

Institut Jean Le Rond d'Alembert

RAPPORT DE STAGE

Développement d'un algorithme de mesure du champ de vitesse d'un écoulement basé sur la détection de particules

Auteur :

Belkacem Anis OUKIL

Master 2 Sciences pour l'Ingénieur

Mention Mécanique des fluides, fondements et applications

Spécialité Modélisation et Simulation en Hydrodynamique.

Encadrant :

Philippe DRUAULT

*Maître de Conférences (HDR)- Chercheur à l'IJLRA Équipe MPIA
Jean-François KRAWCZYNSKI*

Maître de Conférences - Chercheur à l'IJLRA Équipe CEPT

Institut Jean le Rond d'Alembert

L’Institut Jean le Rond d’Alembert est un laboratoire de recherche scientifique affilié à la Sorbonne Université et au CNRS. Les domaines d’activité sont principalement la mécanique, l’acoustique et l’énergétique. Situé sur le Campus de Jussieu (UPMC) et Saint-Cyr-l’Ecole, le laboratoire regroupe, depuis 2007, 5 équipes de recherches :

- FCIH : Fluides Complexes et Instabilités Hydrodynamiques.
- MPIA : Modélisation, Propagation et Imagerie Acoustique.
- CEPT : Combustion, Énergies Propres et Turbulence.
- LAM : Lutherie, Acoustique, Musique.
- MISES : Mécanique et Ingénierie des Solides et Structures.

Le stage entrepris s’inscrit dans le cadre de l’étude de la turbulence par des mesures optiques et est donc relié aux équipes CEPT et MPIA à travers mes superviseurs.

Remerciements

J’aimerai remercier mes superviseurs, Jean-François KRAWCZYNSKI et Philippe DRUAULT, qui m’ont donné la chance de pouvoir travailler sur un sujet scientifique aussi poussé et intéressant que la turbulence et qui m’ont aidé et encouragé tout le long du stage avec leurs conseils et leurs expériences dans le domaine. En plus de l’ambiance familiale présente durant ce stage avec eux, cela m’a beaucoup enrichie, et me permettra de mieux évoluer dans le monde de la recherche.

Je remercie aussi les stagiaires, scientifiques et personnels du laboratoire qui savent créer une ambiance de travail où l’on aime travailler pour la science.

Cette expérience clos mon master 2 de Modélisation et Simulation en Hydrodynamique, et je ne pouvais pas espérer mieux comme fin non seulement en matière de niveau scientifique mais aussi en termes d’épanouissement.

Résumé

En turbulence, il est important de caractériser le comportement des petites échelles et leurs influences sur la nature de l'écoulement. L'utilisation de techniques de mesure optique non intrusives telle que la PTV permet de capturer l'ensemble des échelles spatio-temporelles dans un but de solidifier les modèles actuelles de turbulence en boîte et de pouvoir approfondir les connaissances en turbulence réelle. Cette technique nécessite la détection des positions de particules, et ce à travers une résolution d'un système linéaire traduisant l'intensité lumineuse réfléchie par les particules illuminées dans une tranche d'écoulement. L'étude qui suit a pour but d'identifier l'algorithme le plus efficace en terme de temps de calcul et de robustesse afin de détecter ces particules et pouvoir compléter et continuer des travaux de recherche sur la caractérisation expérimentale par PTV d'une turbulence issue de l'interaction de jets turbulents.

Abstract

In turbulence, it is important to characterize the behaviour of small scales and their influences on the nature of the flow. The use of non-intrusive optical measurement techniques such as PTV makes it possible to capture all spatio-temporal scales in order to solidify current box turbulence models and to deepen knowledge of real turbulence. This technique requires the detection of particle positions through a resolution of a linear system reflecting the light intensity reflected by the illuminated particles in a flow slice. The following study aims to identify the most efficient algorithm in terms of computation time and robustness to detect these particles and to complete and continue research work on experimental characterization by PTV of a turbulence resulting from the interaction of turbulent jets.

Table des matières

Introduction	1
1 Algorithmes de résolution	2
1.1 Problème mathématique	2
1.2 Différentes classes d'algorithmes	3
1.2.1 Un mot sur les <i>row/column action methods</i>	3
1.2.2 Méthodes <i>Active-set</i>	3
1.2.3 Méthodes itératives	4
1.3 Les algorithmes testés	4
1.3.1 Least Squares Non Negative (LSQNONNEG)	4
1.3.2 Fast Non Negative Least Squares (FNNLS)	5
1.3.3 Brasilai Borwein-Non Negative Least Squares (BB-NNLS)	5
1.3.4 Limited-memory -Broyden Fletcher Goldfarb Shanno- Bounded (L-BFGS-B)	5
1.3.5 Algebraic Reconstruction Technique (ART)	5
1.3.6 Simultaneous Multiplicative Algebraic Reconstruction Technique (SMART)	6
1.4 Autres algorithmes	7
2 Analyse des algorithmes à partir des matrices générées synthétiquement	8
2.1 Critères d'étude	8
2.1.1 Erreur relative L_2	8
2.1.2 Norme L_1	9
2.2 Matrice A générée de façon aléatoire	9
2.2.1 <i>Sparsity</i> de la matrice A et du vecteur solution x	9
2.2.2 Configuration de matrice carrée à matrice très sous-déterminée	9
2.2.3 Résultats et interprétations	10
2.2.3.1 Comparaison du temps de calcul	13
2.2.3.2 Comparaison des normes L_1 et des erreurs relatives L_2	15
2.3 Matrice A bande	16
2.3.1 Matrices bandes à dimension et <i>sparsity</i> de A fixée : LSQNONNEG, FNNLS, BBNLNS et L-BFGS-B	17
2.3.2 Matrices bandes à <i>sparsity</i> de A et x fixée : ART, SMART, LSQNONNEG	17
2.3.2.1 LSQNONNEG	18
2.3.2.2 ART k	19
2.3.2.3 ART rand	20
2.3.2.4 ART sym	21
2.3.2.5 SMART	22
2.3.2.6 Normes L_1 et temps de calculs de LSQNONNEG, ART k,rand,sym, et SMART	25
3 Validation à partir de matrices réalistes issues d'images de particules	26
3.1 Tests sur des images de particules 32×32 pixels	27
3.1.1 Vecteur position x issu des algorithmes comparé au vecteur position exacte - Détection des positions des particules	27
3.1.2 Temps de calcul et normes L_1	33
3.2 Reconstruction du champ de vecteurs vitesses	34
Bilan et perspectives	36

Annexes

Annexe 1

1	Algorithmes
1.1	FNNLS
1.2	BB-NNLS
1.3	L-BFGS-B
1.4	ART k,rand et sym
1.5	SMART

Annexe 2

1	Résultats des tests sur matrice générée aléatoirement
1.1	10%
1.2	20%

Table des figures

1.1	Exemple : cheminement de l'approximation de la solution par ART	6
2.1	Image pixélisé 32x32 d'un champ turbulent à un instant	8
2.2	Evolution du temps de calcul des algorithmes : <i>lsqnonneg</i> , <i>fnnls</i> , <i>bbnnls</i> , <i>lbfgsb</i> en fonction de la <i>sparsity</i> de \mathbf{x} pour chaque matrice (de <i>sparsity</i> fixe 5%)	13
2.3	Evolution du temps de calcul des algorithmes : <i>lsqnonneg</i> , <i>fnnls</i> , <i>bbnnls</i> , <i>lbfgsb</i> en fonction de T-[1-4] (de <i>sparsity</i> fixe 5%) pour chaque <i>sparsity</i> de \mathbf{x}	14
2.4	Évolution du temps de calcul des algorithmes <i>fnnls</i> , <i>bbnnls</i> , <i>lbfgsb</i> normalisés par rapport à <i>lsqnonneg</i> en fonction de T-[1-4] (<i>sparsity</i> fixée à 5%) et pour une <i>sparsity</i> de \mathbf{x} de 0.15%	15
2.5	Exemple d'une matrice bande sous-déterminée 1024×215600 utilisée dans les tests	16
2.6	Résidus LSQNONNEG sur une matrice bande A de <i>sparsity</i> $\sim 4\%$	18
2.7	Solution finale LSQNONNEG sur une matrice bande A de <i>sparsity</i> $\sim 4\%$	18
2.8	Résidus de toutes les itérations (200) ART Kaczmarz - <i>sparsity</i> $\sim 4\%$	19
2.9	Résidus de la solution à l'itération finale ART Kaczmarz - <i>sparsity</i> $\sim 4\%$	19
2.10	Solution finale ART k et LSQNONNEG sur une matrice bande A de <i>sparsity</i> $\sim 4\%$	20
2.11	Résidus de toutes les itérations (200) ART Random - <i>sparsity</i> $\sim 4\%$	20
2.12	Résidus de la solution à l'itération finale ART Random - <i>sparsity</i> $\sim 4\%$	20
2.13	Solution finale ART Rand et LSQNONNEG sur une matrice bande A de <i>sparsity</i> $\sim 4\%$	21
2.14	Résidus de toutes les itérations (200) ART Symetric - <i>sparsity</i> $\sim 4\%$	21
2.15	Résidus de la solution à l'itération finale ART Symetric - <i>sparsity</i> $\sim 4\%$	21
2.16	Solution finale ART Sym et LSQNONNEG sur une matrice bande A de <i>sparsity</i> $\sim 4\%$	22
2.17	Résidus de toutes les itérations (200) SMART - <i>sparsity</i> $\sim 4\%$	22
2.18	Résidus de la solution à l'itération finale SMART - <i>sparsity</i> $\sim 4\%$	23
2.19	Résidus de la solution à l'itération finale LSQNONNEG - 1024×10245 <i>sparsity</i> $\sim 4\%$	23
2.20	Solution finale SMART et LSQNONNEG sur une matrice bande A 1024×10245 de <i>sparsity</i> $\sim 4\%$	23
2.21	Solution finale SMART et LSQNONNEG sur une matrice bande A 1024×10245 de <i>sparsity</i> $\sim 4\%$ - Zoom sur les différences	24
2.22	Solution finale SMART et LSQNONNEG sur une matrice bande A 1024×215600 de <i>sparsity</i> $\sim 4\%$	24
2.23	Solution finale SMART et LSQNONNEG sur une matrice bande A 1024×215600 de <i>sparsity</i> $\sim 4\%$ - Zoom sur une partie des différences	25
3.1	Matrice \mathbf{A} 1001×215600 pour une image 32×32 pixels - PPP5	26
3.2	Images de particules utilisées avec leurs positions exactes ("+" rouge)	27
3.3	Vecteur position exacte des particules pour les deux images	27
3.4	Vecteur position \mathbf{x} (orange) des algorithmes ART sur les deux images PPP5 et PPP10, comparé au vecteur position exacte (bleu)	28
3.5	Position des particules ("o" bleu) détectée par les algorithmes ART sur les deux images PPP5 et PPP10, comparé aux positions exactes ("+" rouge)	29
3.6	Vecteur position \mathbf{x} (orange) de l'algorithme BBNLNS sur les deux images PPP5 et PPP10, comparé au vecteur position exacte (bleu)	30
3.7	Position des particules ("o" bleu) détectée par l'algorithme BBNLNS sur les deux images PPP5 et PPP10, comparé aux positions exactes ("+" rouge)	30
3.8	Vecteur position \mathbf{x} (orange) de l'algorithme L-BFGS-B sur les deux images PPP5 et PPP10, comparé au vecteur position exacte (bleu)	31
3.9	Vecteur position \mathbf{x} (orange) de l'algorithme SMART sur les deux images PPP5 et PPP10, comparé au vecteur position exacte (bleu)	31
3.10	Position des particules ("o" bleu) détectée par l'algorithme SMART sur les deux images PPP5 et PPP10, comparé aux positions exactes ("+" rouge)	32
3.11	Vecteur position \mathbf{x} (orange) de l'algorithme LSQNONNEG sur les deux images PPP5 et PPP10, comparé au vecteur position exacte (bleu)	32

3.12 Position des particules ("o" bleu) détectée par l'algorithme LSQNONNEG sur les deux images PPP5 et PPP10, comparé aux positions exactes ("+" rouge)	33
3.13 Position des particules ("o" bleu) détectée par l'algorithme LSQNONNEG sur les deux images successives, comparé aux positions exactes ("+" rouge)	34
3.14 Position des particules ("o" bleu) détectée par l'algorithme SMART sur les deux images successives, comparé aux positions exactes ("+" rouge) - Densité de 0.08 particules par pixel.	34
3.15 Double vortex de Lamb-Oseen	35
3.16 Champs de vitesses reconstruit à partir des détections des algorithmes LSQNONNEG et SMART	35

Liste des tableaux

1.1	Méthodes de balayage des lignes de A_{ij} pour l'algorithme ART	6
2.1	Format de présentation des résultats	10
2.2	Résultats des algorithmes sur la matrice T-1 (5% <i>sparsity</i> de A)	10
2.3	Résultats des algorithmes sur la matrice T-2 (5% <i>sparsity</i> de A)	11
2.4	Résultats des algorithmes sur la matrice T-3 (5% <i>sparsity</i> de A)	12
2.5	Résultats des algorithmes sur la matrice T-4 (5% <i>sparsity</i> de A)	12
2.6	Norme L_1 de l'algorithme <i>lsqnonneg</i> pour différentes <i>sparsity</i> de x et pour les deux matrices T-3 (500 x 15000) et T-4 (100 x 20000)	15
2.7	Norme L_1 de l'algorithme <i>fnnls</i> pour différentes <i>sparsity</i> de x et pour les deux matrices T-3 (500 x 15000) et T-4 (100 x 20000)	15
2.8	Résultats des algorithmes sur une matrice bande 600×256000 ($\sim 6\%$ <i>sparsity</i> de A)	17
2.9	Norme L_1 et temps de calcul des algorithmes LSQNONNEG, SMART, ART k, et sym pour les matrices bandes 1024×10245 et 1024×215600 de <i>sparsity</i> 4%	25
2.10	Norme L_1 et temps de calcul de l'algorithme LSQNONNEG, SMART, ART rand pour les matrices bandes 1024×10245 et 1024×215600 de <i>sparsity</i> 4%	25
3.1	Temps des calcul et norme L_1 des algorithmes ART et BBNNLS sur les deux images de densité de particules respective PPP5 et PPP10	33
3.2	Temps des calcul et norme L_1 des algorithmes L-BFGS-B, LSQNONNEG et SMART sur les deux images de densité de particules respective PPP5 et PPP10	33
3.3	Résultats des algorithmes sur la matrice T-4 (10%) <i>sparsity</i> de A	
3.4	Résultats des algorithmes sur la matrice T-4 (20%) <i>sparsity</i> de A	

Introduction

Présente dans de nombreux phénomènes naturels, ou industriels, la turbulence du fluide est un comportement complexe encore mal compris. Il est nécessaire de l'étudier afin d'en déceler les caractéristiques à différentes échelles et ainsi pouvoir prédire certains événements, ou encore de l'utiliser. Cette turbulence 'réelle' s'écarte souvent des écoulements turbulents de laboratoire pour lesquels des modèles sont conçus et peuvent être vérifiés.

