervalle $[-1/2, 3/2]$, qui est le domaine de convergence des méthodes de point fixe produites à partir de la fonction f.

Dans [62], un contrôle des perturbations est introduit sur la base de deux résultats classiques de perturbation des valeurs propres et des sous-espaces propres d'une matrice symétrique réelle. A l'itération n, la norme euclidienne de la perturbation, $||E||_2$, doit être inférieure à l'écart entre la N-ième et la $(N+1)$-ième valeur propre de la matrice D_n, ξ, et l'écart induit par cette perturbation sur le sous-espace des N plus petits vecteurs propres est inversement proportionnel à : $\xi - ||E||_2$.

L'erreur entre l'itéré D_n et la matrice densité exacte D est décomposée en deux contributions :

- l'écart entre D_n et le projecteur orthogonal sur le sous-espace engendré par les N plus petits vecteurs propres de D_n, qui est relié à l'écart entre les valeurs propres de D_n et celle de D. Ce terme est noté ϵ_n^λ,
- l'écart entre le sous-espace engendré par les N plus petits vecteurs propres de D_n et le sous-espace engendré par les N plus petits vecteurs propres de F. Ce terme est noté ϵ_n^Θ.

Dans les premières itérations, ϵ_n^Θ est très faible puisque la suite est initialisée avec la matrice F' elle-même. Il croît au fur et à mesure des itérations à cause des troncatures sur les matrices D_n. Au contraire, ϵ_n^λ est maximal dans les premières itérations, puis il converge vers 0 (si l'algorithme converge).

Le nombre d'itérations nécessaires pour atteindre une précision donnée sur ϵ^λ est calculé avant de démarrer l'algorithme. La précision recherchée sur ϵ^Θ en fin d'algorithme est également fixée *a priori*. Le seuil de troncature est déterminé à chaque itération de manière à ce que le critère sur ϵ_n^Θ soit satisfait, ainsi que des critères de convergence et de stabilité : augmentation stricte du *gap* de D_n, maintient des valeurs propres extrêmes de D_n dans l'intervalle $[0, 1]$.

Dans les tests numériques présentés, le coût de ce contrôle de l'erreur représente de l'ordre de 20% du coût total.

Pour terminer avec les méthodes de projection, signalons que la clé de l'efficacité des implémentations est de disposer d'une implémentation efficace des produits de matrices creuses. Dans les processeurs actuels, la vitesse des calculs est limitée par le taux de transfert des données de la mémoire vers le processeur. Les performances observées (nombre d'opérations par seconde) sont bien plus élevées pour des produits de matrices pleines que pour des produits de matrices creuses puisque ces dernières

ont un taux d'accès à la mémoire ramené au nombre d'opérations plus important. De nombreuses équipes ont développé des implémentations spécifiques pour le produit de matrices creuses, dans le cadre de l'implémentation d'algorithmes d'ordre N, consistant à utiliser des structures de données particulières au prix de la prise en compte d'un plus grand nombre de termes [115, 116, 119, 120].

A.3 Les méthodes de décomposition de domaine

Un grand nombre de méthodes basées sur un paradigme de type "décomposition de domaine" ont été proposées [9, 42, 46, 76]. L'intérêt actuel pour ces approches est probablement induit par la domination, depuis plusieurs années, des calculateurs parallèles parmi les moyens de calcul de pointe. La méthode proposée par W. Yang [80] est la plus citée dans la littérature des méthodes d'ordre N pour les modèles Hartree-Fock et DFT. Elle a été notamment reprise récemment dans des calculs DFT mettant en jeu plus d'un million d'atomes [66, 67]. C'est cette méthode qui est détaillée dans les paragraphes suivants.

La méthode "Divide-and-Conquer" de Yang consiste à definir des sous-ensembles de fonctions de base comme dans la méthode MDD. Une première différence importante avec la méthode MDD est que la matrice densité est calculée et non les orbitales moléculaires du système. La matrice densité totale est formée par sommation des contributions des matrices densité calculées sur chaque sous-domaine. avec une pondération pour les fonctions de base appartenant à plusieurs sous-domaines. La matrice densité sur chaque sous-domaine est calculée en diagonalisant l'opérateur local F, sans tenir compte des domaines voisins. Le nombre de vecteurs propres de l'opérateur local à prendre en compte dans la matrice densité locale est déterminée en choisissant les N orbitales locales minimisant l'énergie totale, ce qui correspond à l'étape d'échange dans la méthode MDD. C'est le seul couplage entre les domaines.

A taille de domaine équivalent, les résultats publiés sont cohérents avec ce qu'on observe en appliquant MDD sans l'étape globale : erreur relative de l'ordre de 10^{-3} sur l'énergie. La précision sur l'énergie est augmentée par agrandissement du recouvrement entre les domaines. Une précision équivalente à ce qui est obtenu avec MDD est obtenue avec des recouvrements représentant de l'ordre de 80% de la taille totale du domaine.

Enfin, cette méthode revient à calculer la matrice densité à partir d'orbitales non orthogonales, ce qui a pour conséquence, en DFT, d'introduire une erreur dans le calcul de l'énergie cinétique, à l'itération SCF suivante. Dans [66, 67], les auteurs utilisent une correction de l'énergie cinétique basée sur une formule développée dans le cadre de calcul de DFT "orbital-free" [38].

Pour terminer avec les méthodes de type "Divide-and-Conquer" signalons la méthode de Seijo et Barandiaran [70, 71], qui repose sur les méthodes classiques de localisation des orbitales moléculaires développées par les chimistes [18, 25, 58].

Là encore, on n'impose pas l'orthogonalité entre les orbitales calculées dans des domaines différents.

A.4 Le passage au cas $S \neq I_{N_b}$

Une première possibilité consiste à calculer une factorisation approchée de S^{-1} en $O(N)$ opérations et à se ramener au cas $S = I$, comme cela est décrit dans le chapitre 5 paragraphe 5.2.2.1. Plusieurs algorithmes ont été proposés [43, 63, 69].

En complément, chacune des méthodes présentées précédemment peut être généralisée au cas $S \neq I$.

Pour les méthodes de projection, on remplace F par $\widetilde{F} = S^{-1}F$. Cette matrice est creuse [34] et elle est calculée par résolution de $S\widetilde{F} = F$.

Parmi les méthodes variationnelles, la méthode DMM est la plus pénalisée par le passage du cas $S = I$ au cas $S \neq I$. Une première généralisation proposée dans [52] consiste à minimiser :

$$\mathrm{Tr}\left(H\ (3DSD - 2DSDSD)\right)$$

à la place de :

$$\mathrm{Tr}\left(H\ (3D^2 - 2D^3)\right).$$

La méthode se trouve pénalisée à la fois par l'augmentation significative du nombre de produits de matrices à chaque itération et par l'augmentation du nombre de termes non nuls.

Annexe B

Calcul de dérivées pour l'étape globale

Cette annexe présente en détail le calcul du gradient et du hessien de la fonctionnelle minimisée au cours d'une étape globale portant sur deux blocs (formules (5.3)-(5.4)).

B.1 Notations et rappel des formules

On précise les notations qui sont un peu différentes de celles de la section 3.3. Comme au chapitre 4, on utilise la matrice T pour exprimer le produit scalaire entre deux blocs voisins.

On se situe au sein d'une itération de l'algorithme MDD. L'étape locale a produit les blocs $(\widetilde{C}_i)_{1\leq i\leq p}$, $\widetilde{C}_i \in \mathcal{M}^{n,m_i}(\mathbb{R})$, qui vérifient :

$$\left\{ \begin{array}{lcl} \left[\widetilde{C}_i\right]^t \widetilde{C}_i & = & I_{m_i} \\ \left[\widetilde{C}_i\right]^t T\widetilde{C}_{i+1} & = & 0 \end{array} \right.$$

Dans l'étape globale complète (sur p blocs) on minimise la fonction f :

$$f(\mathcal{U}) = \sum_{i=1}^{p} \mathrm{Tr}\Big(H_i C_i(\mathcal{U})[C_i(\mathcal{U})]^t\Big),$$

où $\mathcal{U} = (U_0, ..., U_p)$ avec la convention :

$$U_0 = U_p = 0.$$

et :

$$\begin{array}{lcl} C_i(\mathcal{U}) & = & \overline{C_i}(\mathcal{U})\Big(\overline{C_i}(\mathcal{U})^t\overline{C_i}(\mathcal{U})\Big)^{-\frac{1}{2}} \\ \overline{C_i}(\mathcal{U}) & = & \widetilde{C}_i + T\widetilde{C}_{i+1}U_i\Big([\widetilde{C}_i]^t TT^t\widetilde{C}_i\Big) - T^t\widetilde{C}_{i-1}U_{i-1}^t\Big([\widetilde{C}_i]^t T^t T\widetilde{C}_i\Big). \end{array}$$

En pratique, on ne traite que 2 blocs à la fois ($p = 2$) et on modifie la formule pour diminuer la taille de la matrice U (voir (5.3)-(5.4)) :

$$\overline{C_1}(\mathcal{U}) = \widetilde{C}_1 + B_2 U \Big(B_1^t T^t \widetilde{C}_1 \Big) \tag{B.1}$$

$$\overline{C_2}(\mathcal{U}) = \widetilde{C}_2 - B_1 U^t \Big(B_2^t T \widetilde{C}_2 \Big), \tag{B.2}$$

où $B_i \in \mathcal{M}^{(n,l_i)}$, avec $l_i \leq m_i$.

Dans toute la suite de cette annexe, les calculs sont réalisés pour les formules (B.1) et (B.2).

Concernant le choix des matrices B_i, on renvoit à 5.1.1.2. Rappelons qu'il est impératif que les colonnes de B_1 (resp. B_2) soient constituées de combinaisons linéaires des colonnes de $T^t\widetilde{C}_1$ (resp. $T\widetilde{C}_2$) :

- pour que l'orthogonalité soit maintenue au cours de l'étape globale,
- pour simplifier le calcul du gradient en $U = 0$ (cf formules (B.31)),
- pour obtenir une relation entre le gradient et les multiplicateurs de Lagrange (voir la formule (B.32), p. 112)).

Pour simplifier la présentation des calculs, on introduit les matrices symétriques :

$$J_i(U) = \overline{C_i}(U)^t \ \overline{C_i}(U)$$

et on pose :

$$E_i(U) = \mathrm{Tr}\Big(H_i C_i(U) [C_i(U)]^t \Big) = \mathrm{Tr}\Big(H_i \overline{C_i} J_i^{-1} [\overline{C_i}]^t \Big).$$

Il y a là un changement de notation par rapport à [11] où J_i représentait

$$\big(\overline{C_i}(U)^t \overline{C_i}(U) \big)^{-1}.$$

Dans toute la suite on ne précise plus les dépendances en U et on notera donc $\overline{C_i}$ à la place de $\overline{C_i}(U)$, J_i à la place de $J_i(U)$, E_i à la place de $E_i(U)$.

B.2 Formules communes aux deux calculs

B.2.1 Dérivées partielles de J_i^{-1}

On suppose que U reste suffisamment faible pour que $\overline{C_i}$ garde un rang égal au nombre de ses colonnes et donc que J_i reste inversible. En dérivant la relation $J_i \, J_i^{-1} = I$:

$$\frac{\partial J_i^{-1}}{\partial U_{hj}} = -J_i^{-1} \, \frac{\partial J_i}{\partial U_{hj}} \, J_i^{-1} \tag{B.3}$$

$$= -J_i^{-1} \left(\frac{\partial \overline{C_i}^t}{\partial U_{hj}} \, \overline{C_i} + \overline{C_i}^t \, \frac{\partial \overline{C_i}}{\partial U_{hj}} \right) \, J_i^{-1} \tag{B.4}$$

B.2.2 Expression des dérivées pour une étape globale à 2 blocs

Pour trois matrices U, V, W dont les dimensions permettent de définir le produit UVW :

$$\frac{\partial\big(VUW\big)_{pq}}{\partial U_{hj}} = V_{ph}W_{jq} \tag{B.5}$$

On utilise cette formule pour calculer les dérivées partielles de $\overline{C_1}$ et $\overline{C_2}$ par rapport à U :

$$\left(\frac{\partial\overline{C_1}}{\partial U_{hj}}\right)_{pq} = \big(B_2\big)_{ph}\big(B_1^tT^t\widetilde{C}_1\big)_{jq} \tag{B.6}$$

$$\left(\frac{\partial\overline{C_2}}{\partial U_{hj}}\right)_{pq} = -\big(B_1\big)_{pj}\big(B_2^tT\widetilde{C}_2\big)_{hq} \tag{B.7}$$

Remarque 1
On constate que $\left(\frac{\partial\overline{C_2}}{\partial U_{hj}}\right)_{pq}$ *peut s'obtenir formellement à partir de* $\left(\frac{\partial\overline{C_1}}{\partial U_{hj}}\right)_{pq}$ *en appliquant à la formule (B.6) les transformations suivantes :*
- *transposition des indices* 1 *et* 2,
- *remplacement de* T *par* T^t,
- *transposition des indices* h *et* j,
- *changement de signe.*