L'observation (expérimentale et numérique) a montré des dépendances entre grandes et petites échelles dans cette classe d'écoulement. Les petites échelles sont fortement affectées par la nature de l'écoulement turbulent (conditions initiales et conditions limites), affectant au retour l'ensemble de l'écoulement ce qui crée une grande variété de problèmes de difficultés différentes. Il est indispensable de pouvoir capturer l'ensemble des échelles spatio-temporelles dans un but d'une meilleure compréhension de ces écoulements et ainsi pouvoir solidifier les modèles pour les adapter à des cas naturels ou industriels.

Cependant, en écoulement turbulent développé, les techniques et mesures usuelles ne permettent pas encore de bien cerner le problème. Et l'accès aux plus petites échelles reste encore hors de portée. Un des chemins qu'ont pris des travaux de recherche ces dernières années à l'Institut Jean le Rond d'Alembert, est la PTV (Particle Tracking Velocimetry)[18][33][34]. Cette technique expérimentale permet d'étudier l'écoulement sans le modifier (méthode non intrusive). En effet, l'écoulement est ensemencé de particules qui réfléchiront la lumière émise par un laser, sur une caméra où une image sera capturée. Deux images successives permettent de construire un champ de vecteurs vitesse attachés aux particules, c'est la partie "tracking". Avant d'y arriver, les particules doivent être détectées sur chaque image. C'est la partie détection et c'est aussi le cadre sur lequel repose ce stage de fin d'études.

La localisation des particules sur une tranche de l'écoulement revient à estimer la bonne intensité lumineuse réfléchie la particule, ce qui permet d'en déduire ses coordonnées sur l'image pixélisée,. Mathématiquement, les modèles prennent la forme d'un système linéaire. [14] ont développé une méthode de détection de particules sur des images pixélisées et la résolution d'un système linéaire tel qu'une matrice \mathbf{A} (dites matrice poids), multipliée par un vecteur \mathbf{x} traduisant la position des particules donne un vecteur \mathbf{b} donnant l'intensité lumineuse des particules correspondantes, le tout discrétisé en pixel avec une résolution élevée. Le densité de particule élevée et la résolution très précise rendent le problème d'inversion linéaire compliqué car plus il y a de particules plus la matrice est sous-déterminée. Il existe évidemment de nombreux algorithmes d'inversion linéaire, notamment l'algorithme utilisé dans [14], mais peu sont efficaces sur des problèmes tel qu'un problème de PTV à forte densité de particule et à forte résolution.

Le but de ce stage et d'identifier l'algorithme d'inversion à la fois le plus robuste et le plus rapide en temps de calcul. Ce rapport de stage sera donc découpé en plusieurs parties :

- la première partie est une explication des détails mathématique liés au problème, tels les conditions imposées sur les solutions ou des détails concernant le caractère unique d'un tel problème. Ceci suivi d'un rapport sur le travail de recherche des algorithmes appropriés.
- la deuxième partie concerne les différents types de tests effectués sur des cas synthétiques afin de pouvoir comparer les algorithmes et l'influence de certains paramètres
- la troisième et dernière partie a pour but de vérifier et de valider les résultats de la deuxième partie sur matrices issues d'images réelles de particules

Chapitre 1

Algorithmes de résolution

Les algorithmes permettant la résolution du problème physique de détection des positions des particules seront présentés dans ce chapitre. Des détails concernant le problème mathématique seront d'abord expliqués, ainsi que les caractéristiques des solutions recherchées. Puis, nous présenterons les différentes classes d'algorithmes en détaillant leur fonctionnement et leurs différences, et enfin nous listerons les algorithmes issus de la littérature et dédiés à la résolution de notre problème mathématique (la détection des particules) que nous testerons dans le chapitre suivant.

1.1 Problème mathématique

Retrouver l'intensité de lumière réfléchie par les particules [14] i.e leur position dans le champ revient à résoudre un problème inverse d'un système linéaire du type :

$$Ax = b \quad (1.1)$$

où le vecteur $x \in \mathbb{R}^M$ représentera la position inconnue des particules, dont sa multiplication avec la matrice $A \in \mathbb{R}^{N \times M}$ dite matrice poids, devrait donner le vecteur $b \in \mathbb{R}^N$ intensité de l'image enregistrée par la caméra, soit la donnée observable.

Due à sa construction, la matrice poids est équivalente à une matrice *sous-déterminée* de grande dimension, soit $N \ll M$ (*short and fat matrix*). Le problème à résoudre possède donc une infinité de solutions [9], puisqu'il présente beaucoup plus d'inconnues que d'équations. La non unicité de la solution dans une telle configuration de ce système linéaire est un obstacle à l'obtention d'une bonne estimation de la position des particules.

De ce fait, afin de se rapprocher le plus possible de la solution, il est judicieux de considérer certains critères propres au problème physique tel que la non-négativité de la solution recherchée (intensité lumineuse positive) ou encore sa *sparsity* [17] qui est le taux d'éléments non nuls de la matrice ou du vecteur, en effet la solution recherchée doit représenter un champ de particules éparses par intervalles irréguliers, caractéristique de l'écoulement turbulent.

Le problème d'inversion (1.1) est en réalité un problème d'optimisation convexe [13][16], dont le but est de minimiser la fonction $f(x) = Ax - b$, soit :

$$\min \|Ax - b\|_2 \quad (1.2)$$

avec la contrainte de non-négativité $x \geq 0$.

Cette contrainte prise en compte, la méthode d'optimisation n'est autre que la méthode des moindres carrés non négatifs NNLS(*NonNegative Least Square*).

Aussi, afin d'obtenir une solution optimale, le problème NNLS doit satisfaire les conditions d'optimalité de Karush-Kuhn-Tucker [1] telles qu'en considérant le gradient $\nabla f(x) = A^T(Ax - b)$:

$$x \geq 0 \quad (1.3)$$

$$\nabla f(x) \geq 0 \quad (1.4)$$

$$\nabla f(x)^T x = 0 \quad (1.5)$$

Le problème d'optimisation convexe est réduit à trouver un vecteur non négatif satisfaisant $\nabla f(x) = 0$. Imposer une contrainte de non-négrativité est numériquement non trivial [15], car on a affaire à des quantités non linéaires expansives.

Donc pour résumer, le problème de minimisation (1.2) nous procurera une solution caractéristique des positions exactes des particules, et cette solution est recherchée avec contrainte de positivité et de *sparsity*. Néanmoins, étant donné l'utilisation d'intensités lumineuses, un processus de seuillage est nécessaire pour détecter le pixel adéquat. On donnera plus de détails concernant ce seuillage dans le chapitre 3, et mettra en avant son caractère restrictif sur les algorithmes pour détecter les particules puis représenter leur champ de vitesse dans l'écoulement turbulent.

La section suivante vise à présenter et détailler le fonctionnement des différentes classes d'algorithmes d'inversion de ce type de problème.

1.2 Différentes classes d'algorithmes

De tels problèmes sont fréquemment rencontrés en science de l'ingénieur, notamment en traitement du signal et de l'image, et dans le traitement des données multispectrales [8][12][24][35].

Notre configuration est donc connue de la littérature. Notamment la contrainte imposée de non-négrativité semble naturelle dans le cas d'étude sur des mesures d'intensité de pixel, de concentration d'éléments chimiques ou de fréquences, afin d'éviter des résultats physiquement absurdes.

Il existe de nombreux algorithmes visant à inverser des systèmes linéaires pour résoudre des problèmes tels que le NNLS (1.2). On peut grossièrement les diviser en deux classes principales : les méthodes dites *active-set*, et les méthodes itératives.

1.2.1 Un mot sur les *row/column action methods*

Avant de discuter des deux méthodes de résolution de systèmes linéaires, il faut savoir que celles-ci présentent des algorithmes qu'on appelle *row-action methods* ou *column-action methods* [27]. Comme leur nom l'indique, ces méthodes "attaquent" la matrice une ligne / colonne à la fois, en appliquant les conditions d'inégalité ou d'égalité dépendant du problème d'optimisation. Les *row-action methods* requièrent une fonction objective $f(x)$ strictement convexe pour converger vers une solution optimale. Les *row-action methods* sont des types d'algorithmes fréquemment utilisés en tomographie [23][26].

1.2.2 Méthodes *Active-set*

Les méthodes *active-set* sont basées sur la caractérisation et la résolution d'un sous-ensemble de la solution dont les éléments respectent la contrainte imposée [32]. Il existe n contraintes d'inégalité dans le NNLS. La i ème contrainte est considérée *active* si le coefficient de régression correspondant est négatif ou nul lorsqu'aucune contrainte y est imposée, sinon elle est considérée comme *passive*. Pour plus de clarté cette idée peut être reformulée par : si l'ensemble *actif* est connu, la solution au problème NNLS est obtenue en traitant les contraintes *actives* comme des contraintes d'égalité au lieu d'inégalité.

Pour trouver cette solution respectant la contrainte (admissible), un ensemble de coefficients de régression admissibles est obtenu. A chaque étape de l'algorithme, les variables sont identifiées et transférées de l'ensemble *actif* au *passif*, de façon à ce que la fonction objective à minimiser décroît strictement. Au bout d'un nombre d'itérations fini, le véritable ensemble *actif* est obtenu par régression linéaire sur l'ensemble des variables libres (sans contraintes).

En général un algorithme *active-set* est de la forme suivante :

- Initialisation : vecteur de départ admissible (le vecteur 0 par exemple)
- En observant les conditions KKT ((1.3)-(1.5), **répéter** les étapes suivantes jusqu'à optimisation :
- résoudre le problème d'égalité défini par l'ensemble *actif*
- calculer le multiplicateur de Lagrange de l'ensemble *actif* et enlever le sous-ensemble correspondant aux multiplicateurs de Lagrange négatifs jusqu'à satisfaire KKT.
- Fin ou répéter

Cet algorithme procède à la résolution en un nombre fini d'itérations. Au cours de la résolution, dès lors que les conditions KKT sont remplies, l'algorithme sort une solution. Néanmoins, en pratique il n'existe pas de façon d'estimer le nombre d'itérations requis pour atteindre ces conditions. La solution s'améliore au fur et à mesure que les itérations se suivent.

Selon [15] les algorithmes *active-set* dépendent grandement du calcul de l'équation normale $\mathbf{w} = \mathbf{A}^T(\mathbf{b} - \mathbf{Ax})$, ce qui peut les rendre désavantageux pour certaines configurations comme des problèmes mal conditionnés (typiquement notre problème).

1.2.3 Méthodes itératives

Les méthodes itératives [19][20][25][30] sont basées sur l'approximation de la solution \mathbf{x} et le raffinement de celle-ci à partir du vecteur initial, itération après itération, jusqu'à atteindre un certain critère de convergence fixé à l'avance.

1.3 Les algorithmes testés

Ainsi, nous choisissons quelques algorithmes *active-set* et itératifs à tester et à comparer dans le chapitre 2 et 3. [14] utilisent l'algorithme (1) LSQNONNEG. Nous le prendrons comme référence, étant donné sa performance inégalée sur des systèmes sous-déterminés et *sparse* tel que celui issu du problème physique de détection des particules.

1.3.1 Least Squares Non Negative (LSQNONNEG)

L'un des algorithmes *active-set* les plus connus de résolution du NNLS est le *lsqnonneg* issu du livre de C.L.Lawson et R.J.Hanson "Solving Least Squares Problem" [32], implémenté dans MATALB comme référence depuis 2006 [37].

En complément de la forme générale d'un algorithme *active-set*, les détails de l'algorithme de Lawson et Hanson sont en **Algorithm** (1), et ci-dessous l'explication de son fonctionnement :

Algorithm 1 lsqnonneg

```

1 : Input :  $\mathbf{A} \in \mathbb{R}^{N \times M}$ ,  $\mathbf{b} \in \mathbb{R}^N$ 
2 : Output :  $x^* \geq 0$  tel que  $x^* = \operatorname{argmin} \| \mathbf{Ax} - \mathbf{b} \|^2$ 
3 : Initialisation :  $\mathbf{x} = \mathbf{0}$ ,  $\mathbf{w} = \nabla f(x) = \mathbf{A}^T(\mathbf{b} - \mathbf{Ax})$ ,  $P = \emptyset$ ,  $R = \{1, 2, \dots, M\}$ 
4 : ▷  $P$  et  $R$  sont respectivement l'ensemble passif et actif
5 : Répéter
6 :   procédure LSQNONNEG( $A, b$ )
7 :     Si  $R \neq \emptyset \wedge [\max_{i \in R}(w_i) > \text{tolerance}]$ 
8 :        $j = \operatorname{argmax}_{i \in R}(w_i)$ 
9 :       Inclure les indices  $j$  dans  $P$  et les retirer de  $R$ 
10 :       $s_P = [(\mathbf{A}_P)^T \mathbf{A}_P]^{-1} (\mathbf{A}_P)^T \mathbf{b}$  ▷  $\mathbf{A}_P$  est la matrice associée à l'ensemble passif  $P$ 
11 :      Si  $\min(s_P) \leq 0$ 
12 :         $\alpha = -\min_{i \in P}[x_i / (x_i - s_i)]$ 
13 :         $\mathbf{x} = \mathbf{x} + \alpha(\mathbf{s} - \mathbf{x})$ 
14 :        Mise à jour de  $R$  et  $P$ 
15 :         $s_P = [(\mathbf{A}_P)^T \mathbf{A}_P]^{-1} (\mathbf{A}_P)^T \mathbf{b}$ 
16 :         $s_R = \mathbf{0}$ 
17 :       $\mathbf{x} = \mathbf{s}$ 
18 :     $\mathbf{w} = \mathbf{A}^T(\mathbf{b} - \mathbf{Ax})$ 

```

Si l'ensemble R est vide, toutes les contraintes sont passives, et tous les coefficients sont positifs, le problème est donc résolu. Cependant, après initialisation, si les w_j , $j \in R$, sont positifs on se retrouvera avec un coefficient de régression positif après transfert des variables correspondantes à l'ensemble des variables libres. Les variables les plus élevées de w_j sont incluses dans P , et le vecteur intermédiaire \mathbf{s} de régression est calculé. En pratique les éléments de \mathbf{w} ne sont pas testés pour être positifs, mais on restreint leur valeurs à une valeur de *tolérance* pour éviter que les imprécisions numériques ne permettent aux variables non admissibles d'intégrer l'ensemble passif. A chaque itération, si tous les coefficients de régression dans \mathbf{s} sont positifs, le vecteur \mathbf{x} est fixé aux valeurs de \mathbf{s} et le nouveau multiplicateur de Lagrange \mathbf{w} est calculé. Ainsi, la boucle principale (étape 7-16) continue de tourner jusqu'à ce qu'il n'y ait plus de contraintes actives à libérer.

La boucle intérieure débute lorsqu'une nouvelle variable est inclue dans l'ensemble P , car il est naturellement possible que certains des coefficients de régression soient négatifs (étape 11). Quelque part sur le segment $\mathbf{d} + \alpha(\mathbf{s} - \mathbf{d})$, $0 \leq \alpha \leq 1$, existe un intervalle où les inégalités sont respectées. A mesure que l'ajustement décroît $\alpha \rightarrow 1$, il est possible de trouver cette valeur de α qui minimise l'ajustement et retient autant de variables possibles dans l'ensemble passif. Pour la valeur optimale, une ou plusieurs variables seront nulles et

donc retirées de l'ensemble P après mise à jour. Ainsi, les nouvelles solutions sans contraintes sont calculées (étape 15). Les coefficients de régression retirés de P sont remis à zéro, et la boucle intérieure est répétée jusqu'à ce que toutes les variables dont la contrainte n'est pas respectée soient retirée de l'ensemble R .

1.3.2 Fast Non Negative Least Squares (FNNLS)

Fast NNLS est aussi un *active-set* basé sur LSQNONNEG. Les auteurs [7] ont modifié l'algorithme de [32], plus précisément l'équation normale

$$\mathbf{w} = \mathbf{A}^T(\mathbf{b} - \mathbf{Ax}) \text{ qui devient } \mathbf{w} = (\mathbf{A}^T\mathbf{x}) - (\mathbf{A}^T\mathbf{A})\mathbf{b} \quad (1.6)$$

et ainsi par implication le vecteur intermédiaire est aussi modifié

$$\mathbf{s}_P = [(\mathbf{A}^T\mathbf{A})_P]^{-1}(\mathbf{A}^T\mathbf{x})_P \quad (1.7)$$

Ces modifications ont pour but d'accélérer le temps de calcul de l'algorithme. Plusieurs versions de cet algorithme ont été développées (Fast combinatorial NNLS [42], FbNNLS), mais on ne retiendra que celle-ci car elle se rapproche le plus d'une possible résolution d'un système sous-déterminé et *sparse* contrairement aux autres qui traitent des problèmes de grandes dimensions certes, mais des problèmes sur-déterminés ou avec de multiples membre de droite, c'est-à-dire $AX = B$ avec X et B des matrices de grandes dimensions.

1.3.3 Brasilai Borwein-Non Negative Least Squares (BB-NNLS)

Les auteurs de [29] ont prolongé l'algorithme d'optimisation quadratique inconditionnel de [5] [40] afin qu'il puisse traiter la contrainte de non-négativité. BB-NNLS est un algorithme itératif qui propose de modifier le paramètre de descente vers la solution α (présent à l'étape 12 et 13 de l'algorithme (1)) par :

$$\alpha^k = \frac{\langle \nabla f^{k-1}, \mathbf{A}^T \mathbf{A} \nabla f^{k-1} \rangle}{\| \mathbf{A}^T \mathbf{A} \nabla f^{k-1} \|_2} \quad (1.8)$$

L'algorithme (détailé en Annexe) a été essentiellement testé par les auteurs sur des matrices sur-déterminées dont la *sparsity* varie de 0.6647 à 33.5167. [29] l'ont aussi testé sur une matrice sous-déterminée à 0.0336% de *sparsity* avec de bons résultats. Cette dernière information se rajoutant au but de non-négativité de l'algorithme nous pousse à le rajouter à la liste des candidats à la résolution de notre problème.

1.3.4 Limited-memory -Broyden Fletcher Goldfarb Shanno- Bounded (L-BFGS-B)

BFGS (Broyden Fletcher Goldfarb Shanno) est un algorithme itératif qui appartient à la famille des méthodes Quasi-Newton (méthodes utilisées pour trouver les zéros et les maxima-minima locaux d'une fonction). Le but est bien entendu de minimiser une fonction $f(x)$, et une condition nécessaire d'optimalité que BFGS utilise est de trouver un point stationnaire dont le gradient est nul.

Nous utiliserons cet algorithme dans sa version L-BFGS-B (Limite memory - BFGS - Bounded [10]) car L-BFGS est conçu pour traiter des problèmes de grande dimension et L-BFGS-B y implémente la possibilité de borner l'étude, afin de respecter la condition de non-négativité pour notre problème.

1.3.5 Algebraic Reconstruction Technique (ART)

Les *Algebraic reconstruction technique* (ART) sont des *row-action methods* souvent utilisées en tomographie [2][3][11][22][26][36]. Le principe consiste en l'approximation (dans notre cas) de l'intensité de lumière réfléchie dans l'espace de reconstruction où celle-ci est légèrement modifiée pour s'accorder avec la donnée mesurée. Étant un algorithme itératif, à chaque itération la solution calculée est approchée de la solution exacte avec un paramètre de relaxation ω compris entre 0 et 1. Cela peut être décrit par l'équation suivante :

$$x_j^{k+1} = x_j^k + A_i \frac{b_i - A_i^T x_j^k}{\|A_i\|_2} \omega \quad (1.9)$$

En effet, pour un cas basique à 2 variables, 2 équations :

$$A_{11}x_1 + A_{12}x_2 = b_1, \text{ ligne 1}$$

$$A_{21}x_1 + A_{22}x_2 = b_2, \text{ ligne 2}$$

on commence par une solution initiale, projetée sur la ligne 1 procurant une solution qui elle-même sera projetée sur la ligne 2 puis vers la ligne 1 etc jusqu'à atteindre la solution exacte aux deux équations, soit l'intersection des deux lignes comme sur la figure suivante.

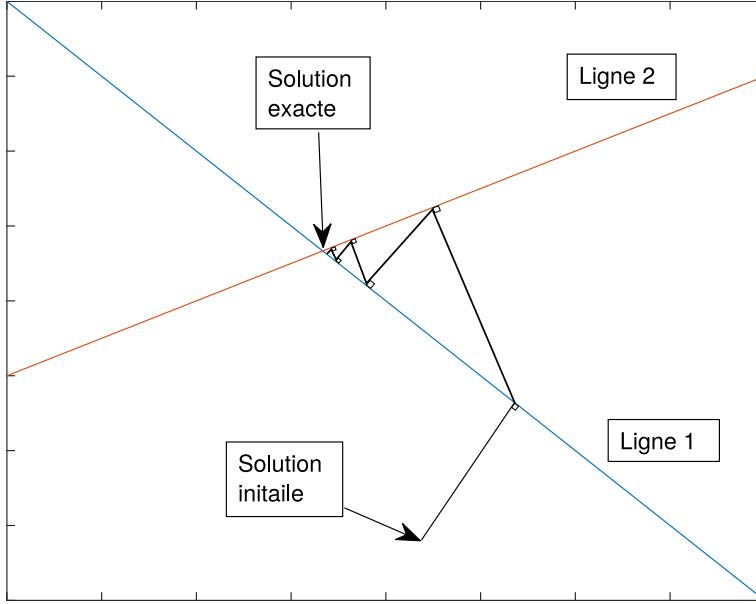


FIGURE 1.1 – Exemple : cheminement de l'approximation de la solution par ART

En d'autres mots, on normalise le vecteur ligne A_i par $(b_i - A_i^T x_j) / \|A_i\|_2$ et on l'ajoute à x_i .

Cette méthode de reconstruction a été originellement proposée par [?] et proposée pour l'imagerie médicale en 1970 par [23] dont il donné le nom de ART.

Le balayage des lignes de A_{ij} peut être de différente nature. L'original est un balayage naturel de la première à la dernière ligne [28], il existe un balayage dit symétrique [4] et un balayage aléatoire [39].

Méthode de balayage	Choix de i et j	Par itération
Original : Kaczmarz	$i = N$	Mise à jour de $N : i = 1, 2, \dots, N$
Symétrique	$i =$ i_0 si $1 \leq i_0 \leq N - 1$ $N - i_0 + 1$ si $N \leq i_0 \leq N - 2$	Mise à jour de $N-1 :$ $i = 1, 2, \dots, N-1$ ou $i = N, N-1, \dots, 2$
Aléatoire	Choix aléatoire de i dans $\{1, 2, \dots, N\}$ avec une probabilité proportionnelle à $\ A_i\ _2$	Mise à jour aléatoire de N

TABLE 1.1 – Méthodes de balayage des lignes de A_{ij} pour l'algorithme ART

Ces algorithmes seront testés à la fin du Chapitre 2, et leurs détails est en Annexe 1.

1.3.6 Simultaneous Multiplicative Algebraic Reconstruction Technique (SMART)

SMART est un dérivé de l'algorithme MART, très proches des algorithmes ART [11][12][22][23]. MART et SMART sont des algorithmes itératifs qui ont pour objectif de produire une solution non-négative de $Ax = b$ par la minimisation de la distance de Kullback-Leiber [31], similaire au carré de la distance Euclidienne, ce qui la rend utile lors de processus d'optimisation :

Pour $a > 0$ et $b > 0$, la distance de Kullback-Leiber est définie telle que :

$$KL(a, b) = a \log\left(\frac{a}{b}\right) + b - a \quad (1.10)$$

avec $KL(a, 0) = +\infty$, $KL(0, b) = b$ et $KL(0, 0) = 0$

Et pour des vecteurs \mathbf{x} et \mathbf{y} :

$$KL(\mathbf{x}, \mathbf{y}) = \sum_{n=1}^N KL(x_n, y_n) \quad (1.11)$$

On la nomme distance, mais en réalité c'est une divergence, et ne remplit pas les conditions usuelles de symétrie d'une distance. Celle-ci est utilisée pour quantifier la différence entre deux distributions de probabilités [31]. Cette quantité est aussi appelée entropie relative.

L'algorithme SMART [12] se limite à la minimisation de cette quantité, par la loi itérative suivante :

$$\log x_j^{k+1} = \log x_j^k + \frac{1}{s_j} \sum_{i=1}^K A_{ij} \log \frac{b_i}{(Ax^k)_i} \quad (1.12)$$

1.4 Autres algorithmes

Il existe évidemment de nombreux autres algorithmes d'inversion de systèmes linéaires $Ax = b$, voici quelques uns qui ont retenu notre attention :

- **FastNNLS** : Version antérieure [7] à l'algorithme choisi plus haut et du même auteur, celle-ci est initialement prévue pour résoudre des systèmes linéaires sur-déterminés et faisant intervenir un grand nombre de membres de droite (Right Hand Side), soit un système du type $AX = B$.

- **FNNLS-b** : Une autre version de l'algorithme choisi, celle-ci cherche d'abord l'ensemble passif et initialise le reste des étapes habituelles du FNNLS avec ce vecteur.

- **Fast Combinatorial NNLS (FcNNLS)** : Dans cette version [42], contrairement au NNLS classique qui choisit l'ensemble passif qui doit être estimé à chaque étape, FcNNLS avec son approche combinatoire identifie les colonnes de A où les ensembles passifs sont identiques, les regroupe et calcule leur pseudoinverse pour résoudre le système. Cependant, le processus d'identification et de regroupement des ensembles passifs rend la méthode très coûteuse.[ref]

- **Least Square Minimal Residual (LSMR)** : Algorithme itératif [19] qui combine plusieurs critères d'arrêts pour définir son point de convergence. Mais les tests entrepris dans [19] ne sont pas adaptés à notre problème puisque les auteurs utilisent des matrices carrées et sur-déterminées.

- **Orthogonal Matching Pursuit (OMP)** : OMP [41] est une extension à l'algorithme Matching Pursuit. OMP est un bon candidat pour une recherche de solutions *sparse* sur un problème sous-déterminé. Cependant la stratégie utilisé dans cet algorithme diffère légèrement de l'approche mathématique utilisée ici ce qui ne permet pas d'imposer mathématiquement une condition de non-négativité, d'autant plus que cet algorithme n'a fait ses preuves que sur des problèmes de petite dimension (de l'ordre de grande de 100×1024).

Ces algorithmes ont été mis de côté après de rapides tests car ne remplissant pas les conditions nécessaires à notre problème, notamment la condition de non-négativité ou encore n'étant pas conçu pour des calculs sur des matrices très sous-déterminées ou *sparse*. Les algorithmes choisis seront testés dans le chapitre suivant. Une première partie n'incluant pas ART (car les matrices auront une structure aléatoire, ce qui n'est pas une caractéristique de la tomographie) initialisera l'étude. Une seconde partie aura pour but de se rapprocher davantage du problème physique avec une génération de matrice bande, ce qui permettra de vérifier l'évolution de l'efficacité des algorithmes choisis.

Chapitre 2

Analyse des algorithmes à partir des matrices générées synthétiquement

Dans ce chapitre, différents types de tests seront établis. Après avoir détaillé les critères qui nous permettront d'étudier les algorithmes testés, nous commencerons par des tests de détection sur des matrices générées aléatoirement sous MATLAB dont les dimensions varieront de carrée à très sous-déterminée, puis celles-ci seront générées de façon à avoir des matrices dites bandes, plus proches de notre problème physique. Ce cheminement nous permettra de cerner le fonctionnement des algorithmes par rapport aux caractéristiques de nos matrices très sous-déterminées (*short and fat matrix*) et *sparse*.