On peut maintenant exprimer les dérivées partielles de J_i, par simple application des règles de dérivation d'un produit :

$$\left(\frac{\partial J_1}{\partial U_{hj}}\right)_{pq} = \frac{\partial\overline{C_1}^t}{\partial U_{hj}}\,\overline{C_1} + \overline{C_1}^t\,\frac{\partial\overline{C_1}}{\partial U_{hj}} \tag{B.8}$$

$$= \big(B_1^tT^t\widetilde{C}_1\big)_{jp}\big(B_2^t\overline{C_1}\big)_{hq} + \big(B_1^tT^t\widetilde{C}_1\big)_{jq}\big(B_2^t\overline{C_1}\big)_{hp} \tag{B.9}$$

En remplaçant $\overline{C_1}$ par $\overline{C_2}$ et en appliquant la remarque 1 :

$$\left(\frac{\partial J_2}{\partial U_{hj}}\right)_{pq} = -\big(B_2^tT\widetilde{C}_2\big)_{hp}\big(B_1^t\overline{C_2}\big)_{jq} - \big(B_2^tT\widetilde{C}_2\big)_{hq}\big(B_1^t\overline{C_2}\big)_{jp} \tag{B.10}$$

B.2.3 Simplification des traces de produit de matrices

Remarque 2
La trace d'un produit de matrices $A^{(1)}...A^{(n)}$ *dont l'une,* $A^{(r)}$, *est une matrice de rang* 1 *de la forme* wz^t *peut être vue comme le produit scalaire du vecteur* z *et du produit des matrices* $A^{(i)}$ *(sauf* $A^{(r)}$*) appliqué au vecteur* w *:*

$$Tr\left(A^{(1)}...A^{(n)}\right) = z^tA^{(r+1)}...A^{(n)}A^{(1)}...A^{(r-1)}w. \tag{B.11}$$

Remarque 3
Si le vecteur w est le vecteur colonne d'indice u d'une matrice, notée $B^{(r)}$, et si le vecteur z est le vecteur colonne d'indice v d'une autre matrice, notée $C^{(r)}$, on a :

$$A_{ij}^{(r)} = B_{iu}^{(r)}\ C_{jv}^{(r)}. \tag{B.12}$$

Dans ce cas :

$$Tr\left(A^{(1)}...A^{(n)}\right) = \left([C^{(r)}]^t A^{(r+1)}...A^{(n)}A^{(1)}...A^{(r-1)}B^{(r)}\right)_{vu} \tag{B.13}$$

Dans ce qui précède, les matrices $\left(\dfrac{\partial \overline{C_i}}{\partial U_{hj}}\right)$ et $\left(\dfrac{\partial J_i}{\partial U_{hj}}\right)$ sont de la forme (B.12) :

$$\left(\frac{\partial \overline{C_1}}{\partial U_{hj}}\right)_{pq} = \left(B_2\right)_{ph}\left(\widetilde{C}_1^t T B_1\right)_{qj} \tag{B.14}$$

$$\left(\frac{\partial \overline{C_2}}{\partial U_{hj}}\right)_{pq} = -\left(B_1\right)_{pj}\left(\widetilde{C}_2^t T^t B_2\right)_{qh} \tag{B.15}$$

$$\left(\frac{\partial J_1}{\partial U_{hj}}\right)_{pq} = \left(\widetilde{C}_1^t T B_1\right)_{pj}\left(\overline{C_1}^t B_2\right)_{qh} + \left(\overline{C_1}^t B_2\right)_{ph}\left(\widetilde{C}_1^t T B_1\right)_{qj} \tag{B.16}$$

$$\left(\frac{\partial J_2}{\partial U_{hj}}\right)_{pq} = -\left(\widetilde{C}_2^t T^t B_2\right)_{ph}\left(\overline{C_2}^t B_1\right)_{qj} - \left(\overline{C_2}^t B_1\right)_{pj}\left(\widetilde{C}_2^t T^t B_2\right)_{qh} \tag{B.17}$$

On appliquera la remarque 3 chaque fois qu'on va rencontrer la trace d'un produit de matrice comportant un terme $\left(\dfrac{\partial \overline{C_i}}{\partial U_{hj}}\right)$ ou $\left(\dfrac{\partial J_i}{\partial U_{hj}}\right)$.

Terminons par une troisième formule qui servira au cours du calcul du hessien de f.

Remarque 4
Si de plus une deuxième matrice $A^{(s)}, s < t$ a une forme analogue à (B.12) :

$$A_{ij}^{(s)} = B_{iw}^{(s)}\ C_{jx}^{(s)} \tag{B.18}$$

alors :

$$Tr\left(A^{(1)}...A^{(n)}\right) = \left([C^{(s)}]^t A^{(s+1)}...A^{(r-1)}B^{(r)}\right)_{xu}\left([C^{(r)}]^t A^{(r+1)}...A^{(n)}A^{(1)}...A^{(s-1)}B^{(s)}\right)_{vw} \tag{B.19}$$

On appliquera la remarque 4 chaque fois qu'on va rencontrer la trace d'un produit de matrice comportant deux termes parmi $\left(\dfrac{\partial \overline{C_i}}{\partial U_{hj}}\right)$ et $\left(\dfrac{\partial J_i}{\partial U_{hj}}\right)$.

B.3 Calcul du gradient

On a posé :

$$E_i = \mathrm{Tr}\Big(H_i \overline{C_i} J_i^{-1} [\overline{C_i}]^t\Big) \tag{B.20}$$

En appliquant la règle de dérivation d'un produit :

$$\frac{\partial E_i}{\partial U_{hj}} = \mathrm{Tr}\left(H_i \frac{\partial \overline{C_i}}{\partial U_{hj}} J_i^{-1} \overline{C_i}^t\right) + \mathrm{Tr}\left(H_i \overline{C_i} \frac{\partial J_i^{-1}}{\partial U_{hj}} \overline{C_i}^t\right) + \mathrm{Tr}\left(H_i \overline{C_i} J_i^{-1} \frac{\partial \overline{C_i}}{\partial U_{hj}}^t\right) \tag{B.21}$$

$$= 2\ \mathrm{Tr}\left(H_i \frac{\partial \overline{C_i}}{\partial U_{hj}} J_i^{-1} \overline{C_i}^t\right) + \mathrm{Tr}\left(H_i \overline{C_i} \frac{\partial J_i^{-1}}{\partial U_{hj}} \overline{C_i}^t\right) \text{ (car } H_i \text{ et } J_i \text{ symétriques)} \tag{B.22}$$

$$= 2\ \mathrm{Tr}\left(H_i \frac{\partial \overline{C_i}}{\partial U_{hj}} J_i^{-1} \overline{C_i}^t\right) - \mathrm{Tr}\left(H_i \overline{C_i} J_i^{-1} \frac{\partial J_i}{\partial U_{hj}} J_i^{-1} \overline{C_i}^t\right) \tag{B.23}$$

$$\overset{not.}{=} 2\mathcal{T}_{ihj}^{(1)} + \mathcal{T}_{ihj}^{(2)} \tag{B.24}$$

Calcul des $\mathcal{T}_{ihj}^{(1)}$

$$\mathcal{T}_{ihj}^{(1)} = \mathrm{Tr}\left(H_i \frac{\partial \overline{C_i}}{\partial U_{hj}} J_i^{-1} \overline{C_i}^t\right)$$

On peut appliquer la remarque 3, qui donne directement :

$$\mathcal{T}_{1hj}^{(1)} = \Big(B_1^t T^t \widetilde{C}_1 J_1^{-1} \overline{C_1}^t H_1 B_2\Big)_{jh} \tag{B.25}$$

Avec les transformations décrites à la remarque 1, ou par un calcul direct :

$$\mathcal{T}_{2hj}^{(1)} = -\Big(B_2^t T \widetilde{C}_2 J_2^{-1} \overline{C_2}^t H_2 B_1\Big)_{hj} \tag{B.26}$$

Calcul des $\mathcal{T}_{ihj}^{(2)}$

$$\mathcal{T}_{ihj}^{(2)} = -\mathrm{Tr}\left(H_i \overline{C_i} J_i^{-1} \frac{\partial J_i}{\partial U_{hj}} J_i^{-1} \overline{C_i}^t\right)$$

Ici encore, on calcule ce terme en appliquant la remarque 3 :

$$\mathcal{T}_{1hj}^{(2)} = -\Big(B_2^t \overline{C_1} J_1^{-1} \overline{C_1}^t H_1 \overline{C_1} J_1^{-1} \widetilde{C}_1^t T B_1\Big)_{hj} - \Big(B_1^t T^t \widetilde{C}_1 J_1^{-1} \overline{C_1}^t H_1 \overline{C_1} J_1^{-1} \overline{C_1}^t B_2\Big)_{jh} \tag{B.27}$$

$$= -2\Big(B_1^t T^t \widetilde{C}_1 J_1^{-1} \overline{C_1}^t H_1 \overline{C_1} J_1^{-1} \overline{C_1}^t B_2\Big)_{jh} \tag{B.28}$$

De la même façon :

$$\mathcal{T}_{2hj}^{(2)} = 2\left(B_2^t T\widetilde{C}_2 J_2^{-1}\overline{C_2}^t H_2\overline{C_2}J_2^{-1}\overline{C_2}^t B_1\right)_{hj} \tag{B.29}$$

B.3.1 Le gradient en $U \neq 0$

On a finalement :

$$\begin{aligned}\frac{1}{2}\frac{\partial f}{\partial U_{hj}} &= \left(B_2^t\left(I_n - \overline{C_1}J_1^{-1}\overline{C_1}^t\right)H_1\overline{C_1}J_1^{-1}\widetilde{C}_1^t T B_1\right)_{hj} \\ &\quad - \left(B_2^t T\widetilde{C}_2 J_2^{-1}\overline{C_2}^t H_2\left(I_n - \overline{C_2}J_2^{-1}\overline{C_2}^t\right)B_1\right)_{hj}\end{aligned} \tag{B.30}$$

B.3.2 Le gradient en $U = 0$

En $U = 0$, on a les simplifications suivantes :

$$\begin{cases}\overline{C_i} = \widetilde{C}_i \\ J_i = I\end{cases}$$

En reportant dans (B.30) et en utilisant le fait que $\widetilde{C_1}^t B_2 = 0$ et $B_1^t\widetilde{C_2} = 0$ (puisque les colonnes de B_1 sont des combinaisons linéaires de $T\,\widetilde{C_1}$) :

$$\frac{1}{2}\frac{\partial f}{\partial U_{hj}} = \left(B_2^t H_1\widetilde{C}_1\widetilde{C_1}^t T B_1\right)_{hj} - \left(B_2^t T\widetilde{C_2}\widetilde{C}_2^t H_2 B_1\right)_{hj} \tag{B.31}$$

Si on appelle comme dans [11] Λ_1 et Λ_2 les multiplicateurs de Lagrange associés à l'étape locale, donc définis par les équations suivantes :

$$\begin{cases}H_1\widetilde{C_1} = \widetilde{C_1}E_1 + T\widetilde{C_2}\Lambda_1^t \\ H_2\widetilde{C_2} = \widetilde{C_2}E_2 + T^t\widetilde{C_1}\Lambda_2\end{cases}$$

on a de manière analogue à la formule (3.14) de [11] :

$$\frac{1}{2}\left(\frac{\partial f}{\partial U_{hj}}\right)(0) = \left((B_2^t T^t\widetilde{C_2})(\Lambda_1^t - \Lambda_2^t)(\widetilde{C_1}^t T^t B_1)\right)_{hj} \tag{B.32}$$

puisque $\widetilde{C_1}^t B_2 = 0$ et $B_1^t\widetilde{C_2} = 0$.

Contrairement à ce qui est affirmé dans [11], on ne peut inverser cette relation et exprimer $\Lambda_1 - \Lambda_2$ en fonction du gradient de f en $U = 0$, car $T^t\widetilde{C_1}$ et $T\widetilde{C_2}$ ne sont pas de rang plein :

- si B_1 est une base de $T^t\widetilde{C_1}$, la matrice $(B_1^t T^t\widetilde{C_1})$ n'est pas carrée,
- si $B_1 = T^t\widetilde{C_1}$, $(B_1^t T^t\widetilde{C_1})$ étant la matrice de Gram de $T^t\widetilde{C_1}$, elle n'est pas inversible.

On peut néanmoins en tirer une information sur l'égalité des multiplicateurs de Lagrange. On reprend les notations des équations (5.5) et (5.6) :

$$B_2 = T\widetilde{C}_2\, Y_a\, \Sigma_a^{-1} \tag{B.33}$$
$$B_1 = T^t\widetilde{C}_1\, [Y']_a\, [\Sigma']_a^{-1} \tag{B.34}$$

En notant G_1 la matrice de Gram de $T^t\widetilde{C_1}$ et G_2 celle de $T\widetilde{C_2}$:

$$\frac{1}{2}\left(\frac{\partial f}{\partial U}\right)(0) = \left(\Sigma_a^{-t} Y_a^t G_2 (\Lambda_1 - \Lambda_2)^t G_1 Y_a' [\Sigma']_a^{-1}\right) \tag{B.35}$$
$$\frac{1}{2}\left(\frac{\partial f}{\partial U}\right)(0) = \left(\Sigma_a^{-t} Y_a^t G_2 (\Lambda_1 - \Lambda_2)^t Y_a' [\Sigma']_a^{t}\right) \tag{B.36}$$
$$G_2(\Lambda_1 - \Lambda_2)^t Y_a' = \frac{1}{2} Y_a \Sigma_a^t \left(\frac{\partial f}{\partial U}\right)(0)[\Sigma']_a^{-t} \tag{B.37}$$

Si le gradient est nul, $(\Lambda_1 - \Lambda_2)^t Y_a'$ est dans le noyau de G_2, qui est égal au noyau de $T\widetilde{C}_2$. On a donc :

$$T\widetilde{C}_2 \Lambda_1^t Y_a' = T\widetilde{C}_2 \Lambda_2^t Y_a' \tag{B.38}$$

Si on reporte cette égalité dans le système (B.3.2)-(B.3.2), on obtient :

$$\begin{cases} H_1 \widetilde{C_1} Y_a' = \left(\widetilde{C_1} E_1 + T\widetilde{C_2}\Lambda_2^t\right) Y_a' \\ H_2 \widetilde{C_2} Y_a' = \left(\widetilde{C_2} E_2 + T^t\widetilde{C_1}\Lambda_2\right) Y_a' \end{cases}$$

On a donc égalité des multiplicateurs de Lagrange quand on projette les équations sur le supplémentaire orthogonal du noyau de $T\widetilde{C_1}$.

Enfin, pour $U \neq 0$, on ne peut faire apparaitre de relation simple entre le gradient et $\Lambda_1 - \Lambda_2$.

B.4 Calcul du hessien pour 2 blocs

Notons que $\left(\frac{\partial \overline{C_i}}{\partial U_{hj}}\right)_{pq}$ ne dépend pas de U, au contraire de $\left(\frac{\partial J_i^{-1}}{\partial U_{hj}}\right)_{pq}$. En dérivant l'expression de $\frac{\partial E_i}{\partial U_{hj}}$ (B.23) par rapport à U_{kl}, on a donc :

$$\begin{aligned} \frac{\partial^2 E_i}{\partial U_{hj}\partial U_{kl}} &= 2\,\mathrm{Tr}\left(H_i \frac{\partial \overline{C_i}}{\partial U_{hj}} \frac{\partial J_i^{-1}}{\partial U_{kl}} \overline{C_i}^t\right) + 2\,\mathrm{Tr}\left(H_i \frac{\partial \overline{C_i}}{\partial U_{hj}} J_i^{-1} \frac{\partial \overline{C_i}^t}{\partial U_{kl}}\right) \\ &+ \mathrm{Tr}\left(H_i \frac{\partial \overline{C_i}}{\partial U_{kl}} \frac{\partial J_i^{-1}}{\partial U_{hj}} \overline{C_i}^t\right) + \mathrm{Tr}\left(H_i \overline{C_i} \frac{\partial^2 J_i^{-1}}{\partial U_{hj}\partial U_{kl}} \overline{C_i}^t\right) + \mathrm{Tr}\left(H_i \overline{C_i} \frac{\partial J_i^{-1}}{\partial U_{hj}} \frac{\partial \overline{C_i}^t}{\partial U_{kl}}\right) \end{aligned}$$

Le troisième et le cinquième termes de cette somme sont égaux :

$$\frac{\partial^2 E_i}{\partial U_{hj}\partial U_{kl}} = 2\ \mathrm{Tr}\left(H_i \frac{\partial \overline{C_i}}{\partial U_{hj}} \frac{\partial J_i^{-1}}{\partial U_{kl}} \overline{C_i}^t\right) + 2\ \mathrm{Tr}\left(H_i \frac{\partial \overline{C_i}}{\partial U_{hj}} J_i^{-1} \frac{\partial \overline{C_i}^t}{\partial U_{kl}}\right)$$
$$+ 2\ \mathrm{Tr}\left(H_i \frac{\partial \overline{C_i}}{\partial U_{kl}} \frac{\partial J_i^{-1}}{\partial U_{hj}} \overline{C_i}^t\right) + \mathrm{Tr}\left(H_i \overline{C_i} \frac{\partial^2 J_i^{-1}}{\partial U_{hj}\partial U_{kl}} \overline{C_i}^t\right) \tag{B.39}$$

Calcul de $\frac{\partial^2 J_i^{-1}}{\partial U_{hj}\partial U_{kl}}$

En dérivant l'expression de $\frac{\partial J_i^{-1}}{\partial U_{hj}}$ (B.4) par rapport à U_{kl}, J_i étant symétrique :

$$\frac{\partial^2 J_i^{-1}}{\partial U_{hj}\partial U_{kl}} = -\frac{\partial J_i^{-1}}{\partial U_{kl}} \frac{\partial J_i}{\partial U_{hj}} J_i^{-1} - \left(\frac{\partial J_i^{-1}}{\partial U_{kl}} \frac{\partial J_i}{\partial U_{hj}} J_i^{-1}\right)^t - J_i^{-1} \frac{\partial^2 J_i}{\partial U_{hj}\partial U_{kl}} J_i^{-1} \tag{B.40}$$

Quelque soit le nombre de blocs, on a : $\dfrac{\partial^2 \overline{C_i}}{\partial U_{hj}\partial U_{kl}} = 0$, donc :

$$\frac{\partial^2 J_i}{\partial U_{hj}\partial U_{kl}} = \frac{\partial \overline{C_i}^t}{\partial U_{hj}} \frac{\partial \overline{C_i}}{\partial U_{kl}} + \frac{\partial \overline{C_i}^t}{\partial U_{kl}} \frac{\partial \overline{C_i}}{\partial U_{hj}} \tag{B.41}$$

et :

$$\boxed{\begin{aligned} \frac{\partial^2 J_i^{-1}}{\partial U_{hj}\partial U_{kl}} = &-\left(\frac{\partial J_i^{-1}}{\partial U_{kl}} \frac{\partial J_i}{\partial U_{hj}} J_i^{-1} + J_i^{-1} \frac{\partial \overline{C_i}^t}{\partial U_{hj}} \frac{\partial \overline{C_i}}{\partial U_{kl}} J_i^{-1}\right) \\ &-\left(\frac{\partial J_i^{-1}}{\partial U_{kl}} \frac{\partial J_i}{\partial U_{hj}} J_i^{-1} + J_i^{-1} \frac{\partial \overline{C_i}^t}{\partial U_{hj}} \frac{\partial \overline{C_i}}{\partial U_{kl}} J_i^{-1}\right)^t \end{aligned} \quad \text{(B.42)}}$$

On remplace cette expression dans $\dfrac{\partial^2 E_i}{\partial U_{hj}\partial U_{kl}}$ (B.39), on a :

$$\begin{aligned} \frac{\partial^2 E_i}{\partial U_{hj}\partial U_{kl}} = &\ 2\ \mathrm{Tr}\left(H_i \frac{\partial \overline{C_i}}{\partial U_{hj}} \frac{\partial J_i^{-1}}{\partial U_{kl}} \overline{C_i}^t\right) + 2\ \mathrm{Tr}\left(H_i \frac{\partial \overline{C_i}}{\partial U_{hj}} J_i^{-1} \frac{\partial \overline{C_i}^t}{\partial U_{kl}}\right) \\ &+ 2\ \mathrm{Tr}\left(H_i \frac{\partial \overline{C_i}}{\partial U_{kl}} \frac{\partial J_i^{-1}}{\partial U_{hj}} \overline{C_i}^t\right) \\ &- 2\ \mathrm{Tr}\left(H_i \overline{C_i} \frac{\partial J_i^{-1}}{\partial U_{kl}} \frac{\partial J_i}{\partial U_{hj}} J_i^{-1} \overline{C_i}^t\right) - 2\ \mathrm{Tr}\left(H_i \overline{C_i} J_i^{-1} \frac{\partial \overline{C_i}^t}{\partial U_{hj}} \frac{\partial \overline{C_i}}{\partial U_{kl}} J_i^{-1} \overline{C_i}^t\right) \end{aligned} \tag{B.43}$$

On remplace maintenant partout $\left(\dfrac{\partial J_i^{-1}}{\partial U_{hj}}\right)_{pq}$ par l'expression (B.4) :

$$\begin{aligned}\frac{\partial^2 E_i}{\partial U_{hj}\partial U_{kl}} &= -2\ \mathrm{Tr}\left(H_i\frac{\partial\overline{C_i}}{\partial U_{hj}}J_i^{-1}\frac{\partial J_i}{\partial U_{kl}}J_i^{-1}\overline{C_i}^t\right) + 2\ \mathrm{Tr}\left(H_i\frac{\partial\overline{C_i}}{\partial U_{hj}}J_i^{-1}\frac{\partial\overline{C_i}^t}{\partial U_{kl}}\right)\\ &\quad - 2\ \mathrm{Tr}\left(H_i\frac{\partial\overline{C_i}}{\partial U_{kl}}J_i^{-1}\frac{\partial J_i}{\partial U_{hj}}J_i^{-1}\overline{C_i}^t\right)\\ &\quad + 2\ \mathrm{Tr}\left(H_i\overline{C_i}J_i^{-1}\frac{\partial J_i}{\partial U_{kl}}J_i^{-1}\ \frac{\partial J_i}{\partial U_{hj}}\ J_i^{-1}\overline{C_i}^t\right) - 2\ \mathrm{Tr}\left(H_i\overline{C_i}J_i^{-1}\ \frac{\partial\overline{C_i}^t}{\partial U_{hj}}\frac{\partial\overline{C_i}}{\partial U_{kl}}\ J_i^{-1}\overline{C_i}^t\right)\end{aligned} \tag{B.44}$$

Notons que le premier et le troisième termes de cette expression ont la même forme.

On pose :

$$\mathcal{S}^{(1)}_{ihjkl} = \mathrm{Tr}\left(H_i\frac{\partial\overline{C_i}}{\partial U_{hj}}J_i^{-1}\frac{\partial J_i}{\partial U_{kl}}J_i^{-1}\overline{C_i}^t\right) \tag{B.45}$$

$$\mathcal{S}^{(2)}_{ihjkl} = \mathrm{Tr}\left(H_i\frac{\partial\overline{C_i}}{\partial U_{hj}}J_i^{-1}\frac{\partial\overline{C_i}^t}{\partial U_{kl}}\right) \tag{B.46}$$

$$\mathcal{S}^{(3)}_{ihjkl} = \mathrm{Tr}\left(H_i\overline{C_i}J_i^{-1}\frac{\partial J_i}{\partial U_{kl}}J_i^{-1}\ \frac{\partial J_i}{\partial U_{hj}}\ J_i^{-1}\overline{C_i}^t\right) \tag{B.47}$$

$$\mathcal{S}^{(4)}_{ihjkl} = \mathrm{Tr}\left(H_i\overline{C_i}\ J_i^{-1}\frac{\partial\overline{C_i}^t}{\partial U_{hj}}\frac{\partial\overline{C_i}}{\partial U_{kl}}\ J_i^{-1}\overline{C_i}^t\right) \tag{B.48}$$

Avec ces notations, les termes du hessien sont donnés par :

$$\frac{\partial^2 E_i}{\partial U_{hj}\partial U_{kl}} = -2\ \left(\mathcal{S}^{(1)}_{ihjkl} + \mathcal{S}^{(1)}_{iklhj}\right) + 2\ \mathcal{S}^{(2)}_{ihjkl} + 2\ \mathcal{S}^{(3)}_{ihjkl} - 2\ \mathcal{S}^{(4)}_{ihjkl} \tag{B.49}$$

Cette expression vérifie la formule d'interversion des dérivées (transposition $(hj) \circlearrowright (kl)$) puisque :

- la formule est symétrisée par rapport aux termes $\mathcal{S}^{(1)}_i$,
- tous les autres $\mathcal{S}^{(t)}_i$ sont symétriques en $((hj),(kl))$.

Calcul des $\mathcal{S}^{(1)}_{iklhj}$

On utilise la simplification présentée à la remarque 4 :

$$\begin{aligned}\mathcal{S}^{(1)}_{1hjkl} &= \left(\left(\widetilde{C}_1^t T B_1\right)^t J_1^{-1}\widetilde{C}_1^t T B_1\right)_{jl}\left(\left(\overline{C_1}^t B_2\right)^t J_1^{-1}\overline{C_1}^t H_1 B_2\right)_{kh}\\ &\quad + \left(\left(\widetilde{C}_1^t T B_1\right)^t J_1^{-1}\overline{C_1}^t B_2\right)_{jk}\left(\left(\widetilde{C}_1^t T B_1\right)^t J_1^{-1}\overline{C_1}^t H_1 B_2\right)_{lh}\end{aligned} \tag{B.50}$$

Pour le deuxième bloc, on applique les règles de passages vues en remarque 1. Il n'y a pas de changement de signe car les deux changements de signe dus à la dérivée de $\overline{C_i}$ et à celle de J_i se compensent :

$$\begin{aligned}\mathcal{S}^{(1)}_{2hjkl} = &\left(\left(\widetilde{C}_2^t T^t B_2\right)^t J_2^{-1} \widetilde{C}_2^t T^t B_2\right)_{hk} \left(\left(\overline{C_2}^t B_1\right)^t J_2^{-1} \overline{C_2}^t H_2 B_1\right)_{lj} \\ &+ \left(\left(\widetilde{C}_2^t T^t B_2\right)^t J_2^{-1} \overline{C_2}^t B_1\right)_{hl} \left(\left(\widetilde{C}_2^t T^t B_2\right)^t J_2^{-1} \overline{C_2}^t H_2 B_1\right)_{kj}\end{aligned} \tag{B.51}$$

Tous les autres termes $\mathcal{S}_i^{(t)}$ s'obtiennent de même en appliquant les remarques 4 et 1.