2.1 Critères d'étude

Afin d'avoir une bonne idée des performances des algorithmes étudiés vis-à-vis du problème NNLS, il est judicieux d'établir certains critères permettant de vérifier l'exactitude ou bien l'éloignement de la solution calculée par rapport à la solution exacte.

Les premiers critères logiques auxquels on peut penser sont naturellement la vérification de la positivité des éléments du vecteur solution, la minimisation de la fonction objective (1.2), la *sparsity* en sortie et bien sûr le temps de calcul.

2.1.1 Erreur relative L_2

A cela, on peut rajouter d'autres critères tels que l'erreur relative L_2 :

$$err_{L_2} = \frac{\sum_{i=1}^{i=n} (x(i)^* - x(i))^2}{\sum_{i=1}^{i=n} (x(i)^*)^2} \quad (2.1)$$

Ce calcul d'erreur nous donnera une information concernant l'amplitude des éléments du vecteur calculé. En effet, la solution recherchée possède des amplitudes piquées, ceci est dû à l'intensité de lumière réfléchie par les particules car le but est de détecter leur position sur une image contenant des pixels à différents degrés de gris comme on peut le voir sur l'image ci-dessous.

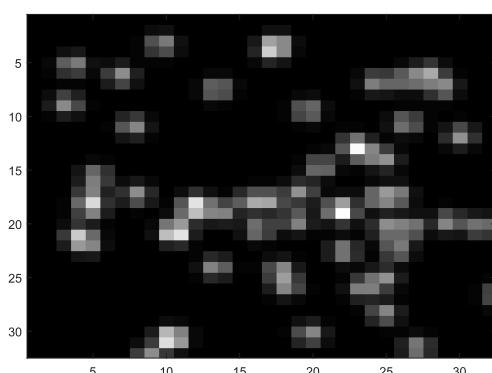


FIGURE 2.1 – Image pixélisé 32x32 d'un champ turbulent à un instant

L'image de la figure (2.1) subira un processus de seuillage qui déterminera la position finale de la particule après détection de l'algorithme. Ce processus sera expliqué plus tard.

Pour une détection de la position des particules, un autre critère pourrait être plus intéressant.

2.1.2 Norme L_1

En effet, un autre critère connu est la norme L1, soit la somme de tous les éléments du vecteur :

$$\text{Norme } L_1 = \|x(i)^* - x(i)\|_{L_1} \quad (2.2)$$

La norme L_1 nous donne une information sur la distance entre les particules, ce qui peut être une bonne vérification de ces algorithmes de détection. Cependant pour ne retenir que la position des particules, il est nécessaire de "binariser" les éléments du vecteurs, ceci permettra de calculer la norme L1 sur des 1 (la particule existe en cette position) et des 0 (aucune particule n'est à cette position) et de délaisser l'amplitude de l'intensité lumineuse.

L'étude de façon complémentaire des résultats d'après ces différents critères nous fournit une bonne idée des performances des algorithmes quant à la détection des positions des particules.

2.2 Matrice A générée de façon aléatoire

2.2.1 Sparsity de la matrice A et du vecteur solution x

Durant tous les tests, sauf précision, la matrice **A** sera initialisée avec 20%, 10% et 5% d'éléments non nuls. Ainsi on pourra suivre l'évolution des performances des algorithmes compte tenue de la *sparsity*, étant donné l'importance de ce paramètre dans ce type de problème, puisque celui-ci allège ou pas les calculs. Le choix de ce pourcentage d'éléments non nuls n'est pas anodin, en effet dans la partie des tests sur des matrices issues des images réelles, celles-ci seront à environ 4% à 6% d'éléments non nuls.

Aussi, les particules étant éparpillées de façon irrégulière et ne remplissant pas tout le champ turbulent, il est logique de rechercher un vecteur solution *sparse*. C'est pour cette raison que nous varierons aussi la *sparsity* du vecteur **x** : 2%, 1%, 0.5% et 0.15% (ce dernier se rapprochant le plus de la solution exacte sur une image 32x32). Il faudra aussi vérifier que cette *sparsity* reste inchangée, car certains algorithmes auront tendance à la modifier pendant le calcul.

2.2.2 Configuration de matrice carrée à matrice très sous-déterminée

On procède ainsi à une batterie de tests en générant les matrices **A** aléatoirement, dans un premier temps, complètement différentes de celle du NNLS pour notre problème physique, et puis se rapprochant petit à petit de ce dernier. L'idée première est d'étudier le comportement de ces algorithmes, mais aussi de mettre en évidence des critères qui pourraient éventuellement nous renseigner davantage sur la progression vers la solution recherchée, lorsqu'on se rapproche d'un problème très sous-déterminé.

Les différentes configurations de matrice test sont présentées ci-dessous :

Matrice	T-1	T-2	T-3	T-4
$N \times M$	5000 x 5000	1000 x 10000	500 x 15000	100 x 20000

Le choix des dimensions des matrices est tel qu'on passe de $\frac{N}{M} = 1$ (carrée) à $\frac{N}{M} = 0.005$ (très sous-déterminé). Ceci dit le nombre de lignes et de colonnes n'est pas de l'ordre d'une image 32 x 32 (testée au chapitre 3) qui génère une matrice 1001×215600 . Nous nous contenterons de ces nombres pour ces premiers tests, afin d'étudier le comportement sur des configurations "simples" et nous nous rapprocherons, par la suite en 2.3.1 et encore plus en 2.3.2, de la configuration de l'image réelle.

2.2.3 Résultats et interprétations

Ci-dessous le format de présentation des résultats :

Matrice T-[1-4]	<i>sparsity</i> de \mathbf{x} [2 % 1 % 0.5 % 0.15 %]
	<i>Err</i> : Erreur relative
Algorithme	<i>sparsitysol</i> : <i>Sparsity</i> de la solution
	<i>t</i> : Temps de calcul
	<i>Nbit</i> : Nombre d'itérations finale
	<i>NormeL1</i> : Pourcentage d'éloignement de la solution exacte, norme L1 binarisée

TABLE 2.1 – Format de présentation des résultats

La valeur minimale sera présentée aussi si celle-ci est en dessous de 0. Pour des raisons de fluidité, seulement la *sparsity* 5% de \mathbf{A} sera présentée ci-dessous car cette dernière est la plus proche du problème physique, le reste (20% et 10%) se trouvera en annexe. Plusieurs simulations ont été entreprises (3 par algorithme et par matrice), les résultats obtenus ne divergent pas d'un jet à l'autre. Nous présenteront les résultats que d'un seul jet puisque les 2 autres sont quasi identiques.

T-1 / Sparsity	2 %	1 %	0.5 %	0.15 %
Lsqnonneg	<i>Err</i> : 1.86e-15	<i>Err</i> : 1.81e-15	<i>Err</i> : 1.04e-15	<i>Err</i> : 6.52e-16
	<i>sparsitysol</i> : 2 %	<i>sparsitysol</i> : 1 %	<i>sparsitysol</i> : 0.5 %	<i>sparsitysol</i> : 0.15 %
	<i>t</i> : 3.550 s	<i>t</i> : 1.457 s	<i>t</i> : 0.678 s	<i>t</i> : 0.228 s
	<i>Nbit</i> : 100	<i>Nbit</i> : 50	<i>Nbit</i> : 25	<i>Nbit</i> : 8
	<i>NormeL1</i> : 0 %	<i>NormeL1</i> : 0 %	<i>NormeL1</i> : 0 %	<i>NormeL1</i> : 0 %
Fnls	<i>Err</i> : 1.81e-15	<i>Err</i> : 2.08e-15	<i>Err</i> : 9.26e-16	<i>Err</i> : 1.61e-15
	<i>sparsitysol</i> : 2 %	<i>sparsitysol</i> : 1 %	<i>sparsitysol</i> : 0.5 %	<i>sparsitysol</i> : 0.15 %
	<i>t</i> : 3.477 s	<i>t</i> : 1.450 s	<i>t</i> : 0.673 s	<i>t</i> : 0.227 s
	<i>Nbit</i> : 100	<i>Nbit</i> : 50	<i>Nbit</i> : 25	<i>Nbit</i> : 8
	<i>NormeL1</i> : 0 %	<i>NormeL1</i> : 0 %	<i>NormeL1</i> : 0 %	<i>NormeL1</i> : 0 %
BBnnls	<i>Err</i> : 6.01e-04	<i>Err</i> : 6.32e-04	<i>Err</i> : 6.04e-04	<i>Err</i> : 9.87e-04
	<i>sparsitysol</i> : 100 %	<i>sparsitysol</i> : 100 %	<i>sparsitysol</i> : 100 %	<i>sparsitysol</i> : 100 %
	<i>t</i> : 4.430 s	<i>t</i> : 3.803 s	<i>t</i> : 3.427 s	<i>t</i> : 2.890 s
	<i>Nbit</i> : 106	<i>Nbit</i> : 90	<i>Nbit</i> : 82	<i>Nbit</i> : 70
	<i>NormeL1</i> : 98 %	<i>NormeL1</i> : 99 %	<i>NormeL1</i> : 99.5 %	<i>NormeL1</i> : 99.840 %
	min : 5.0e-06	min : 3.1e-06	min : 1.5e-06	min : 4.4e-07
L-bfgs-B	<i>Err</i> : 9.01e-08	<i>Err</i> : 3.60e-07	<i>Err</i> : 3.21e-07	<i>Err</i> : 5.23e-07
	<i>sparsitysol</i> : 6.18 %	<i>sparsitysol</i> : 83.3 %	<i>sparsitysol</i> : 31.42 %	<i>sparsitysol</i> : 0.44 %
	<i>t</i> : 14.290 s	<i>t</i> : 12.239 s	<i>t</i> : 11.774 s	<i>t</i> : 11.427 s
	<i>Nbit</i> : 59	<i>Nbit</i> : 48	<i>Nbit</i> : 46	<i>Nbit</i> : 44
	<i>NormeL1</i> : 4.180 %	<i>NormeL1</i> : 82.3 %	<i>NormeL1</i> : 30.920 %	<i>NormeL1</i> : 0.280 %

TABLE 2.2 – Résultats des algorithmes sur la matrice T-1 (5% *sparsity* de \mathbf{A})

T-2 / Sparsity	2 %	1 %	0.5 %	0.15 %
Lsqnonneg	<i>Err</i> : 1.82e-15 <i>sparsity sol</i> : 2.01 % t : 5.212 s <i>Nbit</i> : 260 <i>Norme_{L1}</i> : 0.01 %	<i>Err</i> : 1.25e-15 <i>sparsity sol</i> : 1 % t : 1.275 s <i>Nbit</i> : 100 <i>Norme_{L1}</i> : 0.09 %	<i>Err</i> : 1.26e-15 <i>sparsity sol</i> : 0.5 % t : 0.590 s <i>Nbit</i> : 50 <i>Norme_{L1}</i> : 0 %	<i>Err</i> : 5.15e-16 <i>sparsity sol</i> : 0.15 % t : 0.176 s <i>Nbit</i> : 15 <i>Norme_{L1}</i> : 0 %
	<i>Err</i> : 1.64e-15 <i>sparsity sol</i> : 2.41 % t : 329.584 s <i>Nbit</i> : 241 <i>Norme_{L1}</i> : 0.09 % min : -2.8e-15	<i>Err</i> : 1.06e-15 <i>sparsity sol</i> : 1 % t : 1.255 s <i>Nbit</i> : 100 <i>Norme_{L1}</i> : 0 %	<i>Err</i> : 1.00e-15 <i>sparsity sol</i> : 0.5 % t : 0.571 s <i>Nbit</i> : 50 <i>Norme_{L1}</i> : 0 %	<i>Err</i> : 4.18e-16 <i>sparsity sol</i> : 0.15 % t : 0.167 s <i>Nbit</i> : 15 <i>Norme_{L1}</i> : 0 %
	<i>Err</i> : 7.98e-02 <i>sparsity sol</i> : 12.61 % t : 7.146 s <i>Nbit</i> : 395 <i>Norme_{L1}</i> : 10.73 %	<i>Err</i> : 3.97e-03 <i>sparsity sol</i> : 21.31 % t : 3.86 s <i>Nbit</i> : 215 <i>Norme_{L1}</i> : 20.31 %	<i>Err</i> : 3.25e-03 <i>sparsity sol</i> : 12.28 % t : 2.112 s <i>Nbit</i> : 115 <i>Norme_{L1}</i> : 11.8 %	<i>Err</i> : 1.11e-03 <i>sparsity sol</i> : 28.07 % t : 1.039 s <i>Nbit</i> : 58 <i>Norme_{L1}</i> : 27.92 %
	<i>Err</i> : 6.77e-06 <i>sparsity sol</i> : 15.24 % t : 39.738 s <i>Nbit</i> : 418 <i>Norme_{L1}</i> : 13.24 %	<i>Err</i> : 9.02e-07 <i>sparsity sol</i> : 8.95 % t : 16.393 s <i>Nbit</i> : 172 <i>Norme_{L1}</i> : 7.95 %	<i>Err</i> : 4.87e-07 <i>sparsity sol</i> : 5.21 % t : 11.067 s <i>Nbit</i> : 115 <i>Norme_{L1}</i> : 4.71 %	<i>Err</i> : 6.47e-07 <i>sparsity sol</i> : 5.62 % t : 8.548 s <i>Nbit</i> : 89 <i>Norme_{L1}</i> : 5.470 %

TABLE 2.3 – Résultats des algorithmes sur la matrice T-2 (5% *sparsity* de **A**)