Calcul des $\mathcal{S}^{(2)}_{iklhj}$

$$\mathcal{S}^{(2)}_{1hjkl} = \left(\left(\widetilde{C}_1^t T B_1\right)^t J_1^{-1} \widetilde{C}_1^t T B_1\right)_{jl} \left(B_2^t H_1 B_2\right)_{kh} \tag{B.52}$$

$$\mathcal{S}^{(2)}_{2hjkl} = \left(\left(\widetilde{C}_2^t T^t B_2\right)^t J_2^{-1} \widetilde{C}_2^t T^t B_2\right)_{hk} \left(B_1^t H_2 B_1\right)_{lj} \tag{B.53}$$

Calcul des $\mathcal{S}^{(3)}_{iklhj}$

$$\begin{aligned}\mathcal{S}^{(3)}_{1hjkl} = &\left(\left(\overline{C_1}^t B_2\right)^t J_1^{-1} \widetilde{C}_1^t T B_1\right)_{kj} \left(\left(\overline{C_1}^t B_2\right)^t J_1^{-1} \overline{C_1}^t H_1 \overline{C_1} J_1^{-1} \widetilde{C}_1^t T B_1\right)_{hl} \\ &+ \left(\left(\overline{C_1}^t B_2\right)^t J_1^{-1} \overline{C_1}^t B_2\right)_{kh} \left(\left(\widetilde{C}_1^t T B_1\right)^t J_1^{-1} \overline{C_1}^t H_1 \overline{C_1} J_1^{-1} \widetilde{C}_1^t T B_1\right)_{jl} \\ &+ \left(\left(\widetilde{C}_1^t T B_1\right)^t J_1^{-1} \widetilde{C}_1^t T B_1\right)_{lj} \left(\left(\overline{C_1}^t B_2\right)^t J_1^{-1} \overline{C_1}^t H_1 \overline{C_1} J_1^{-1} \overline{C_1}^t B_2\right)_{hk} \\ &+ \left(\left(\widetilde{C}_1^t T B_1\right)^t J_1^{-1} \overline{C_1}^t B_2\right)_{lh} \left(\left(\widetilde{C}_1^t T B_1\right)^t J_1^{-1} \overline{C_1}^t H_1 \overline{C_1} J_1^{-1} \overline{C_1}^t B_2\right)_{jk}\end{aligned} \tag{B.54}$$

$$\begin{aligned}\mathcal{S}^{(3)}_{2hjkl} = &\left(\left(\overline{C_2}^t B_1\right)^t J_2^{-1}\widetilde{C}_2^t T^t B_2\right)_{kj}\left(\left(\overline{C_2}^t B_1\right)^t J_2^{-1}\overline{C_2}^t H_2\overline{C_2}J_2^{-1}\widetilde{C}_2^t T^t B_2\right)_{hl}\\ &+\left(\left(\overline{C_2}^t B_1\right)^t J_2^{-1}\overline{C_2}^t B_1\right)_{kh}\left(\left(\widetilde{C}_2^t T^t B_2\right)^t J_2^{-1}\overline{C_2}^t H_2\overline{C_2}J_2^{-1}\widetilde{C}_2^t T^t B_2\right)_{jl}\\ &+\left(\left(\widetilde{C}_2^t T^t B_2\right)^t J_2^{-1}\widetilde{C}_2^t T^t B_2\right)_{lj}\left(\left(\overline{C_2}^t B_1\right)^t J_2^{-1}\overline{C_2}^t H_2\overline{C_2}J_2^{-1}\overline{C_2}^t B_1\right)_{hk}\\ &+\left(\left(\widetilde{C}_2^t T^t B_2\right)^t J_2^{-1}\overline{C_2}^t B_1\right)_{lh}\left(\left(\widetilde{C}_2^t T^t B_2\right)^t J_2^{-1}\overline{C_2}^t H_2\overline{C_2}J_2^{-1}\overline{C_2}^t B_1\right)_{jk}\end{aligned} \tag{B.55}$$

Calcul des $\mathcal{S}^{(4)}_{iklhj}$

$$\mathcal{S}^{(4)}_{1hjkl} = \left(B_2^t B_2\right)_{hk}\left(\left(\widetilde{C}_1^t T B_1\right)^t J_1^{-1}\overline{C_1}^t H_1\overline{C_1}J_1^{-1}\widetilde{C}_1^t T B_1\right)_{lj} \tag{B.56}$$

$$\mathcal{S}^{(4)}_{2hjkl} = \left(B_1^t B_1\right)_{hk}\left(\left(\widetilde{C}_2^t T^t B_2\right)^t J_2^{-1}\overline{C_2}^t H_2\overline{C_2}J_2^{-1}\widetilde{C}_2^t T^t B_2\right)_{lj} \tag{B.57}$$

B.4.1 Expression du hessien complet pour 2 blocs

B.4.1.1 Cas $U = 0$

Les termes $\mathcal{S}^{(1)}_{iklhj}$ et $\mathcal{S}^{(3)}_{iklhj}$ sont nuls car :

$$\overline{C_1}^t B_2 = \widetilde{C_1}^t B_2 = 0$$

En effet, $\widetilde{C_1}$ est orthogonal à toutes les colonnes de $T\widetilde{C_2}$ donc à toute combinaison linéaire de ces colonnes, et les colonnes de B_2 sont toutes des combinaisons linéaires de colonnes de $T\widetilde{C_2}$. De même : $\overline{C_2}^t B_1 = 0$.

On a donc :

$$\begin{aligned}\frac{1}{2}H_{hjkl} &= \left(B_2^t H_1 B_2\right)_{hk} \left(\left(\widetilde{C}_1^t T B_1\right)^t \widetilde{C}_1^t T B_1\right)_{lj} \\ &- \left(B_2^t B_2\right)_{hk} \left(\left(\widetilde{C}_1^t T B_1\right)^t \widetilde{C}_1^t H_1 \widetilde{C}_1 \widetilde{C}_1^t T B_1\right)_{lj} \\ &+ \left(\left(\widetilde{C}_2^t T^t B_2\right)^t \widetilde{C}_2^t T^t B_2\right)_{hk} \left(B_1^t H_2 B_1\right)_{lj} \\ &- \left(\left(\widetilde{C}_2^t T^t B_2\right)^t \widetilde{C}_2^t H_2 \widetilde{C}_2 \widetilde{C}_2^t T^t B_2\right)_{hk} \left(B_1^t B_1\right)_{lj} \end{aligned} \tag{B.58}$$

Avec des notations évidentes, on récrit (B.58) sous la forme :

$$H_{hjkl} = \sum_{i=1}^{4} L_{hk}^{(i)} \quad R_{lj}^{(i)}, \tag{B.59}$$

avec $L^{(i)} \in \mathcal{M}^{l_2,l_2}$ et $R^{(i)} \in \mathcal{M}^{l_1,l_1}$, toutes ces matrices étant symétriques.

Le produit "matrice hessienne" fois "vecteur X" devient, en écrivant le vecteur X sous une forme matricielle, $X \in \mathcal{M}^{l_2,l_1}$ comme on l'a fait pour le gradient :

$$(HX)_{hj} = \sum_{k=1}^{l_2} \sum_{l=1}^{l_1} H_{hjkl} X_{kl} \tag{B.60}$$

$$= \sum_{i=1}^{4} \sum_{k=1}^{l_2} \sum_{l=1}^{l_1} L_{hk}^{(i)} \; X_{kl} \; R_{lj}^{(i)} \tag{B.61}$$

$$HX = \sum_{i=1}^{4} L^{(i)} \; X \; R^{(i)} \tag{B.62}$$

Le calcul des matrices B_i par une décomposition en valeurs singulières permet d'obtenir des expressions très simples pour les matrices $L^{(i)}$ et $R^{(i)}$. Ces simplifications n'ont pas été utilisées dans l'implémentation "1D" et les tests numériques présentés au chapitre 4 n'en bénéficient pas. Elles ont par contre été utilisées dans l'implémentation "2D/3D", mais la correction due à la répartition quelconque des domaines (cf section 5.1.2) rendent le gain moins intéressant.

Le reste de cette section est consacré à l'exposé de ces simplifications.

Dès que les colonnes de B_i forment une base orthonormée, $L^{(2)} = I_{l_2}$ et $R^{(4)} = I_{l_1}$. De plus, dans le cas "1D", on a par exemple pour B_2 (cf, section 5.1.1.2, formules (5.5) à (5.6)) :

$$\left(T\widetilde{C}_2^k\right)^t B_2 = Y_a \Sigma_a \tag{B.63}$$

avec $Y_a^t Y_a = I$ et Σ_a la matrice diagonale des valeurs singulières supérieures à ϵ_g. $L^{(3)}$, la matrice de Gram de $Y_a \Sigma_a$, est donc une matrice diagonale formée des carrés des valeurs singulières apparaissant dans la matrice Σ_a. On peut faire le même calcul sur B_1 et prouver que $R^{(1)}$ est diagonale, et fournie par la décomposition en valeurs singulières de $T^t \widetilde{C}_1$.

Sur les huit matrices apparaissant dans (B.62), seules quatre d'entre elles, $L^{(1)}$ et $R^{(3)}$, doivent être explicitement calculées par la formule (B.62). Ces formules présentent également l'avantage d'éviter le calcul explicite des matrices de Gram, ce qui est à éviter notamment pour des matrices mal conditionnées (voir [95], p. 241).

Le cas "2D" est malheureusement moins favorable. On a toujours $B_i^t B_i = I$, mais B_i est le produit de deux matrices provenant de deux décompositions SVD différentes (calcul de l'intersection). On a maintenant :

$$\left(T\widetilde{C}_2^k\right)^t B_2 = Y_a \Sigma_a Y_A, \tag{B.64}$$

la matrice Y_A provenant du calcul de l'intersection. Donc le terme $L^{(3)}$ devient :

$$Y_A^t \Sigma_a^t \Sigma_a Y_A. \tag{B.65}$$

Il y a toujours quelques valeurs singulières strictement inférieures à 1 et supérieures à ϵ_g dans Σ_a donc la matrice $L^{(3)}$ n'est plus diagonale. Néanmoins la formule (B.65) permet de la calculer à faible coût. Il en va de même pour la matrice $R^{(1)}$.

B.4.1.2 Cas $U \neq 0$

Dans ce cas, le produit matrice-vecteur avec la matrice hessienne de f garde une expression du type (B.62) avec plus de termes. Cette expression n'a pour le moment été utilisée dans aucune implémentation de MDD. On l'indique ici pour mémoire.

En prenant tous les termes $\mathcal{S}^{(p)}$ dans le hessien, on fait apparaître une expression de la forme :

$$H_{hjkl} = \sum_{i=1}^{10} L_{hk}^{(i)} \quad R_{lj}^{(i)} + \sum_{i=1}^{6} \tilde{L}_{hk}^{(i)} \quad \tilde{R}_{lj}^{(i)}. \tag{B.66}$$

On a donc :

$$HX \quad = \textstyle\sum_{i=1}^{10} L^{(i)} \; X \; R^{(i)} + \sum_{i=1}^{6} \tilde{L}^{(i)} \; X \; \tilde{R}^{(i)}. \tag{B.67}$$

B.5 Implémentation du calcul du gradient et du hessien pour 2 blocs

On donne ici la liste des opérations à réaliser pour calculer simultanément le gradient et les matrices $L^{(i)}$ et $R^{(i)}$ dans le cas d'un calcul "2D" ou "3D".

On note G_2 la matrice $Y_A^t \Sigma_a^t \Sigma_a Y_A$ (voir équation (B.65)) obtenue au cours de la préparation de B_2 et qui stocke la valeur de $\left(T\widetilde{C}_2^k\right)^t B_2$. G_1 représente l'analogue en remplaçant B_2 par B_1 et $T\widetilde{C}_2$ par $T^t\widetilde{C}_1^k$.

Le calcul se fait en deux séquences d'opérations très similaires. La première séquence est :

(1) Calcul et stockage de $H_1 B_2$.

(2) Prendre la transposée du résultat de l'opération (1), puis multiplier à droite par $\widetilde{C}_1$, ce qui fournit $(H_1 B_2)^t \widetilde{C}_1$, résultat que l'on stocke.

(3) Multiplier à gauche le résultat de l'opération (1) par B_2^t, ce qui fournit $L^{(1)}$.

(4) Multiplier à droite le résultat de l'opération (2) par G_1, ce qui donne le premier terme du gradient.

(5) Multiplier G_1 à gauche par sa transposée, ce qui donne $R^{(1)}$.

(6) Multiplier à gauche G_1 par la diagonale stockant $\widetilde{C}_1^t H_1 \widetilde{C}_1$ (gardé en mémoire depuis l'étape locale), puis multiplier le résultat à gauche par G_1^t.

$L^{(3)}$, $L^{(4)}$, $R^{(3)}$ et le deuxième terme du gradient sont obtenus en faisant la même séquence avec permutation des indices 1 et 2, en modifiant seulement les item (2) et (4) respectivement par :

(2) Calculer et stocker $\widetilde{C}_2^t H_2 B_1$.

(4) Multiplier à gauche le résultat de (2) pat G_2^t.

La mémoire auxilliaire qui doit être utilisée pour ces opérations se limite au stockage des résultats des opérations (1) et (2), ce qui ne représente que $nl_{max} + l_{max}^2$.

La complexité des opérations (1) à (6) est :

$$n^2 l_2 + n^2 l_2 + n l_2^2 + l_2 n l_1 + l_1 m_1 l_1 + m_1 l_1 + l_1 m_1 l_1$$

donc la complexité totale du calcul du gradient et des matrices composant le hessien est majorée par :

$$4n^2 l_{max} + 4nl_{max}^2 + 4ml_{max}^2 + 2ml_{max} \sim 4n^2 l_{max} + 4nl_{max}^2 + 4ml_{max}^2. \qquad \text{(B.68)}$$

On a toujours les relations :

$$l_{max} \leq m \leq n.$$

Dans le cas d'une base grossière, l'écart n'est pas très grand et tous les termes de (B.68) sont significatifs. Si la base est plus fine, on peut atteindre $n \approx 10m$ et le terme dominant est alors : $4n^2 l_{max}$.

Annexe C

Mémoire et nombre d'opérations pour chaque étape

Cette annexe comporte deux parties. La première est consacrée à la mémoire utilisée par le code, la seconde au décompte du nombre d'opérations arithmétiques générées par une simulation. Dans les deux cas, on va balayer le code et exprimer ces quantités pour chaque étape du calcul, puis globalement pour l'ensemble du code en dégageant les grands termes. Au-delà du fait que mémoire et nombre d'opérations évoluent de manière linéaire avec N, ce que l'on sait depuis la phase de conception de l'algorithme, l'objectif est de :

- faire apparaître le poids relatif des différentes parties du calcul,
- montrer l'évolution des ces estimations lors du passage d'une répartition des atomes et des domaines de "1D" à "2D" puis "3D", ou lors du passage à des bases très riches.

Pour simplifier les expressions, on se place dans le cas où la taille des domaines est uniforme ($n_i = n$), ainsi que le nombre d'orbitales par domaine ($m_i = m$). On note :

- N, le nombre total d'orbitales à calculer,
- p, le nombre de domaines, $N = pm$
- $\overline{p}$, le nombre de couples de domaines qui se recouvrent. Pour une décomposition cartésienne comme présentée à la figure 5.1, on a : $\overline{p} \approx \alpha p$ avec $\alpha = 1$ en 1D, $\alpha = 4$ en 2D et $\alpha = 13$ en 3D.
- M_1, la somme des orbitales calculées sur tous les domaines voisins d'un domaine donné. Comme on a pris les m_i uniformes, M_1 est égal à m fois le nombre de domaines voisins. Dans le cas d'une décomposition cartésienne, on a : $M_1 = 2\alpha m$.
- M_2, la somme des orbitales calculées sur tous les domaines voisins communs à un couple de domaines voisins donnés (en rapport avec l'étape globale, voir section 5.1.2). $M_2 = 0$ en 1D. Pour les dimensions supérieures, ce nombre dépend de la position relative des deux domaines du couple. Par exemple sur

la figure 5.1, deux domaines alignés ont quatre voisins communs alors que deux domaines en diagonale en ont seulement deux. Pour une décomposition cartésienne, on a : $M_2 \leq \alpha m$.
- b_L, la taille des blocs utilisés par les routines LAPACK pour les algorithmes de diagonalisation et de SVD.

Dans ces paramètres, N représente le *grand* terme par excellence : c'est le terme qu'on veut prendre aussi grand que possible. Avec N, p et $\overline{p}$ doivent aussi être considérés comme des grands termes. De plus, le poids de $\overline{p}$ augmente avec la dimensionalité du système.

En bases localisées, bien que le nombre m_i d'orbitales calculées par domaines puisse varier de manière importante quand n est constant, n et m sont intrinsèquement reliés par le nombre de fonctions de bases par électrons. Dans l'esprit de la décomposition de domaine, m et n sont des petits termes devant N, puisqu'ils représentent des quantités locales. Il faut cependant nuancer. Tout d'abord, en pratique, on n'aura effectivement cette relation de domination que si le nombre de domaine p est grand et d'autant plus que le rapport n/m l'est aussi (influence de la qualité de la base utilisée). De plus, n et m augmentent tous les deux fortement avec la dimensionnalité du système, on a par exemple estimé que n devrait atteindre 10^4 pour des simulations "3D". Enfin, n augmente également avec la délocalisation des électrons. Dans les développements qui vont suivre, on aura donc tendance à considérer que n et m peuvent également être grands.

C.1 Mémoire

Les tableaux d'entiers ne sont pas comptabilisés, car ils représentent une infime part de la mémoire totale.

L'organisation de la mémoire dans le code a été exposée à la section 5.2.3. Rappelons qu'on définit un seul tableau de travail pour tous les stockages temporaires du calcul. On doit déterminer la taille de ce tableau *a priori* en fonction des paramètres du calcul. Pour cela, on procède récursivement. Le nombre de mots mémoires nécessaires pour une routine est la somme de la dimension des tableaux qu'elle utilise directement et du plus grand espace mémoire utilisé par l'un des sous-programmes qu'elle appelle.

On note :
- $\mathfrak{M}^{MDD}$ la mémoire totale,
- $\mathfrak{M}^{perm}$ la quantité de données stockées pendant tout le calcul,
- $\mathfrak{M}^{loc}$ et $\mathfrak{M}^{glo}$ la mémoire nécessaire pour faire respectivement l'étape locale et l'étape globale.

Si on applique la règle du calcul de la mémoire au niveau le plus haut du code :

$$\mathfrak{M}^{MDD} = \mathfrak{M}^{perm} + max\left\{\mathfrak{M}^{loc}, \mathfrak{M}^{glo}\right\}.$$

C.1.1 Données du problème et solution

Le tableau C.1 rappelle la taille des structures de données servant à définir le problème et sa solution. Pour un calcul séquentiel, si on n'écrit pas les matrices sur disque, $\mathfrak{M}^{perm}$ représente la somme des lignes de ce tableau.

Variable	Nombre de réels
C	$N\,n = pnm$
(H_i)	$\frac{1}{2}\,p\,n^2$
(S_i)	$\frac{1}{2}\,p\,n^2$
$\left(S^{(ij)}\right)$	$\overline{p}n^2$

TAB. C.1 – Données utilisées pendant l'ensemble du calcul.

Les deux dernières lignes de ce tableau disparaissent si $S = I$ ou si on fait une réorthogonalisation complète en début de calcul (cf 5.2.2.1).

Rappelons que des options ont été implémentées pour ne pas garder en mémoire les matrices H_i, S_i et $S^{(ij)}$ pendant tout le calcul. Dans ce cas, $\mathfrak{M}^{perm} = N\,n$

C.1.2 Etape locale

Pendant la résolution de tous les blocs, on doit conserver les énergies des vecteurs de chaque orbitale.

Pendant la résolution de tous les blocs d'une même couleur, on doit stocker les vecteurs en attente de classement (échange). L'utilisateur donne *a priori* un domaine de variation des m_i : $[m_{max}, m_{min}]$ de sorte que le nombre de vecteurs à choisir pour chaque bloc peut être restreint à : $m_{max} - m_{min}$. On note p_{coul} le nombre maximal de blocs dans une couleur de l'étape locale. Avec ces notations, le nombre de données à stocker pour l'échange est :

$$n\,p_{coul}\,(m_{max} - m_{min}) + N + p_{coul}\,(m_{max} - m_{min})$$

La résolution de chaque problème local (4.3) consiste à

1. calculer une base orthonormée de $\mathcal{V}$, représentée par la matrice V,
2. assembler et diagonaliser la matrice $V^t H_i V$.

La seule donnée commune à ces deux étapes est la base orthonormée de $\mathcal{V}$, pour laquelle on prévoit n^2 réels. Il reste donc maintenant à déterminer laquelle de ces deux étapes utilise le plus grand espace mémoire.

$$\mathfrak{M}^{loc} = (n+1)\,p_{coul}\,(m_{max} - m_{min}) + N + n^2 + max\left\{\mathfrak{M}^{\mathcal{V}}, \mathfrak{M}^{diago}\right\}.$$

Mémoire pour le calcul de $\mathcal{V}$. Il faut :

- nM_1 réels pour le stockage de la matrice (4.5),
- pour le calcul de la SVD de cette matrice, sous l'hypothèse[1] $n \leq M_1 \leq 2n$:
  - $n + b_L(3n + M_1)$ si on passe par *dgesvd*, qui utilise l'algorithme QR,
  - $n + b_L(7n^2 + 4n)$ si on passe par *dgesdd*, qui utilise l'algorithme Divide-and-Conquer.

Mémoire pour la diagonalisation de $V^t H_i V$. En tenant compte du calcul du produit de matrices et de l'espace de travail pour la routine de diagonalisation :

- $2n^2 + 3\,b_L\,n + n$, en diagonalisant par *dsyev*, qui utilise l'algorithme QR,
- $2n^2 + b_L(2n^2 + 6n + 1) + n$, en diagonalisant par *dsyevd*, qui utilise l'algorithme Divide-and-Conquer.

Bilan pour l'ensemble de l'étape locale. Si on est très contraint en mémoire, on devra choisir les algorithmes basés sur QR pour les deux sous-étapes. C'est alors la phase de diagonalisation qui est la plus consommatrice : $\mathfrak{M}^{\mathcal{V}} < \mathfrak{M}^{diago}$. On a au total :

$$\begin{aligned} \mathfrak{M}^{loc} &= (n+1)\,p_{coul}\,(m_{max} - m_{min}) + N + 3n^2 + 3nb_L + n && \text{(C.1)} \\ &\sim 3n^2 + (n+1)\,p_{coul}\,(m_{max} - m_{min}), && \text{(C.2)} \end{aligned}$$

et on a toute latitude quant au choix de b_L.

Dans l'autre extrémité où on est assez riche en mémoire pour cette étape locale, on pourra se permettre d'utiliser les routines basées sur l'algorithme Divide-and-Conquer, c'est alors le calcul de $\mathcal{V}$ qui est dimensionnant :

$$\begin{aligned} \mathfrak{M}^{loc} &= (n+1)\,p_{coul}\,(m_{max} - m_{min}) + N + n + n^2 + b_L(7n^2 + 4n) + nM_1 && \text{(C.3)} \\ &\sim (7b_L + 1)\,n^2 + nM_1 + (n+1)\,p_{coul}\,(m_{max} - m_{min}), && \text{(C.4)} \end{aligned}$$

tous les termes de cette somme étant du même ordre. Cette fois-ci b_L apparait dans le terme dominant, donc on peut avoir à diminuer l'efficacité des calculs des routines LAPACK à cause de contraintes de mémoire.

Si on a choisi de ne pas stocker toutes les matrices locales ($S \neq I$), il faut ajouter aux expressions (C.1) et (C.4) :

$$\frac{3}{2}\,n^2,$$

pour le stockage des matrices locales du bloc en cours de résolution.

[1] raisonnable en bases localisées, compte tenu de la relation entre M_1 et m, car en "2D" et en "3D" chaque domaine aura au moins 6 domaines voisins.

C.1.3 Etape globale

Aucun tableau conséquent ne doit être stocké pendant toute l'étape globale, donc l'espace mémoire est déterminé par la résolution d'un seul problème global sur un couple de domaines, qui consiste en deux étapes :

1. calcul des matrices B_i,
2. minimisation de la fonctionnelle.

Pour simplifier, on suppose que les deux matrices B_i ont le même nombre de colonnes, qu'on note l.

Comme pour l'étape locale, il faut déterminer laquelle de ces deux sous-étapes demande le plus de stockage temporaire et lui ajouter l'espace correspondant aux données devant être stockées du début à la fin de la résolution, à savoir :

- les matrices B_i : $2\,n\,l$,
- les matrices G_i (voir section B.5 sur l'implémentation du calcul du gradient et du hessien pendant l'étape globale) : $2\,m\,l$.

$$\mathfrak{M}^{glo} = 2\,n\,l + 2\,m\,l + max\left\{\mathfrak{M}^{B}, \mathfrak{M}^{minim}\right\}.$$

On estime maintenant le stockage temporaire nécessaire pour chacune des deux sous-étapes. Les notations se rapportent aux formules de la page 85 : on calcule l'étape globale sur les domaines i et $i+1$. On note j les domaines voisins communs aux deux domaines i et $i+1$.

Calcul des matrices B_i. Données qui persistent pendant tout ce calcul :

- une base orthonormée de $S^{(i+1,i)}\,\widetilde{C}_i$: $n\,m$ réels,

Il y a trois décompositions SVD à calculer (voir p. 85) et il faut ajouter aux données ci-dessus le plus grand des trois stockages intermédiaires pour ces opérations. La plus grande matrice à traiter est $S^{(i+1,i)}\,\widetilde{C}_i$ et de plus il faut aussi stocker les vecteurs singuliers à gauche pour un calcul rapide des matrices G_i. Cette étape est donc déterminante. Elle demande :

- $2\,n\,m + m + b_L\,(n+3m)$ pour une SVD par l'algorithme QR, sous l'hypothèse (raisonnable) $n > 2m$,
- $2\,n\,m + m + b_L\,(7\,m^2 + 4\,m)$ pour une SVD par l'algorithme Divide-and-Conquer, sans hypothèse sur n et m.

Au total, pour le calcul des matrices B_i, on a donc, pour une SVD en QR :

$$\mathcal{M}^{B} = 3\,n\,m + m + b_L\,(n + 3\,m)$$

ou , pour une SVD en DC :

$$\mathcal{M}^{B} = 3\,n\,m + m + b_L\,(7\,m^2 + 4\,m).$$

Si on a choisit de ne pas stocker les matrices locales, on doit ajouter $2\,n^2$ dans les expressions précédentes.