T-3 / Sparsity	2 %	1 %	0.5 %	0.15 %
Lsqnonneg	<i>Err</i> : 1.27e+00 <i>sparsity sol</i> : 3.31 % t : 8.924 s <i>Nbit</i> : 522 <i>Norme_{L1}</i> : 4.847 %	<i>Err</i> : 1.10e+00 <i>sparsity sol</i> : 3.34667 % t : 12.375 s <i>Nbit</i> : 614 <i>Norme_{L1}</i> : 3.593 %	<i>Err</i> : 9.28e-16 <i>sparsity sol</i> : 0.5 % t : 0.723 s <i>Nbit</i> : 82 <i>Norme_{L1}</i> : 0 %	<i>Err</i> : 7.32e-16 <i>sparsity sol</i> : 0.153 % t : 0.193 s <i>Nbit</i> : 23 <i>Norme_{L1}</i> : 0 %
	<i>Err</i> : 1.30e+00 <i>sparsity sol</i> : 3.34667 % t : 416.889 s <i>Nbit</i> : 502 <i>Norme_{L1}</i> : 4.72 % min : -5.3e-01	<i>Err</i> : 9.49e-01 <i>sparsity sol</i> : 3.34 % t : 266.51 s <i>Nbit</i> : 501 <i>Norme_{L1}</i> : 3.147 % min : -1.4e+00	<i>Err</i> : 7.43e-16 <i>sparsity sol</i> : 0.72 % t : 133.337 s <i>Nbit</i> : 108 <i>Norme_{L1}</i> : 0.027 % min : -1.1e-15	<i>Err</i> : 5.39e-16 <i>sparsity sol</i> : 0.153 % t : 0.177 s <i>Nbit</i> : 23 <i>Norme_{L1}</i> : 0 %

BBnnls	<i>Err</i> : 3.35e+00 <i>sparsity sol</i> : 71.84 % t : 1.134 s <i>Nbit</i> : 87 <i>NormeL1</i> : 70.187 %	<i>Err</i> : 2.01e+00 <i>sparsity sol</i> : 11.4733 % t : 3.713 s <i>Nbit</i> : 272 <i>NormeL1</i> : 11.26 %	<i>Err</i> : 1.70e-01 <i>sparsity sol</i> : 4.58 % t : 4.929 s <i>Nbit</i> : 385 <i>NormeL1</i> : 4.173 %	<i>Err</i> : 4.01e-03 <i>sparsity sol</i> : 13.32 % t : 1.644 s <i>Nbit</i> : 121 <i>NormeL1</i> : 13.167 %
	<i>Err</i> : 3.30e+00 <i>sparsity sol</i> : 92.16 % t : 3.232 s <i>Nbit</i> : 44 <i>NormeL1</i> : 90.307 %	<i>Err</i> : 1.68e+00 <i>sparsity sol</i> : 87.39 % t : 16.576 s <i>Nbit</i> : 180 <i>NormeL1</i> : 86.487 %	<i>Err</i> : 8.65e-06 <i>sparsity sol</i> : 5.213 % t : 48.19 s <i>Nbit</i> : 549 <i>NormeL1</i> : 4.713 %	<i>Err</i> : 1.29e-06 <i>sparsity sol</i> : 2.126 % t : 11.533 s <i>Nbit</i> : 177 <i>NormeL1</i> : 1.973 %

TABLE 2.4 – Résultats des algorithmes sur la matrice T-3 (5% *sparsity* de \mathbf{A})

T-4 / Sparsity	2 %	1 %	0.5 %	0.15 %
Lsqnonneg	<i>Err</i> : 1.12e+00 <i>sparsity sol</i> : 0.5 % t : 0.174 s <i>Nbit</i> : 101 <i>NormeL1</i> : 2.47 %	<i>Err</i> : 1.22e+00 <i>sparsity sol</i> : 0.5 % t : 0.170 s <i>Nbit</i> : 100 <i>NormeL1</i> : 1.49 %	<i>Err</i> : 1.37e+00 <i>sparsity sol</i> : 0.495 % t : 0.163 s <i>Nbit</i> : 102 <i>NormeL1</i> : 0.975 %	<i>Err</i> : 1.50e+00 <i>sparsity sol</i> : 0.39 % t : 0.322 s <i>Nbit</i> : 137 <i>NormeL1</i> : 0.530 %
	<i>Err</i> : 1.12e+00 <i>sparsity sol</i> : 0.5 % t : 0.133 s <i>Nbit</i> : 101 <i>NormeL1</i> : 2.470 %	<i>Err</i> : 1.22e+00 <i>sparsity sol</i> : 0.505 % t : 0.121 s <i>Nbit</i> : 101 <i>NormeL1</i> : 1.480 %	<i>Err</i> : 1.31e+00 <i>sparsity sol</i> : 0.515 % t : 46.696 s <i>Nbit</i> : 103 <i>NormeL1</i> : 0.960 % min : -6.1e-02	<i>Err</i> : 1.19e+00 <i>sparsity sol</i> : 0.505 % t : 36.435 s <i>Nbit</i> : 101 <i>NormeL1</i> : 0.45 % min : -2.1e+00
	<i>Err</i> : 7.80e+00 <i>sparsity sol</i> : 97.7 % t : 0.035 s <i>Nbit</i> : 19 <i>NormeL1</i> : 95.745 %	<i>Err</i> : 9.49e+00 <i>sparsity sol</i> : 98 % t : 0.060 s <i>Nbit</i> : 35 <i>NormeL1</i> : 97.04 %	<i>Err</i> : 1.05e+01 <i>sparsity sol</i> : 89.95 % t : 0.118 s <i>Nbit</i> : 67 <i>NormeL1</i> : 89.485 %	<i>Err</i> : 7.24e+00 <i>sparsity sol</i> : 14.41 % t : 0.245 s <i>Nbit</i> : 114 <i>NormeL1</i> : 14.345 %
	<i>Err</i> : 7.53e+00 <i>sparsity sol</i> : 97.5 % t : 0.370 s <i>Nbit</i> : 15 <i>NormeL1</i> : 95.63 %	<i>Err</i> : 9.38e+00 <i>sparsity sol</i> : 98.3 % t : 0.497 s <i>Nbit</i> : 19 <i>NormeL1</i> : 97.33 %	<i>Err</i> : 9.05e+00 <i>sparsity sol</i> : 93.3 % t : 5.177 s <i>Nbit</i> : 93 <i>NormeL1</i> : 92.81 %	<i>Err</i> : 7.36e+00 <i>sparsity sol</i> : 34.62 % t : 16.666 s <i>Nbit</i> : 380 <i>NormeL1</i> : 34.470 %

TABLE 2.5 – Résultats des algorithmes sur la matrice T-4 (5% *sparsity* de \mathbf{A})

2.2.3.1 Comparaison du temps de calcul

la figure (2.2) représente l'évolution du temps de calcul en échelle log pour chaque algorithme en fonction de la *sparsity* pour les quatre matrices :

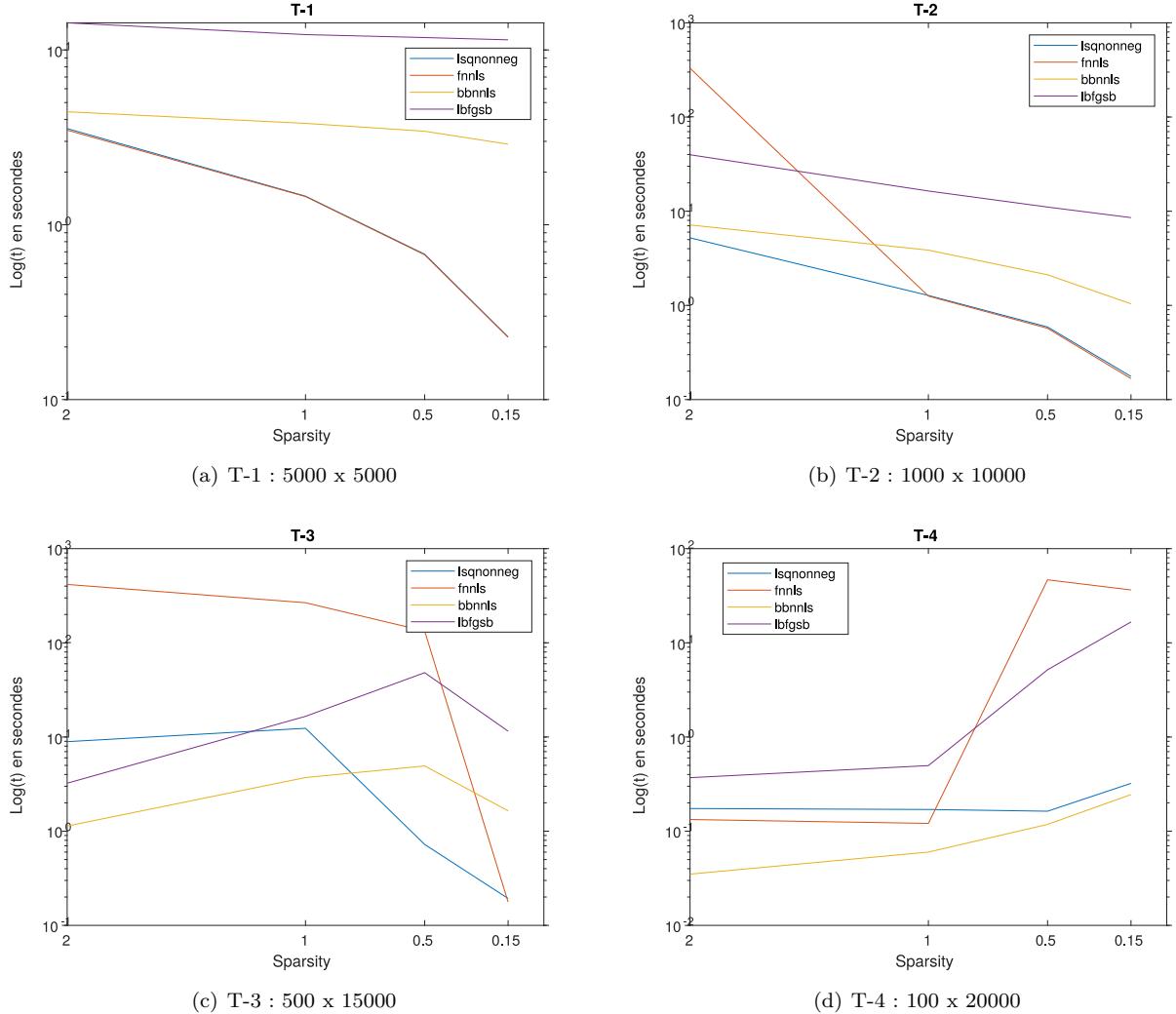


FIGURE 2.2 – Evolution du temps de calcul des algorithmes : *lsqnonneg*, *fnnls*, *bbnlsls*, *lbfgsb* en fonction de la *sparsity* de \mathbf{x} pour chaque matrice (de *sparsity* fixe 5%)

La similarité entre les deux algorithmes *active-set* LSQNONNEG et FNNLS (courbes bleue et rouge) est assez visible sur la matrices T-1 (carrée, $\frac{N}{M} = 1$) quelque soit le nomnbre d'éléments non nuls de \mathbf{x} . Pour les *sparsity* 1%, 0.5% et 0.15% de T-2 ($\frac{N}{M} = 0.1$), celle-ci est toujours présente, mais disparaît complètement lorsque la configuration devient de plus en plus sous-déterminée (T-3 avec $\frac{N}{M} \sim 0.03$ et T-4 avec $\frac{N}{M} = 0.005$). Ce changement de comportement est du à leur nature *active-set* et au calcul de l'équation normale (1.6). En effet ce calcul devient de plus en plus couteux pour FNNLS à mesure que le nombre de colonnes de la matrice s'éloigne du nombre de lignes, étant donné sa double opération matricielle effectuée durant sa recherche de variables actives. Par ailleurs la solution est belle et bien retrouvée (Norme L_1 similaire entre 0% et 1% ce qui est plutot satisfaisant). Néanmoins, sur des configurations simples, l'algorithme *FNNLS* se voit réstreindre son champs d'application, en plus des valeurs négatives obtenues sur T-4, ce qui place LSQNONNEG en tête de liste.

En ce qui concerne les deux algorithmes itératifs BBNLNS et L-BFGS-B, (courbes violette et jaune), ceux-ci ne se rapprochent pas assez de la solution (Norme L_1 trop élevée) pour concurrencer LSQNONNEG, même si pour T-4 le temps de calcul de BBNLNS est légèrement plus faible que LSQNONNEG.

La figure (2.3) représente l'évolution du temps de calcul en échelle log des algorithmes en fonctions des différentes matrices pour chaque *sparsity* :

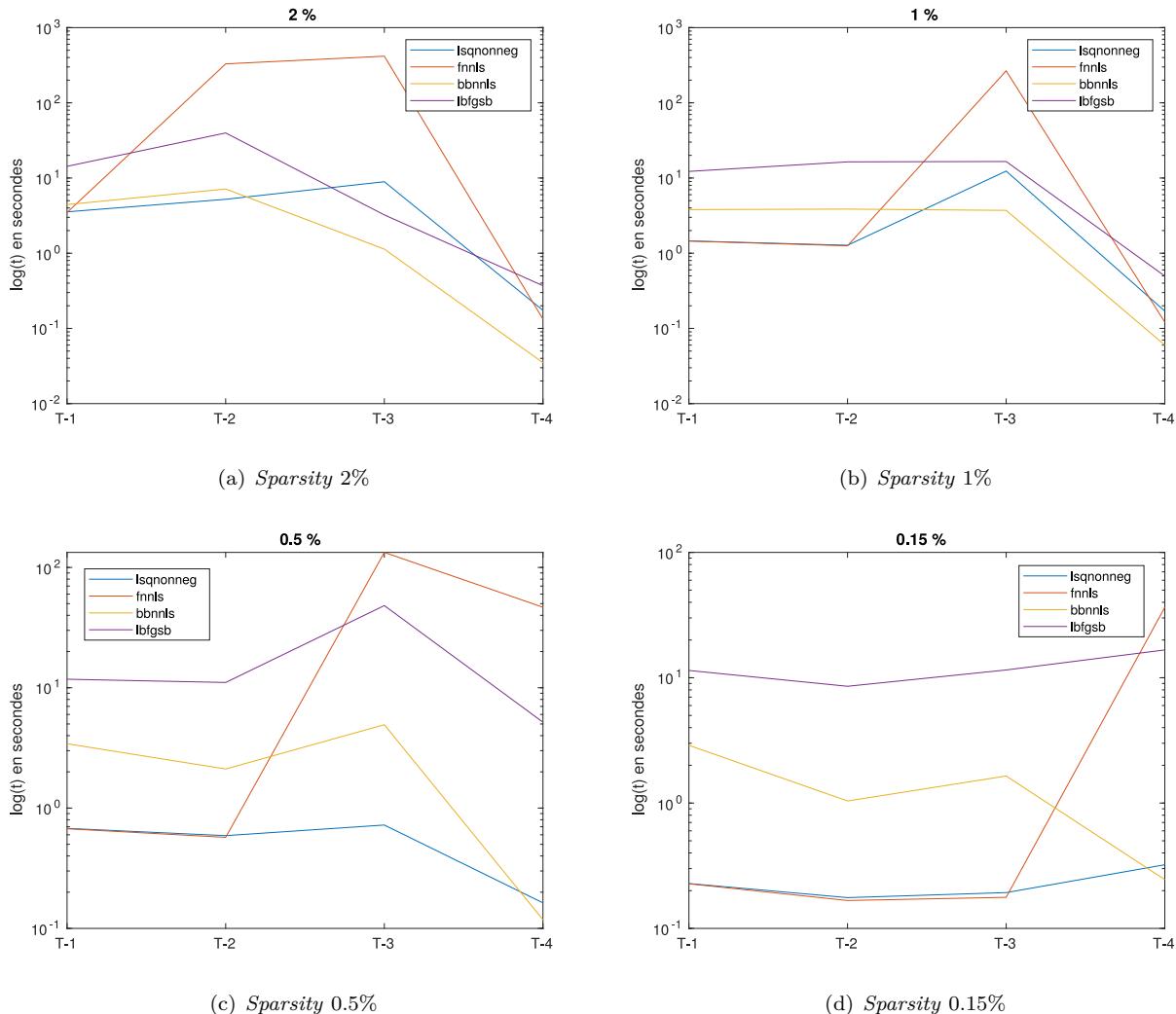


FIGURE 2.3 – Evolution du temps de calcul des algorithmes : *lsqnonneg*, *fnnls*, *bbnlsls*, *lbfgsb* en fonction de T-[1-4] (de *sparsity* fixe 5%) pour chaque *sparsity* de \mathbf{x}

Sur cette figure, on peut d'abord confirmer l'une des observations précédentes : FNNLS n'est pas un algorithme approprié pour des problèmes sous-déterminés simples. Aussi, plus la solution recherchée est *sparse*, plus FNNLS met du temps à la retrouver sur un système sous-déterminé ((d) courbe rouge).