	Stockage en mémoire		Stockage sur disque	
	$\mathfrak{M}^B$	$\mathfrak{M}^{minim}$	$\mathfrak{M}^B$	$\mathfrak{M}^{minim}$
QR	$3\,n\,m$	$2\,n^2 + 8\,l^2 + 2\,n\,m$	$2\,n^2 + 3\,n\,m$	$3\,n^2 + 8\,l^2 + 2\,n\,m$
DC	$3\,n\,m + 7\,b_L\,m^2$	$2\,n^2 + 8\,l^2 + 2\,n\,m$	$2\,n^2 + 3\,n\,m + 7\,b_L\,m^2$	$3\,n^2 + 8\,l^2 + 2\,n\,m$

TAB. C.2 – Comparaison de la mémoire nécessaire pour l'étape globale pour différentes options de calcul.

Minimisation de la fonctionelle. Les données persistant pendant tout le calcul sont : le gradient, la direction de descente et les matrices $L^{(i)}$ et $R^{(i)}$, ce qui représente au total $8\,l^2$ réels.

Les opérations à réaliser sont :

- le calcul du gradient et des matrices composant le hessien : $n^2 + 2\,n\,l$,
- l'inversion itérative, par l'algorithme SYMMLQ : $6\,l^2$ réels (voir section B.5),
- la recherche linéaire : $2\,n\,m + 2\,n^2$. Le terme en n^2 provient de l'évaluation de la fonctionnelle et son coefficient pourrait être réduit à 1 en remplaçant des BLAS 3 par des BLAS 1.

C'est la recherche linéaire qui utilise le plus de mémoire, conduisant pour l'ensemble de la minimisation à :

$$2\,n^2 + 8\,l^2 + 2\,n\,m$$

Si on a choisit de ne pas stocker les matrices locales, on doit ajouter n^2 dans les expressions précédentes.

Bilan pour l'ensemble de l'étape globale. On a finalement quatre cas de figures en croisant les options QR/Divide-and-Conquer et stockage des matrices locales en mémoire/sur disque. Le tableau C.2 récapitule les formules obtenues pour chacun de ces cas, en négligeant les termes d'ordre 1 en n, m.

La mémoire nécessaire pour la minimisation de la fonctionnelle est clairement plus grande que pour le calcul des B_i, quand on fait les SVD par un algorithme QR, mais si on prend l'option Divide-and-Conquer, c'est le contraire, tant que (en suppposant $b_L \geq 16$, ce qui est courant) $n \lesssim 8m$ ce qui est en général le cas en bases localisées.

Au total pour l'étape globale, il faut donc, pour une SVD en QR :

$$\mathfrak{M}^{glo} \sim 2\,n^2 + 2\,n\,m + 2\,n\,l + 2\,m\,l + 8\,l^2, \tag{C.5}$$

les termes de cette somme étant tous du même ordre, mais ordonnés en sens décroissant ou pour une SVD en DC :

$$\begin{aligned} \mathfrak{M}^{glo} &= 3\,n\,m + 2\,n\,l + b_L\,(7\,m^2 + 4\,m) + 2\,m\,l + m && \text{(C.6)} \\ &\sim 3\,n\,m + 2\,n\,l + 7\,b_L\,m^2 + 2\,m\,l. && \text{(C.7)} \end{aligned}$$

Si on a choisit de ne pas stocker les matrices locales, on doit ajouter n^2 dans (C.5) et $2\,n^2$ dans (C.7).

C.1.4 Mémoire totale en fonction des options

Le tableau ci-dessous récapitule les expressions des termes dominants (en allégeant par la simplification $l = m$) pour l'étape locale et l'étape globale, et sous les options algorithmes en QR et algorithmes en Divide-and-Conquer quand on n'écrit pas sur les disques.

Les deux expressions obtenues sur la ligne QR d'une part et sur la ligne Divide-and-Conquer d'autre part sont assez proches et on peut dire que dès qu'on pourra utiliser les routines basées sur l'algorithme Divide-and-Conquer dans l'une des deux étapes on pourra aussi l'utiliser sur l'autre.

Quand on travaille en QR, c'est l'étape globale qui prend le plus de mémoire, et dans le cas D-C, c'est l'étape locale.

	Etape locale	Etape globale
QR	$3n^2$ $+(n+1)\,p_{coul}\,(m_{max} - m_{min})$	$2n^2 + 4nm + 10m^2$
D-C	$(7b_L + 1)\,n^2 + nM_1$ $+(n+1)\,p_{coul}\,(m_{max} - m_{min})$	$5\,n\,m + (7\,b_L + 2)\,m^2$

Quelle que soit la variante de l'algorithme utilisée, la mémoire totale est dominée par $\mathfrak{M}^{perm}$.

On remarque également que l'écart QR/D-C est beaucoup plus faible que le poids du stockage des matrices locales (cf tableau C.1), dès qu'on a un nombre de domaines important.

C.2 Nombre d'opérations

On ne peut estimer *a priori* le nombre d'opérations d'un calcul MDD, car l'algorithme est itératif et nous n'avons pas d'estimation du nombre d'itérations nécessaires pour atteindre une précisions données.

On calculera donc seulement le nombre d'opération pour la phase d'initialisation, $\mathfrak{F}^{init}$, le nombre d'opérations pour une étape locale, $\mathfrak{F}^{loc}$, et le nombre d'opérations pour une étape globale, $\mathfrak{F}^{glo}$.

Les deux paragraphes suivants donnent des estimations du nombre d'opérations pour la diagonalisation et la décomposition en valeurs singulières des matrices. Elles seront utilisées dans toute la suite de ce chapitre.

Nombre d'opérations pour la décomposition en valeurs singulières par une méthode directe d'une matrice à n lignes et m colonnes ($m < n$). Les méthodes directes pour le calcul de la SVD d'une matrice procèdent en deux phases : la mise sous forme bidiagonale et le calcul de la SVD de la matrice bidiagonale. La mise sous forme bidiagonale se fait par une méthode basée sur les matrices de Householder et coûte : $4\,m^2\,(n - \frac{1}{3}m)$ opérations. C'est pour le calcul de la SVD de la matrice bidiagonale que porte l'effet du choix de l'algorithme (QR ou Divide-and-Conquer) : les deux algorithmes prennent théoriquement $O(m^3)$ opérations, même si le second est beaucoup plus rapide. On notera donc comme expression du nombre d'opérations : $4\,m^2\,n + \kappa_{SVD}\,n^3$, κ_{SVD} étant une constante dépendant de la méthode choisie.

Nombre d'opérations pour diagonalisation par une méthode directe d'une matrice symétrique d'ordre n. Les méthodes directes de diagonalisation d'une matrice symétrique sont aussi décomposées en deux phases. La matrice est d'abord mise sous forme tridiagonale, puis celle-ci est diagonalisée. Ici encore, le choix de l'algorithme ne porte que sur la deuxième phase. Dans le cas Divide-and-Conquer, on peut négliger le nombre d'opérations de la deuxième phase devant la première. Ce qui donne environ $9\,n^3$ opérations pour la diagonalisation QR et $3n^3$ opérations pour la diagonalisation Divide-and-Conquer. Ces estimations correspondent à peu près aux performances observées en pratique sur des matrices quelconques [95]. On représentera une diagonalisation par $\kappa_D\,n^3$.

C.2.1 Initialisation

La phase d'initialisation a été décrite dans la section 5.2.2.2.

Elle est composée de deux opérations consommatrices de CPU et d'une suite de quelques étapes locales :

- la transformation des matrices locales, dans le cas d'une réorthogonalisation locale de S,
- l'initialisation des C_i par diagonalisation des matrices H_i.

Transformation des matrices locales. Pour chaque domaine i, on effectue les opérations suivantes :

(1) Factorisation de Cholesky de la matrice S_i (*dpptrf*) : $\frac{1}{3}n^3$ opérations,

(2) Inversion du facteur de Cholesky obtenu en (1) (*dtptri*) : $\frac{1}{2}n^3$ opérations,

(3) Transformation de la matrice H_i : deux produits d'une matrice symétrique par une matrice triangulaire (*dtrmm*) : $2n^3$ opérations,

(4) Transformation de la matrice $S^{(ij)}$ pour tous les voisins j, tels que $j > i$: deux produits de matrices pleines par des matrices triangulaires (*dtrmm*) : $2n^3$ opérations.

Au total, on a donc environ $(3\,p + \overline{p})\,n^3$ opérations.

Initialisation des C_i On n'a pas de contraintes de mémoire dans cette phase du code. On diagonalise donc les matrices H_i par l'algorithme Divide-and-Conquer dans lequel le nombre d'opérations est déterminé principalement par l'étape de transformation en matrice tridiagonale : $\frac{8}{3}\,n^3$ opérations.

Au total, l'initialisation représente : $\frac{8}{3}\,p\,n^3$.

C.2.2 Etape locale

On balaye ci-dessous les différentes phases de la résolution du problème local sur le domaine i :

(1) Construction de la matrice (4.5) : calcul du produit : $S^{(ij)}C_j$ pour tous les domaines voisins de i : $2\,n^2\,m$ opérations pour chaque voisin, donc $2\,n^2\,M_1$ opérations au total,

(2) Décomposition en valeurs singulières de la matrice (4.5) : $4\,n^2\,M_1 + \kappa_{SVD}\,n^3$ opérations d'après l'expression donnée en préambule, en supposant $M_1 > n$. On note, comme dans le chapitre 4, $\tilde{m}$ la dimension du noyau de cette matrice. On a $m < \tilde{m} < n$.

(3) Projection de H_i sur $\mathcal{V}$: calcul de $V^t H_i V$, ce qui représente $4\,n^2\,\tilde{m}$ opérations.

(4) Diagonalisation de la matrice obtenue en (3) : $\kappa_D\,\tilde{m}^3$ opérations.

(5) Produit des vecteurs propres par V : $2\,n\,\tilde{m}^2$ opérations.

Si on totalise le nombre d'opérations ci-dessus :

$$2\,n^2\,M_1 + \kappa_{SVD}\,n^3 + 4\,n^2\,\tilde{m} + \kappa_D\,\tilde{m}^3 + 2\,n\,\tilde{m}^2$$

les termes étant rangés par ordre décroissant. Aucun de ces termes n'est négligeable devant les autres dans tous les cas de figure. Pour donner une expression un peu plus simple du nombre d'opérations dans l'étape locale, mais plus grossière, on peut cependant majorer $\tilde{m}$ par n et M_1 par $2\,n$.

Le nombre d'opérations pour une étape locale complète est donc de la forme : $K_{loc}\,p\,n^3$, avec $K_{loc} = 10 + \kappa_{SVD} + \kappa_D$.

Suivant la méthode choisie, on a vu que κ_D passe de 3 à 9, ce qui est significatif dans K_{loc} : on peut donc s'attendre à observer une nette influence du choix des algorithmes sur le temps CPU de résolution d'une étape locale.

C.2.3 Etape globale

On balaye ci-dessous les différentes phases de la résolution du problème local sur *un* couple de domaines voisins (i, j).

Le calcul des matrices B_i. On reprend les notations du chapitre 5 p. 85. Le calcul d'une matrice B_i est divisé en 7 opérations importantes :

(1) Calcul de la matrice $S^{(i,i+1)}\,\widetilde{C}^k_{i+1}$: $2\,n^2\,m$ opérations.

(2) Décomposition en valeurs singulières : $4\,n\,m^2 + \kappa_{SVD}\,m^3$ opérations.

(3) Calcul de la matrice : $\left[(S^{(i,j)}\,\widetilde{C}^k_j)_{j\in\{\text{domaines recouvrant } i \text{ et } i+1\}}\right]$: $2\,n^2\,M_2$ opérations.

(4) Décomposition en valeurs singulières (sous l'hypothèse $M_2 > n$) : $4\,n^2\,M_2 + \kappa_{SVD}\,n^3$ opérations.

(5) Calcul de la matrice $Q^t_A Q_B$: $l_A n l_B$ opérations

(6) Décomposition en valeurs singulières : $4\,l_A^2\,l_B + \kappa_{SVD} l_A^3$

(7) Produit de Q_A par les vecteurs singuliers sélectionnés : $n l_A l_B$ opérations.

On peut écrire les relations suivantes entre l_A et l_B : $l_B > l_A$, $n > l_B > \tilde{m}$.

Au total, pour le calcul des deux matrices, en négligeant les termes en l_A et les termes en m^2, on obtient :

$$12\,n^2\,M_2 + 2\,\kappa_{SVD}\,n^3 + 4\,n^2\,m.$$

Calcul de la direction de descente. Le calcul du gradient et des matrices composant le hessien coûte (voir annexe 1) : $4n^2 l$ opérations.

La résolution du système linéaire est faite par une méthode itérative. On ne peut prévoir le nombre d'itérations (qu'on note κ_{it}), mais on sait que le coût de chaque itération est égal au nombre d'opérations d'un produit matrice vecteur, soit $6\,l^3$ opérations.

Le calcul de la direction de descente représente donc : $4n^2 l + 6\,\kappa_{it}\,l^3$. Les tests numériques en "1D" ont montré que l est assez petit par rapport à n (5 à 10 fois plus faible) et que le nombre d'itérations κ_{it} est assez variable : de l'ordre de 20 en général, il peut monter jusqu'à quelques centaines. Finalement, le coût de calcul de la direction de descente reste faible par rapport au coût de calcul des matrices B_i.

Recherche linéaire. Donnons ici quelques informations sur l'implémentation de la recherche linéaire. Comme on applique la méthode de Newton en $U = 0$, la direction de descente U_d étant fixée, on cherche le réel γ minimisant $f(\gamma U_d)$.

Pour deux blocs notés 1 et 2, on a :

$$\overline{C_1}(\gamma) = \widetilde{C}_1 + \gamma\, B_2 U_d\Big(B_1^t T^t \widetilde{C}_1\Big) \tag{C.8}$$

$$\overline{C_2}(\gamma) = \widetilde{C}_2 - \gamma\, B_1 U_d^t\Big(B_2^t T \widetilde{C}_2\Big). \tag{C.9}$$

Les facteurs $B_2 U_d\Big(B_1^t T^t \widetilde{C}_1\Big)$ et $B_1 U_d^t\Big(B_2^t T \widetilde{C}_2\Big)$ sont calculés une seule fois pour toutes les itérations de recherche linéaire, ce qui coute $2\,l^2\,(n+m)$ opérations.

A chaque itération de la recherche linéaire, on fait les opérations suivantes :

(1) Calcul des matrices $\overline{C_1}(\gamma)$ et $\overline{C_2}(\gamma)$: $4\,n\,m$ opérations.

(2) Orthogonalisation des colonnes de ces matrices (méthode de Householder) : $4nm^2$ opérations.

(3) Calcul de l'énergie de chaque bloc : $4\,n^2\,m$ opérations.

Au total, si on note κ_{lin} le nombre d'itérations de la recherche linéaire (qui, d'après les tests numériques faits en "1D" est rarement supérieur à 4), on a au total :

$$2\,l^2\,(n+m) + 4\,\kappa_{lin}\,(n^2\,m + n\,m^2 + n\,m),$$

expression dans laquelle on peut ne retenir que : $4\,\kappa_{lin}\,n^2\,m$

Total des opérations pour l'étape globale. Le calcul des matrices B_i est dominant, donc pour une étape globale complète :

$$\overline{p}\,\left(12\,n^2\,M_2 + 2\,\kappa_{SVD}\,n^3\right).$$

Cette expression peut être écrite, en prenant $M_2 \sim n$: $K_{glo}\,\overline{p}\,n^3$, avec $K_{glo} = 12 + 2\,\kappa_{SVD}$.

C.2.4 Nombre d'opérations pour une itération de MDD

Chacune des deux étapes est en $O(n^3)$. La phase d'initialisation complète (*i.e.* la phase d'initialisation décrite en 5.2.2.2) représente un peu plus qu'une itération de MDD.

La résolution d'un problème local coute à peu près le même nombre d'opérations que la résolution d'un problème global, mais l'étape globale complète représente sensiblement plus d'opérations puisqu'il faut résoudre pour tous les couples de domaines voisins. Si on se réfère encore une fois à une répartition cartésienne des domaines, la relation $\overline{p} \approx \alpha p$ avec $\alpha = 1$ en 1D, $\alpha = 4$ en 2D et $\alpha = 13$ en 3D, donne l'évolution du poids relatif des étapes locales et globales dans le nombre d'opérations. On observe effectivement sur la répartition du temps CPU dans nos tests numériques que les deux étapes ont à peu près le même poids en "1D". Dans un calcul "3D", l'étape globale représenterait ainsi plus de 90% du nombre d'opérations.

Bibliographie

Références en Chimie Computationnelle

Ouvrages généraux

[1] E. Cancès, M. Defranceschi, W. Kutzelnigg, C. Le Bris, and Y. Maday (2003), *Computational Quantum Chemistry : a Primer*, in : Handbook of Numerical Analysis, Special volume, Computational Chemistry, volume X, C. Le Bris guest editor, Ph. G. Ciarlet Editor, North-Holland.

[2] E. Cancès, C. Le Bris, Y. Maday ; *Méthodes Mathématiques en Chimie Quantique*, Mathématiques & Applications, 53, Springer.

[3] R.M. Martin (2004), *Electronic Structure. Basic Teory and Practical Methods*, Cambridge University Press.

[4] R.G. Parr and W. Yang (1989), *Density Functional Theory of Atoms and Molecules*, Oxford Univeristy Press.

[5] A. Szabo and N. Ostlund (1982), *Modern Quantum Chemistry : An Introduction to Advanced Electronic Structure Theory*, MacMillan.

Thèses

[6] M. Barrault. *Développement de méthodes rapides pour le calcul de structures électroniques*, thèse de l'Ecole Nationale des Ponts et Chaussées, 2005.

[7] C.M. Goringe (1995), *D. Phil Thesis*, Oxford University.

Articles

[8] E. Anglada, E. Artacho,J.M. Junquera and J.M. Soler (2002), *Systematic generation of finite-range atomic basis sets for linear-scaling calculations*, Phys. Rev. B *66*, 205101-205104.

[9] K. Babu, S. Gadre (2003), *Ab initio quality one-electron properties of large molecules : Development and testing of molecular tailoring approach*, J. Comp. Chem *24* 484-495.

[10] R. Baer et al. (2003), *Improved Fermi operator expansion methods for fast electronic calculations*, J. Chem. Phys. *119*, 4117-4125.

[11] M. Barrault, E. Cancès, W. W. Hager, C. Le Bris. (2007) Multilevel domain decomposition for electronic structure calculations. *J. Comp. Phys.*, *222*, 86-109.

[12] G. Bencteux, M. Barrault, E. Cancès, W. W. Hager and C. Le Bris (2008) *Domain decomposition and electronic structure computations : a promising approach* in : Partial Differential Equations. Modeling and Numerical Simulation, 147-164, Roland Glowinski, Pekka Neittaanmaki (eds.), Springer.

[13] X. Blanc (2000), *A Mathematical insight into ab initio simulation of the solid phase* in : Mathematical methods and models for ab initio quantum chemistry. Lecture Notes in Chemistry Vol 74, pp 133-158, M. Defranceschi et C. Le Bris (Ed.), Springer, 2000.

[14] D.R. Bowler et al. (1997), *A comparison of linear scaling tight binding methods*, Modelling Simul. Mater. Sci. Eng. *5*, 199-222.

[15] D. Bowler, T. Miyazaki and M. Gillan (2002), *Recent progress in linear scaling ab initio electronic structure theories*, J. Phys. Condens. Matter *14*, 2781-2798.

[16] D. Bowler, R. Choudhury, M. Gillan and T. Miyazaki (2006), *Recent progress with large-scale ab initio calculations : the CONQUEST code*, phys. stat. sol. b *243*, 989.

[17] S.F. Boys (1950), *Electronic wavefunction I. A general method of calculation for the stationary states of any molecular system*, Proc. Roy. Soc. A 200, 542-554.

[18] S.F. Boys (1960), *Construction of Some Molecular Orbitals to Be Approximately Invariant for Changes from One Molecule to Another*, Rev. Mod. Phys. *32* 296-299.

[19] E. Cancès and C. Le Bris (2000), *Can we outperform the DIIS approach for electronic structure calculations*, Int. J. Quantum Chem. *79* 82-90.

[20] M. Challacombe (2000), *Linear scaling computation of the Fock matrix, V. Hierarchical cubature for numerical integration of the exchange-correlation matrix*, J. Chem. Phys. *113*, 10037-10043.

[21] J.R. Chelikowsky, Y. Saad and N. Trouiller (1994), *Finite difference pseudopotential method : electronic structure calculations without a basis*, Phys. Rev. Lett. *72*, 1240-1243.

[22] L. Colombo and S. Goedecker (1994), *Efficient linear scaling algorithm for Tight-Binding molecular dynamics*, Phys.Rev. Lett. *73*, 122-125.

[23] I. Dabo, B. Kozinsky, N.E. Singh-Miller and N. Marzari (2008), *Electrostatics in periodic boundary conditions and real-space corrections*, Phys. Rev. B 77, 115139.

[24] I.P. Daykov, T.A. Arias and T.D. Engeness *Robust Ab Initio Calculation of Condensed Matter : Transparent Convergence through Semicardinal Multiresolution Analysis*, Phys. Rev. B *90*, 216402.

[25] C. Edmiston and K. Ruedenberg (1965) *Localized Atomic and Molecular Orbitals. II*, J. Chem. Phys. *43*, 4916-4919.

[26] J.L. Fattebert and J. Bernholc (2000), *Towards grid-based O(N) density-functional theory methods : Optimized nonorthogonal orbitals and multigrid acceleration*, Phys. Rev. B *62*, 1713-1722.

[27] G. Feng and T.L. Beck (2006), *Nonlinear multigrid eigenvalue solver utilizing nonorthogonal localized orbitals* phys. stat. sol. *243*, 1054-1062.

[28] H. Feng, J. Bian, L. Li, W. Yang, (2004), *An efficient method for constructing nonorthogonal localized molecular orbitals*, J. Chem. Phys. *120*, 9458-9466.

[29] M.J. Frisch, G.W. Trucks, H.B. Schlegel, G.E. Scuseria, M.A. Robb, J.R. Cheeseman, V.G. Zakrzewski, J.A. Montgomery, R.E. Stratmann, J.C. Burant, S. Dapprich, J.M. Millam, A.D. Daniels, K.N. Kudin, M.C. Strain, O. Farkas, J. Tomasi, V. Barone, M. Cossi, R. Cammi, B. Mennucci, C. Pomelli, C. Adamo, S. Clifford, J. Ochterski, G.A. Petersson, P.Y. Ayala, Q. Cui, K. Morokuma, D.K. Malick, A.D. Rabuck, K. Raghavachari, J.B. Foresman, J. Cioslowski, J.V. Ortiz, B.B. Stefanov, G. liu, A. Liashenko, P. Piskorz, I. Kpmaromi, G. Gomperts, R.L. Martin, D.J. Fox, T. Keith, M.A. Al-Laham, C.Y. Peng, A. Nanayakkara, C. Gonzalez, M. Challacombe, P.M.W. Gill, B.G. Johnson, W. Chen, M.W. Wong, J.L. Andres, M. Head-Gordon, E.S. Replogle and J.A. Pople, Gaussian 98 (Revision A.7), Gaussian Inc., Pittsburgh PA 1998.

[30] G. Galli and F. Mauri (1994), *Electronic structure calculations and molecular dynamics simulations with linear system size scaling*, Physical Review B *50*, 4316-4326.

[31] G. Galli (2000), *Large scale electronic structure calculations using linear scaling methods*, Phys. Stat. Sol. B *217*, 231-249.

[32] A. Genoni, K. Merz, M. Sironi, (2008), *A Hylleraas functional based perturbative technique to relax the extremely localized molecular*, J. Chem. Phys. *129*, 054101.

[33] L. Genovese, A. Neelov, S. Goedecker, T. Deutsch, S. Ghasemi, A. Will, D. Caliste, O. Zilberberg, M. Rayson, A. Bergman and R. Schneider(2008), *Daubechies wavelets as a basis set for density functional pseudopotential calculations*, J. Chem. Phys. *129*, 014109.

[34] A. Gibson, R. Haydock and J.P. Lafemina (1993), *Ab initio electronic-structure computations with recursion methods*, Phys. Rev. B *47*, 9229-9237.

[35] S. Goedecker (1998), *The decay properties of the finite temperature density matrix in metals*, Physical Review B *58*, 3501-3502.

[36] S. Goedecker (1999), *Linear scaling electronic structure methods*, Rev. Mod. Phys. *71*, 1085-1123.

[37] S. Goedecker (2003), *Linear Scaling Methods for the Solution of Schrödinger's Equation*, in : Handbook of Numerical Analysis, Special volume, Computational Chemistry, volume X, C. Le Bris guest editor, Ph. G. Ciarlet Editor, North-Holland.

[38] N. Govind, Y. Wang, and E.A. Carter (1999) *Electronic-structure calculations by first-principles density-based embedding of explicitly correlated systems* J. Chem. Phys. *110*, 7677.

[39] R.J. Harrison, G.I. Fann, T. Yanai, Z. Gan and G. Beylkin (2004), *Multiresolution quantum chemistry : Basic theory and initial applications*, J. Chem. Phys, *121*, 11587 .

[40] V. Heine (1980), *Solid State physics : Advances in Research and Aplications*, in : F. Seitz, C. Turnbull, H. Ehrenreich (Eds), vol 35, Academic Press, New-York.

[41] R. Hoffmann (1963), *An Extended Hückel Theory. I. Hydrocarbons* , J. Chem. Phys *39*, 1397.

[42] C. Jacob, J. Neugebauer, L. Visscher (2008), *A flexible implementation of frozen-density embedding for use in multilevel simulations*, Journal of Computational Chemistry *29* 1011-1018.

[43] B. Jansik, S. Host, P. Jorgensen, J. Olsen, T. Helgaker (2007), *Linear-scaling symmetric square-root decomposition of the overlap matrix*, J Chem Phys *126* 124104.

[44] W. Kohn (1959), *Analytic properties of Bloch waves and Wannier functions*, Phys. Rev. *115*, 809-821.

[45] W. Kohn (1996), *Density functional and density method scaling linearly with the number of atoms*, Phys. rev. lett. *76*, 3168-3171.

[46] J. Korchowiec, F. L. Gu, A. Imamura, B. Kirtman, Y. Aoki (2005), *Elongation method with cutoff technique for linear SCF scaling*, Int. J. of Quant. Chem. *102* 785-794.