Afin de départager les algorithmes par rapport au temps de calcul, la figure 2.3 (d) est tracée ci-dessous en échelle linéaire. Les résultats ont été normalisés par rapport à LSQNONNEG, ce qui permet de souligner l'efficacité de cet algorithme par rapport aux trois autres pour une recherche de solution \mathbf{x} à 0.15% d'éléments non nuls :

- Dès que l'on passe à une matrice très sous-déterminée, FFNLS ne convient plus : il est ~ 113 fois plus lent que LSQNONNEG ;
- L-BFGS-B est quasiment tout le temps plus lent que LSQNONNEG ;
- BBNLS est plus rapide que sur T-4. Or malgré ce résultat, la norme L_1 de BBNLS sur cette matrice les départage et donne l'avantage à LSQNONNEG.

Une faible *sparsity* joue donc un rôle important sur le temps de calcul de ces algorithmes. En effet, la combinaison faible *sparsity* et matrice très sous-déterminée, tend à multiplier le temps de calcul par ~ 40 pour FNNLS, et par des coefficients plus faibles (3,2) pour les algorithmes itératifs BBNLS et L-BFGS-B.

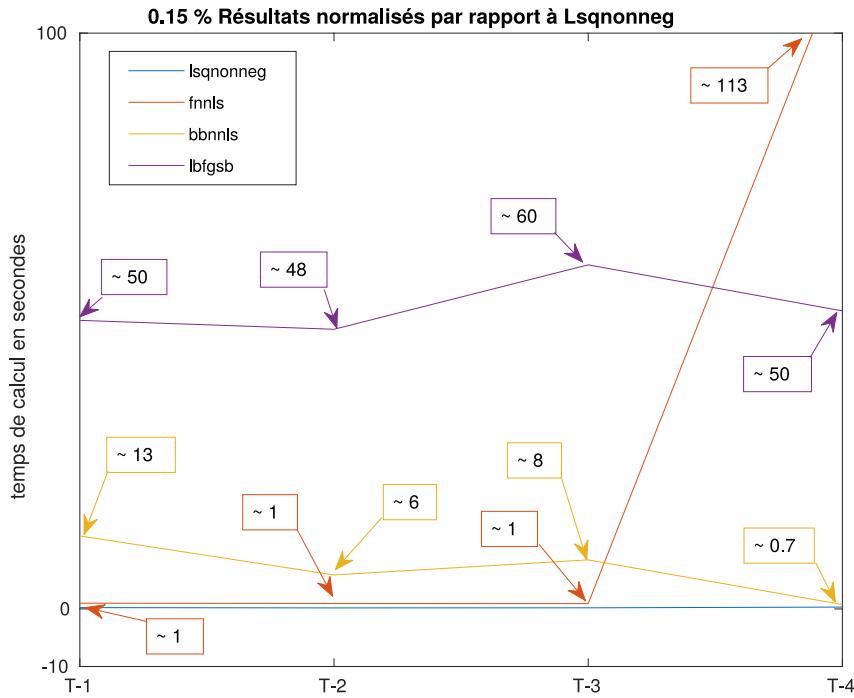


FIGURE 2.4 – Évolution du temps de calcul des algorithmes *fnnls*, *bbnlsls*, *lbfgsb* normalisés par rapport à *lsqnonneg* en fonction de T-[1-4] (*sparsity* fixée à 5%) et pour une *sparsity* de \mathbf{x} de 0.15%

Dans la section suivante, nous étudions les résultats en termes de normes L_1 , des explications seront données par rapports aux valeurs des erreurs relatives L_2 , et ainsi nous pourrons conclure quant à l'algorithme le plus efficace sur des matrices dont les éléments ont été générés aléatoirement.

2.2.3.2 Comparaison des normes L_1 et des erreurs relatives L_2

La norme L_1 nous donne une information sur la position des particules de la solution calculée par rapport à la position exacte, 0% étant une détection parfaite. Un résultat acceptable pour ces premiers tests tiendrait dans l'intervalle [0; 5] %.

Ainsi, pour les algorithmes *active-set LSQNONNEG* et *FNNLS*, quelque soit la *sparsity* recherchée de \mathbf{x} et quelque soit la taille de la matrice, ceux-ci fournissent une solution très proche de la solution exacte en terme de positions. Néanmoins, sur les matrices très sous-déterminées T-3 ou encore T-4, on peut observer un certain rapprochement vers la solution exacte lorsque le nombre d'éléments non nuls (*sparsity*) diminue.

Lsqnonneg / <i>sparsity</i>	2 %	1 %	0.5 %	0.15 %
T-3	4.847 %	3.593 %	0 %	0 %
T-4	2.47 %	1.49 %	0.975 %	0.530 %

TABLE 2.6 – Norme L_1 de l'algorithme *lsqnonneg* pour différentes *sparsity* de \mathbf{x} et pour les deux matrices T-3 (500 x 15000) et T-4 (100 x 20000)

Fnnls / <i>sparsity</i>	2 %	1 %	0.5 %	0.15 %
T-3	4.72 %	3.147 %	0.027 %	0 %
T-4	2.47 %	1.48 %	0.96 %	0.45 %

TABLE 2.7 – Norme L_1 de l'algorithme *fnnls* pour différentes *sparsity* de \mathbf{x} et pour les deux matrices T-3 (500 x 15000) et T-4 (100 x 20000)

Ce résultat est favorable quant à la recherche d'une solution très *sparse*.

Les algorithmes itératifs BBNLNS et L-BFGS-B voient aussi leur norme L_1 diminuer avec la *sparsity* de la solution recherchée, mais aucun d'eux ne fournit de résultats acceptables et ce quelque soit la taille de la matrice.

En ce qui concerne l'erreur relative L_2 , celle-ci n'est pas un critère pertinent pour ces premiers tests à génération aléatoire. En effet, son calcul, relié à un calcul d'une énergie, donne une information sur l'amplitude de l'intensité lumineuse. Les données étant générées aléatoirement dans cette partie, on ne peut conclure quant à la pertinence de cette erreur.

Pour conclure, ces tests sur des matrices dont la structure des éléments est générée aléatoirement ont montré que les algorithmes itératifs BBNLNS et L-BFGS-B ne sont pas à retenir par rapport à la solution fournie et aux temps de calculs, et ce quelque soit la dimension de la matrice générée. Quant aux algorithmes *active-set*, FNNLS a montré son inefficacité face à un problème sous-déterminé et *sparse*, ce qui confirme, pour l'instant, la supériorité de LSQNONNEG, dont les résultats sont satisfaisants.

Cependant ces matrices sont générées aléatoirement, or le problème physique de détection des particules repose sur des matrices bandes. Les sections suivantes visent donc à étudier ces mêmes algorithmes sur des matrices plus proches du problème physique, tout en utilisant les mêmes critères d'analyse afin d'en tirer les différences ou les similitudes entre ces deux types de matrices, et de compléter les résultats précédents.

2.3 Matrice A bande

Dans cette section nous générerons des matrices bandes et très sous-déterminées proches du problème physique. Nous présenterons des résultats dans deux parties distinctes. La première avec une dimension et *sparsity* de **A** fixée, et dont nous varierons celle du vecteur **x** et où l'on testera essentiellement LSQNONNEG, FNNLS, BBNLNS et L-BFGS-B afin de compléter les résultats précédents.

Dans la deuxième partie nous procéderons à la même méthode qu'en 2.2.2 (matrice carré -> sous-déterminée) mais cette fois-ci les deux *sparsity* de **A** et de **x** fixées, dans cette partie on testera essentiellement l'évolution de la solution obtenue par les algorithmes ART très utilisés en tomographie.

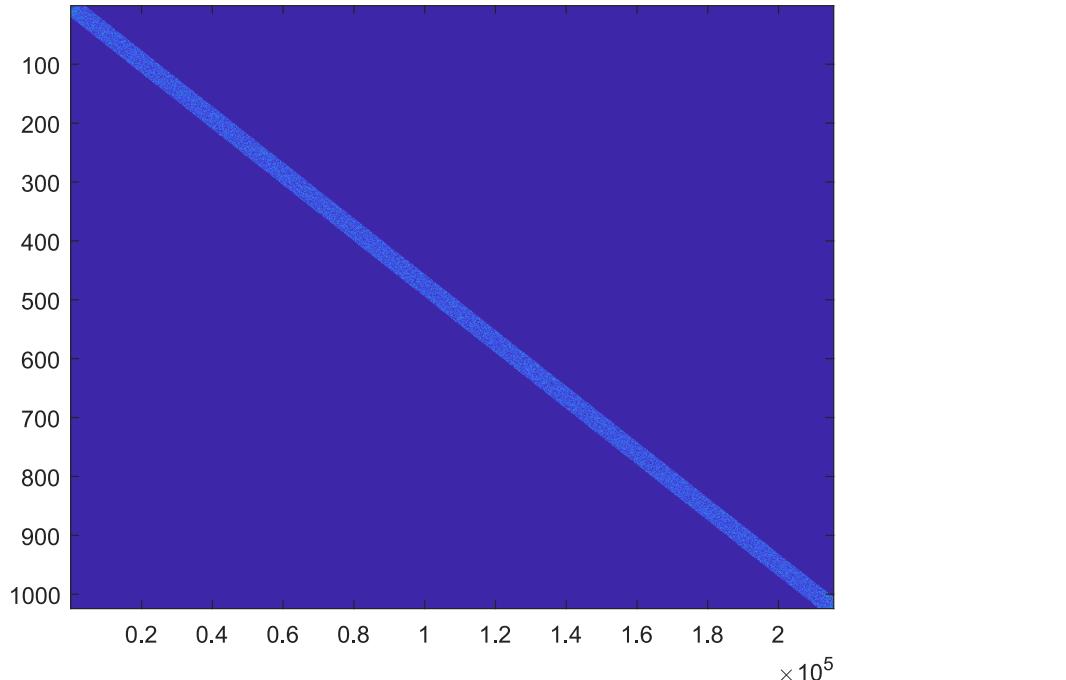


FIGURE 2.5 – Exemple d'une matrice bande sous-déterminée 1024×215600 utilisée dans les tests

2.3.1 Matrices bandes à dimension et *sparsity* de A fixée : LSQNONNEG, FNNLS, BBNLNS et L-BFGS-B

La matrice utilisée est de taille 600×256000 , dont les éléments sont générés aléatoirement en imposant une certaine diagonalité de la structure (voir la figure (2.5)), avec une *sparsity* de $\sim 6\%$. Similaire au problème physique, par le rapport $\frac{N}{M}$ et par sa *sparsity*, l'utilisation de cette matrice a pour but de pousser les algorithmes à des conditions extrêmes et ainsi conclure définitivement quant à leur efficacité sur un problème tel que le nôtre.

Les résultats ci-dessous sont présentés sous la même forme que le tableau (2.1).

600 x 256000	2 %	1 %	0.5 %	0.148 %
Lsqnonneg	<i>Err</i> : 1.04e+00 <i>sparsity sol</i> : 0.234 % t : 20.078 s <i>Nbit</i> : 599 <i>Norme_{L1}</i> : 2.225 %	<i>Err</i> : 1.08e+00 <i>sparsity sol</i> : 0.233 % t : 21.982 s <i>Nbit</i> : 599 <i>Norme_{L1}</i> : 1.231 %	<i>Err</i> : 1.16e+00 <i>sparsity sol</i> : 0.234 % t : 20.389 s <i>Nbit</i> : 598 <i>Norme_{L1}</i> : 2.273 %	<i>Err</i> : 1.46e+00 <i>sparsity sol</i> : 0.232 % t : 20.197 s <i>Nbit</i> : 607 <i>Norme_{L1}</i> : 0.377 %
Fnnls	<i>Err</i> : 1.04e+00 <i>sparsity sol</i> : 0.238 % t : 20.093 s <i>Nbit</i> : 599 <i>Norme_{L1}</i> : 2.22 %	<i>Err</i> : 1.08e+00 <i>sparsity sol</i> : 0.233 % t : 17.894 s <i>Nbit</i> : 597 <i>Norme_{L1}</i> : 1.232 %	<i>Err</i> : 1.17e+00 <i>sparsity sol</i> : 0.236 % t : 18.128 s <i>Nbit</i> : 598 <i>Norme_{L1}</i> : 0.730 %	Temps de calcul > 2 heures
BBnls	<i>Err</i> : 8.10e+00 <i>sparsity sol</i> : 99.9 % t : 6.930 s <i>Nbit</i> : 140 <i>Norme_{L1}</i> : 97.975 %	<i>Err</i> : 1.08e+01 <i>sparsity sol</i> : 99.5 % t : 7.520 s <i>Nbit</i> : 153 <i>Norme_{L1}</i> : 98.514 %	<i>Err</i> : 1.35e+01 <i>sparsity sol</i> : 90.3 % t : 7.102 s <i>Nbit</i> : 139 <i>Norme_{L1}</i> : 89.878 %	<i>Err</i> : 1.40e+01 <i>sparsity sol</i> : 52 % t : 19.085 s <i>Nbit</i> : 361 <i>Norme_{L1}</i> : 51.927 %
L-bfgs-B	<i>Err</i> : 8.10e+00 <i>sparsity sol</i> : 99.9 % t : 73.082 s <i>Nbit</i> : 98 <i>Norme_{L1}</i> : 97.998 %	<i>Err</i> : 1.07e+01 <i>sparsity sol</i> : 99.9 % t : 70.180 s <i>Nbit</i> : 95 <i>Norme_{L1}</i> : 98.963 %	<i>Err</i> : 1.32e+01 <i>sparsity sol</i> : 96.6 % t : 100.696 s <i>Nbit</i> : 1114 <i>Norme_{L1}</i> : 96.105 %	<i>Err</i> : 1.41e+01 <i>sparsity sol</i> : 99.7 % t : 118.849 s <i>Nbit</i> : 142 <i>Norme_{L1}</i> : 99.576 %

TABLE 2.8 – Résultats des algorithmes sur une matrice bande 600×256000 ($\sim 6\% \text{ sparsity de } \mathbf{A}$)

FNNLS est toujours très proche de LSQNONNEG. Cependant, pour un système très *sparse*, celui-ci a été arrêté car son temps de calcul dépassait les 2 heures (sur tous les calculs, car plusieurs tests ont été effectués), ce qui le place hors compétition pour la suite.

De même L-BFGS-B ne fournit finalement que des solutions dont la norme $L_1 \geq 96\%$ sur les 4 *sparsity* testées.

BBNLS a le temps de calcul le plus bas, mais encore une fois la norme L_1 n'est pas satisfaisante, malgré le rapprochement vers la solution pour les systèmes très *sparse*.

LSQNONNEG, quant à lui a un temps de calcul supérieur aux résultats des tests sur des matrices générées aléatoirement, et ce à cause de la taille des matrices de ces derniers tests. Néanmoins, la solution fournie est très proche de la solution exacte (0.377%) pour un système de *sparsity* identique au problème physique. Ce qui fait de lui l'algorithme de référence, comme suggéré en introduction.

2.3.2 Matrices bandes à *sparsity* de A et x fixée : ART, SMART, LSQNONNEG

Après avoir donc mis de côté FNNLS, BBNLNS et L-BFGS-B, il est temps de passer aux tests de comparaison des algorithmes Algébraic Reconstruction Technique (ART Kaczmarz, Random, Symetric et SMART) à LSQNONNEG. Ces algorithmes sont très utilisés en tomographie, il est donc judicieux de les tester étant donné les nombreuses similarités du problème physique de détection de particules au domaine de la tomographie.

Deux matrices bandes seront utilisées : 1024×10245 "faiblement" sous-déterminée, et 1024×215600 fortement sous-déterminée. La *sparsity* des deux matrices est fixée à $\sim 4\%$, et 0.14% pour \mathbf{x} . A noter que la deuxième configuration est identique à une matrice issue d'une image de particules 32×32 pixels de densité de particules notée PPP5 (voir Chapitre 3).

L'analyse des algorithmes ART est différente des sections précédentes. En effet, le résidus ($r = b - Ax$) est calculé et analysé à chaque itération pour vérifier le cheminement vers la solution. Une bonne solution equivaut à un résidu $r \rightarrow 0$ à la dernière itération. Aussi, la solution (la vectorisation des positions des particules sur une image) sera présentée pour chaque algorithme, et comparée à l'algorithme de référence LSQNONNEG. On rappelle que le but est de rechercher une solution *sparse* et à amplitude piquée.

2.3.2.1 LSQNONNEG

La figure (2.6) présente les résidus de LSQNONNEG à la dernière itération.

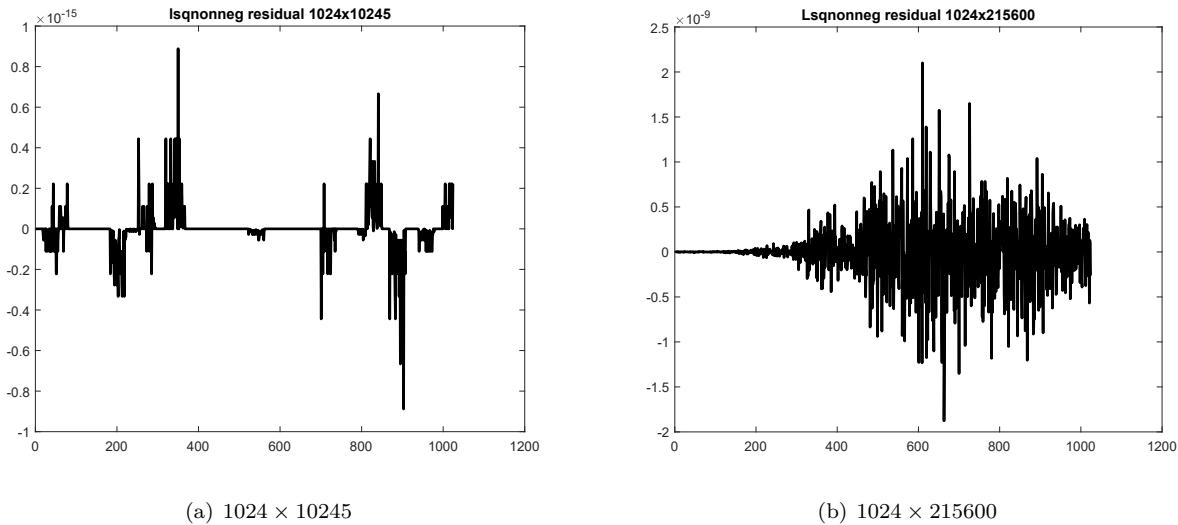


FIGURE 2.6 – Résidus LSQNONNEG sur une matrice bande A de *sparsity* $\sim 4\%$

Sur les deux matrices, le résidu est d'un ordre de grandeur (10^{-15} et 10^{-9}), à noter que cette différence est due à la taille de la matrice) très faible, ce qui maintient LSQNONNEG en tête de liste. Ci-dessous, les solutions finales sur les deux configurations pour cet algorithme.

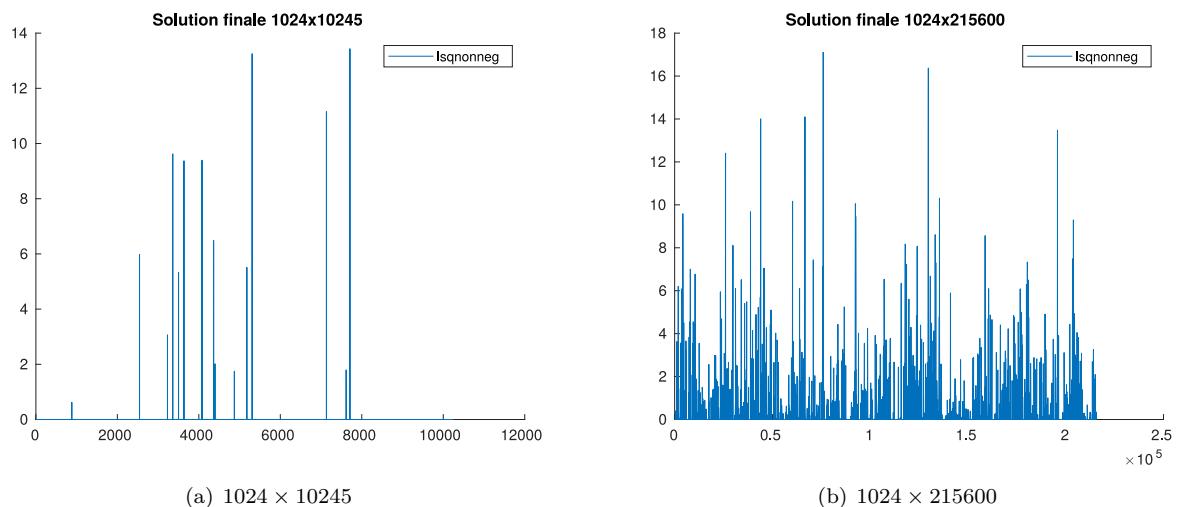


FIGURE 2.7 – Solution finale LSQNONNEG sur une matrice bande A de *sparsity* $\sim 4\%$

Dans les deux cas la *sparsity* de la solution est parfaitement respectée (0.14%). Les amplitudes sont bien piquées, ce qui est recherché. Sur la matrice 1024×10245 le temps de calcul est faible : $0.2109s$, et sur 1024×215600 celui-ci monte à $38.88s$ mais reste raisonnable.

LSQNONNEG sera donc l'algorithme de référence, et on testera les algorithmes ART dans les sections suivantes. Aussi, de multiples tests ont été lancés, les résultats sont identiques.

Comme expliqué au chapitre 1, ART possède 3 méthodes de balayages (qu'on nommera 'ART k', 'ART rand' et 'ART sym'), chacune sera considérée et testée comme un algorithme à part entière.

2.3.2.2 ART k

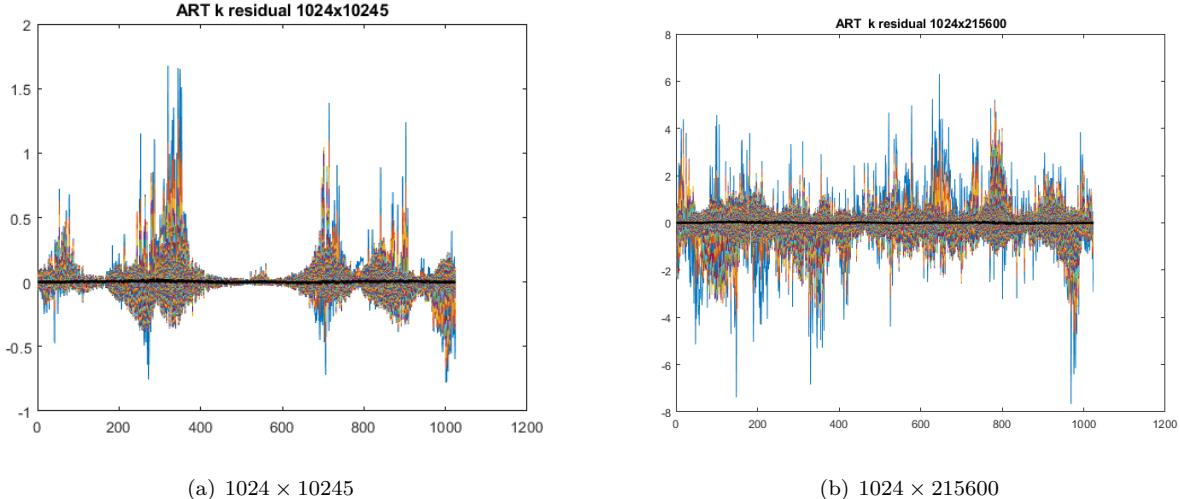


FIGURE 2.8 – Résidus de toutes les itérations (200) ART Kaczmarz - $\text{sparsity} \sim 4\%$

La figure (2.9) détaille le résidu après 200 itérations avec ART k.

La bande noire qu'on peut observer au niveau de l'axe horizontal est le résidu de la dernière itération. Ce résidu est présenté ci-dessous :

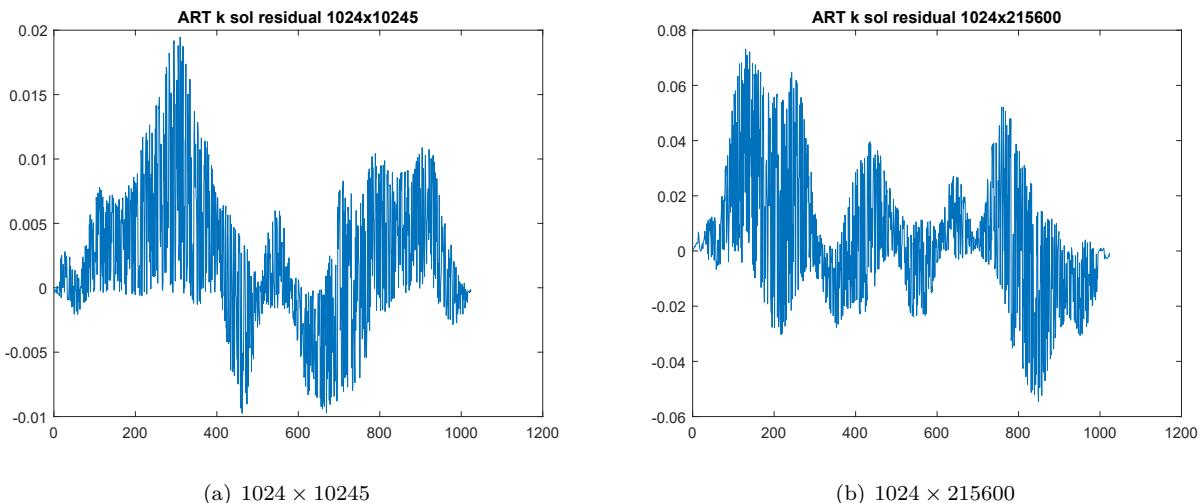


FIGURE 2.9 – Résidus de la solution à l'itération finale ART Kaczmarz - $\text{sparsity} \sim 4\%$

L'ordre de grandeur en amplitude de ces deux courbes est de 10^{-1} . Cette méthode de balayage ne retrouve donc pas la bonne solution recherchée à 10^{-1} près.

On peut émettre la même observation, cette fois-ci qualitative, sur le tracé des positions des particules issues de ART k, comparé à LSQNONNEG dans la (figure 2.10).