[47] X.-P. Li, R.W. Nunes and D. Vanderbilt (1993), *Density-matrix electronic structure method with linear system size scaling*, Phys. Rev. B *47*, 10891-10894.

[48] N. Marzari, D. Vanderbilt (1997), *Maximally localized generalized Wannier functions for composite energy bands*, Physical review B *56*, 12847-12865.

[49] P. Maslen, C. Ochsenfeld, C.A. White, M.S. Lee and M. Head-Gordon (1998), *Locality and Sparsity of Ab Initio One-particle Density Matrices and Localized Orbitals*, J. Chem. Phys. *102*, 2215-2222.

[50] T. Miyazaki, D.R. Bowler, R. Choudhury and M.J. Gillan (2007), *Density functional calculations of Ge(105) : Local basis sets and O(N) methods*, Phys. Rev. B *76*, 115327.

[51] A.M. Niklasson (2002), *Expansion algorithm for the density matrix*, Phys. Rev. B *66*, 155115-155120.

[52] R. Nunes, D. Vanderbilt (1994), *Generalization of the density-matrix method to a nonorthogonal basis*, Phys. Rev. B *50*, 17611-17614.

[53] P. Ordejón, D.A. Drabold, M.D. Grumbach and R.M. Martin (1993), *Unconstrained minimization approach for electronic computations that scales linearly with system size*, Phys. Rev. B *48*, 14646-14649.

[54] P. Ordejón, D.A. Drabold, R.M. Martin and M.D. Grumbach (1995), *Linear system-size scaling methods for electronic-structure calculations*, Phys. Rev. B *51*, 1456-1476.

[55] P. Ordejón (1998), *Order N tight binding methods for electronic structure and molecular dynamics*, Computational Materials Science *12*, 157-191.

[56] J.E. Pask and P.A. Sterne (2005), *Finite elements in ab initio electronic-structure calculations*, in : Handbook of Materials Modeling, S. Yip (ed.), p. 423, Springer, Dordrecht.

[57] A. Palser and D. Manopoulos (1998), *Canonical purification of the density matrix in electronic structure theory*, Phys. Rev. B *58*, 12704-12711.

[58] J. Pipek and P. Mezey (1989), *A fast intrinsic localization procedure applicable for ab initio and semiempirical linear combination of atomic orbital wave functions* J. Chem. Phys. *90* 4916 .

[59] M.J. Rayson, P.R. Briddon (2008), *Rapid iterative method for electronic-structure eigenproblems using localised basis functions*, Computer Physics Communications, *178*, 128-134.

[60] M.J. Rayson (2007), *Low-complexity method for large-scale self-consistent ab initio electronic structure calculations without localization*, Physical Review B *75*, 153203.

[61] E.H. Rubensson and P. Salek (2005) *Systematic sparse matrix error control for linear scaling electronic structure calculations*, J Comput Chem, *26* 1628-1637.

[62] E.H. Rubensson, E. Rudberg and P. Salek (2008) *Density matrix purification with rigourous error control*, J Comput Chem, *26* 1628-1637.

[63] E.H. Rubensson, N. Bock, E. Holmström and A. Niklasson (2008), *Recursive inverse factorization*, J. Chem. Phys. *128*, 104105.

[64] P. Salek, S. Host, L. Thogersen, P. Jorgensen, P. Manninen, J. Olsen, B. Jansik, S. Reine, F. Pawlowski, E. Tellgren, T. Helgaker, S. Coriani (2007) *Linear-scaling implementation of molecular electronic self-consistent field theory*, J. Chem. Phys. *126* 114110.

[65] R. Schneider, T. Rohwedder, A. Neelov and J. Blauert, *Direct minimization for calculating invariant subspaces in density functional computations of the electronic structure*, arXiv :0805.1190 (May 2008).

[66] F. Shimojo, R.K. Kalia, A. Nakano and P. Vashishta (2005) *Embedded divide-and-conquer algorithm on hierarchical real-space grids : parallel molecular dynamics simulation based on linear-scaling density functional theory*, Comp. Phys. Comm *167*, 151-164.

[67] F. Shimojo, R.K. Kalia, A. Nakano and P. Vashishta (2008) *Divide-and-conquer density functional theory on hierarchical real-space grids : Parallel implementation and applications*, Physical Review B *77*, 085103.

[68] E. Schwegler, M. Challacombe *Linear scaling computation of the Fock matrix Theor. Chem. Acc.*,104 :344–349, 2000.

[69] S. Schweizer, J. Kussmann, B. Doser and C. Ochsenfeld (2008) *Linear-Scaling Cholesky Decomposition*, J Compu Chem, *29* 1004-1010.

[70] L. Seijo and Z. Barandiarán (2004), *Parallel, linear-scaling building-block and embedding method based on localized orbitals and orbital-specific basis sets*, J. Chem. Phys., *121*, 6698

[71] L. Seijo, Z. Barandiarán and José M. Soler (2007), *Order-N and embedded-cluster first-prinicples DFT calculations using SIESTA/Mosaico* Theoret. Chem. Acc., *118*, 541.

[72] C.-K. Skylaris and P. D. Haynes, (2007), *Achieving plane wave accuracy in linear-scaling density functional theory applied to periodic systems : A case study on crystalline silicon*, J. Chem. Phys. *127*, 164712.

[73] E. Tsuchida (2007), *Augmented Orbital Minimization Method for Linear Scaling Electronic Structure Calculations*, J. Phys. Soc. Jap. *76*, 034708.

[74] E. Tsuchida (2008), *Ab initio molecular dynamics simulations with linear scaling : application to liquid ethanol*, J. Phys. :cond. mat. *20*, 294212.

[75] J. VandeVondele and J. Hutter (2007), *Gaussian basis sets for accurate calculations on molecular systems in gas and condensed phases*, J. Chem. Phys. *128*, 114105-1 – 114105-9

[76] L.-W. Wang, Z. Zhao and J. Meza (2008), *Linear-scaling three-dimensional fragment method for large-scale electronic structure calculations*, Phys. Rev. B *77*, 165113.

[77] V. Weber, J. VandeVondele, J. Hutter and A.M.N. Niklasson (2008), *Direct energy functional minimization under orthogonality constraints*, J. Chem. Phys. *128*, 084113-1 – 084113-9.

[78] V. Weber, J. Hutter (2008), *A smooth l_1-norm sparseness function for orbital based linear scaling total energy minimisation*, J. Chem. Phys. *128*, 064107.

[79] C.A. White, P. Maslen, M.S. Lee and M. Head-Gordon (1997), *The tensor properties of energy gradients within a non-orthogonal basis*, Chem. Phys. Lett. *276*, 133-138.

[80] W. Yang and T. Lee (1995), *A density-matrix divide-and-conquer approach for electronic structure calculations of large molecules*, J. Chem. Phys. *163*, 5674-5678.

Liens vers des codes d'utilisation courante

[81] ABINIT : *http ://www.abinit.org/*.

[82] BigDFT : *http ://www-drfmc.cea.fr/sp2m/L_Sim/BigDFT/index.html*.

[83] CONQUEST : *http ://www.conquest.ucl.ac.uk/*

[84] CPMD : *http ://www.cpmd.org/*.

[85] CRYSTAL : *http ://www.crystal.unito.it/*.

[86] DMol : *http ://www.accelrys.com/cerius2/dmol3.html*.

[87] Gaussian : *http ://www.gaussian.com/*.

[88] OPENMX : *http ://www.openmx-square.org/*.

[89] ONETEP : *http ://www2.tcm.phy.cam.ac.uk/onetep/*.

[90] QUICKSTEP : *http ://cp2k.berlios.de/quickstep.html*.

[91] SIESTA : *http ://www.uam.es/departamentos/ciencias/fismateriac/siesta/*.

[92] VASP : *http ://cms.mpi.univie.ac.at/vasp/*.

Références en Mathématiques appliquées

Ouvrages généraux

[93] Z. Bai, J. Demmel, J. Dongarra, A. Ruhe, H. van der Vorst, editors. (2000), *Templates for the Solution of Algebraic Eigenvalue Problems : A Practical Guide*, SIAM.

[94] A. Björck, (1996), *Numerical Methods for least squares problems*, SIAM.

[95] J.W. Demmel (1997), *Applied Numerical Algebra*, SIAM Press, Philadelphia, PA.

[96] G. Golub and C.F. Van Loan (1996), *Matrix Computations*, The John Hopkins Univ. Press.

[97] J. Nocedal and S.J. Wright (1999), *Numerical Optimization*, Springer, New York.

[98] B.N. Parlett (1980), *The Symmetric Eigenvalue Problem*, Prentice-Hall, Englewood Cliffs, NJ.

[99] G. Strang, (1996), *Linear Algebra and Its Applications*, Thomson, Belmont, CA.,

Articles

[100] P. Arbenz, U.L. Hetmaniuk, R.B. Lehoucq and R.S. Tuminaro, *A comparison of eigensolvers for large-scale 3D modal analysis using AMG-preconditioned iterative methods*, Int. J. Numer. Meth. Engng 64 (2005) 204-236.

[101] M. Benzi and N. Razouk (2007), *Decay bounds and $O(n)$ algorithms for approximating functions of sparse matrices*, ETNA, *28*, 16-39.

[102] C. Bischof, G. Quintana-Orti (1998), *Computing Rank-Revealing QR Factorizations of Dense Matrices*, ACM Transactions on Mathematical Software, *24*, 226-253.

[103] J. Demmel, O. Marques, B.N. Parlett and C. Vömel (2008), *Performance and Accuracy of LAPACK's Symmetric Tridiagonal Eigensolvers*, SIAM J. Scientific Computing, *30*, 1508-1526.

[104] Z. Drmač, Z. Bujanović (2008), *On the Failure of Rank-Revealing QR Factorization Software – A Case Study*, ACM Transactions on Mathematical Software, *35*, art. nř12.

[105] W. Hager and H. Zhang A new conjugate gradient method with guaranteed descent and an efficient line search. *SIAM Journal on Optimization*, 16 (2005), 170-192.

[106] U.L. Hetmaniuk and R.B. Lehoucq, *Multilevel methods for eigenspace computations in structural dynamics*, Proceedings of the 16th International Conference on Domain Decomposition Methods, Courant Institute, New-York, January 12-15, 2005.

[107] C. Le Bris (2005), *Computational chemistry from the perspective of numerical analysis*, Acta Numerica, *14*, 363-444.

[108] C. Paige and M. Saunders (1975), Solution of sparse indefinite systems of linear equations, SIAM J. Numer. Anal., *12*, 617-629.

[109] Y. Zhou, Y. Saad (2008), *Block Krylov Schur method for large symmetric eigenvalue problems*, Numerical Algorithms *47*, 341-359.

[110] D.C. Sorensen (1982), *Newton's method with a model trust region modification*, SIAM J. Numer. Anal. *19*, 409-426.

[111] H. Weyl (1912), *The laws of asymptotic distribution of the eigenvalues of linear partial differential equations*, Math. Ann., *71*, 441-479.

[112] C. Yang, J.C. Meza and L.-W. Wang (2007), *A trust region direct constrained minimization algorithm for the Kohn-Sham equation*, SIAM J. Sci. Comput. *29* 1854-1875.

Références en Informatique : Calcul Haute Performance (HPC)

[113] Almasi, George and Bhanot, Gyan and Chen, Dong and Eleftheriou, Maria and Fitch, Blake and Gara, Alan and Germain, Robert and Gunnels, John and Gupta, Manish and Heidelberg, Philip and Pitman, Mike and Rayshubskiy, Aleksandr and Sexton, James and Suits, Frank and Vranas, Pavlos and Walkup, Bob and Ward, Chris and Zhestkov, Yuriy and Curioni, Alessandro and Andreoni, Wanda and Archer, Charles and Moreira, José and Loft, Richard and

Tufo, Henry and Voran, Theron and Riley, Katherine (2005) *Early Experience with Scientific Applications on the Blue Gene/L Supercomputer*, in :Euro-Par 2005 Parallel Processing, LNCS 2648, 560-570.

[114] E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney and D. Sorensen, *LAPACK users' guide, 3rd edition*, SIAM 1999.

[115] N. Bock, E. Rubensson, P. Salek, A. Niklasson, M. Challacombe (2008), *Cache oblivious storage and access heuristics for blocked matrix-matrix multiplication*, arXiv :0808.1108 (Août 2008).

[116] DR Bowler, T Miyazaki and M Gillan (2001), *Parallel sparse matrix multiplication for linear scaling electronic structure calculations*, Comp. Phys. Comm, *137*, 255-273.

[117] *Intel®Math Kernel Library for Linux. User's Guide.* October 2007.

[118] W. P. Petersen and P. Arbenz. *Introduction to Parallel Computing.* Oxford University Press, 2004.

[119] E. Rubensson, E. Rudberg and P. Salek (2007) *A hierarchic sparse matrix data structure for large-scale Hartree-Fock/Kohn-Sham calculations*, J. Chem. Phys. *28*, 2531-2537.

[120] C. Saravanan, Y. Shao, R. Baer, PN. Ross and M. Head-Gordon (2003), *Sparse matrix multiplications for linear scaling electronic structure calculations in an atom-centered basis set using multiatom blocks*, J. Comp. Chem. *24*, 618-622.

[121] http ://www.top500.org/, site classant les calculateurs les plus puissants (nombre d'opérations par seconde) au monde.