ART k produit ce qui s'apparente à du bruit. Sur les deux détections, le bruit est concentré autour des positions retrouvées par LSQNONNEG (mieux visible sur (2.10 - a)). Mais l'algorithme ne fournit pas de solutions assez piquées. Cela est confirmé par le résidu de la figure (2.9), trop élevé.

Aussi, une analogie avec une certaine conservation de l'énergie peut être liée aux amplitudes des intensités lumineuses observées ici : LSQNONNEG répartit l'énergie reçue comme désiré mieux que ART k.

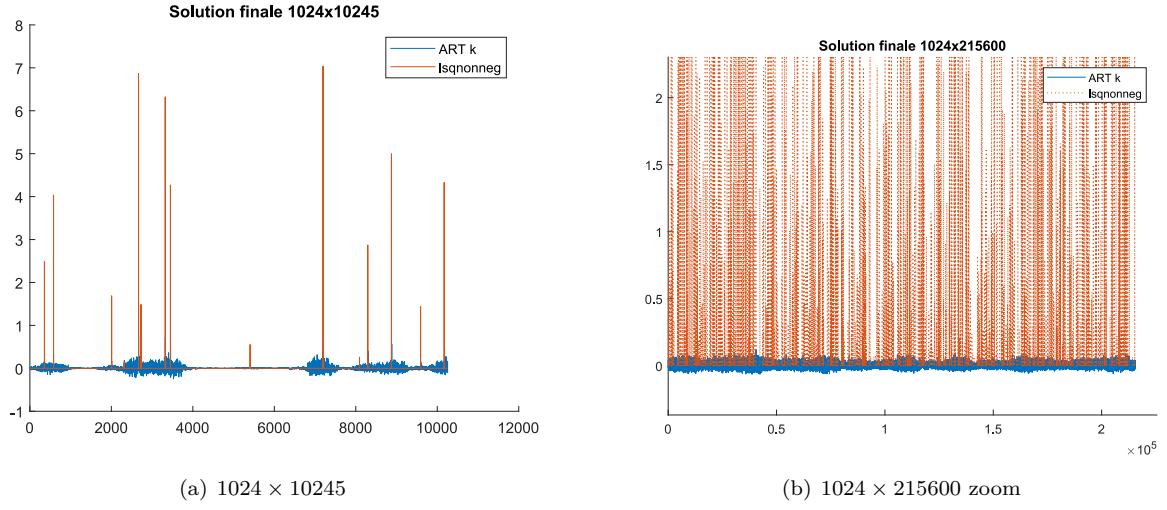


FIGURE 2.10 – Solution finale ART k et LSQNONNEG sur une matrice bande A de $sparsity \sim 4\%$

2.3.2.3 ART rand

De la même façon que précédemment, les résidus de toutes les itérations (200) sont tracés et la dernière itération (bande noire) est tracée sur la figure suivante (2.12) pour les deux matrices :

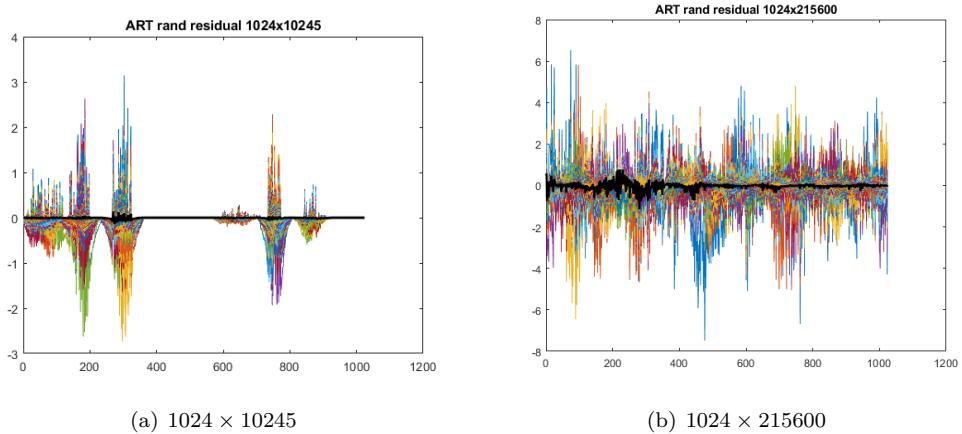


FIGURE 2.11 – Résidus de toutes les itérations (200) ART Random - $sparsity \sim 4\%$

L'ordre de grandeur de ces résidus n'est pas satisfaisant. Cela peut s'observer sur la solution finale (figure (2.13)). En effet, comparée à LSQNONNEG, ART rand ne fournit pas d'amplitudes assez piquées.

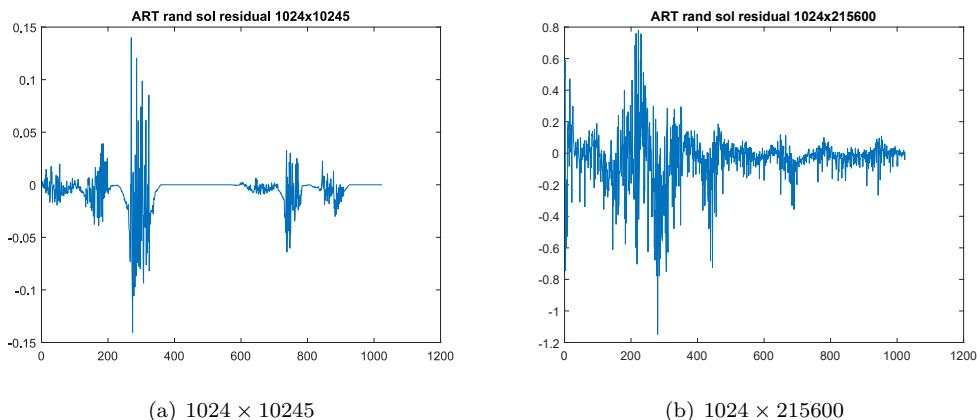


FIGURE 2.12 – Résidus de la solution à l'itération finale ART Random - $sparsity \sim 4\%$

Cependant, on peut remarquer que ces amplitudes sont plus piquées qu'avec ART k. La méthode de balayage ART rand est donc légèrement plus efficace pour ce problème.

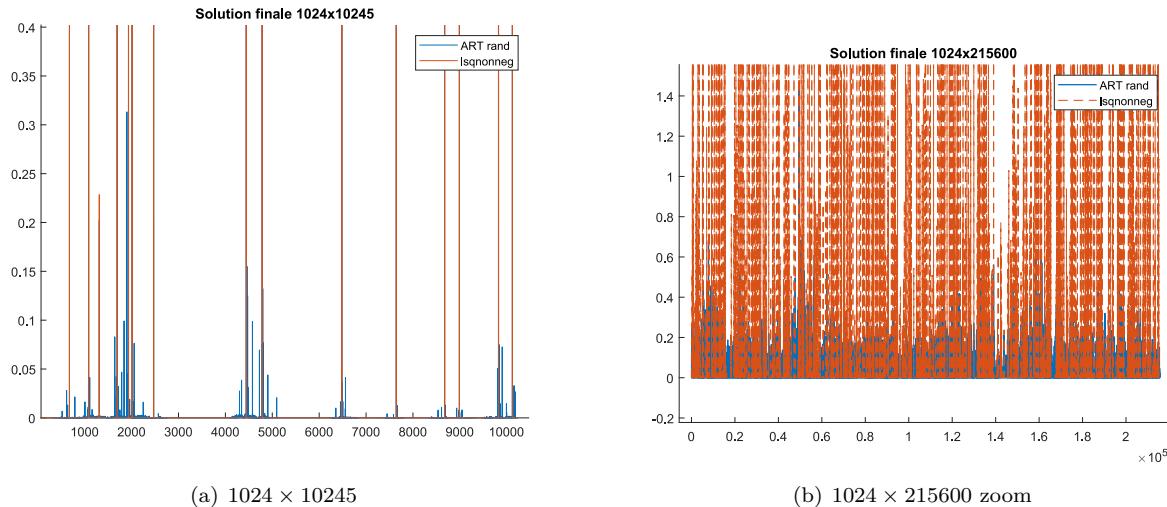


FIGURE 2.13 – Solution finale ART Rand et LSQNONNEG sur une matrice bande A de *sparsity* $\sim 4\%$

2.3.2.4 ART sym

De manière similaire, les résidus de toutes les itérations et de la dernière itération sont tracés sur les deux figures suivantes :

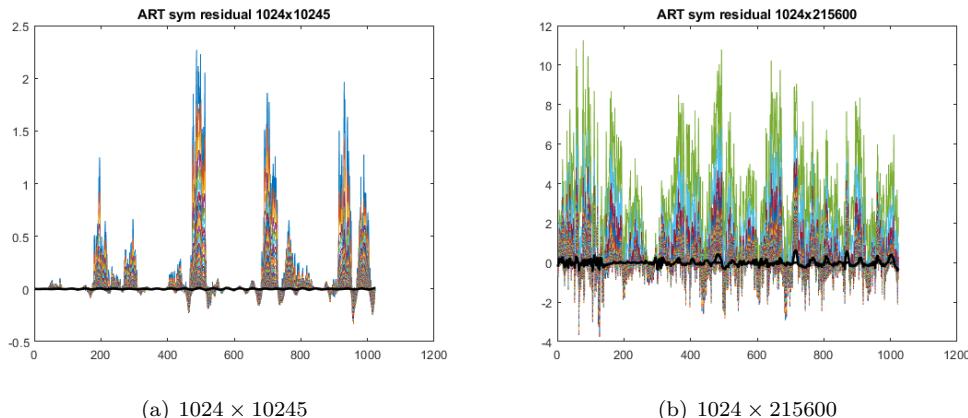


FIGURE 2.14 – Résidus de toutes les itérations (200) ART Symetric - *sparsity* $\sim 4\%$

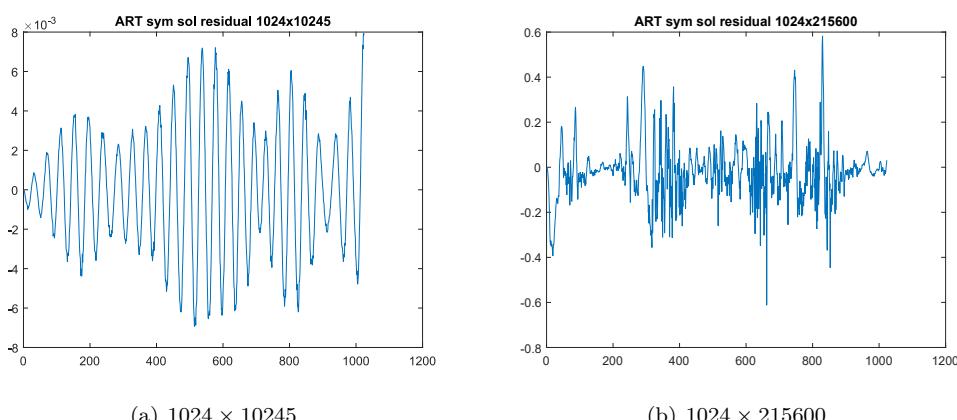


FIGURE 2.15 – Résidus de la solution à l’itération finale ART Symetric - *sparsity* $\sim 4\%$

Même observation qu'avec ART rand et ART k, les solutions se concentrent autour des bonnes positions, mais ne sont pas assez piquées. A remarquer que ART sym fournit les solutions les plus piquées par rapport aux deux autres méthodes de balayages.

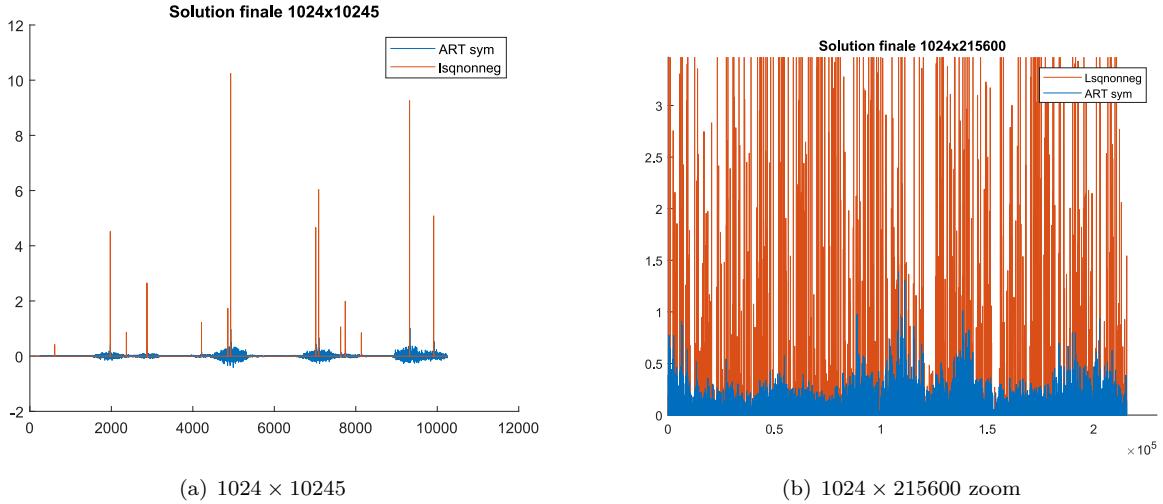


FIGURE 2.16 – Solution finale ART Sym et LSQNONNEG sur une matrice bande A de $sparsity \sim 4\%$

Ainsi, pour les trois méthodes de balayages ART, les résidus sont plus ou moins similaires (10^{-1} , à un facteur 0.1 près) mais toujours trop élevés pour être acceptables. Cela se traduit bien sur les tracés des solutions pas assez piquées comparées à celle fournies par LSQNONNEG.

Cette recherche de solutions piquées est très importante pour la suite, car après le processus de seuillage qui coupera les amplitudes les plus basses afin de ne laisser que les plus hautes synonymes de bonne intensité lumineuse (i.e position d'une particule), les solutions telles que celles fournies par les trois ART seront inexistantes au bout de quelques images (soit quelques instants). Ceci compromet bien sur le but global de reconstruction du champ de vitesse et d'étude du champ turbulent.

2.3.2.5 SMART

SMART, version multiplicative et simultanée de ART, possède un comportement différent de ce dernier.

Comme précédemment, les résidus de toutes les itérations et de la dernière sont tracés pour les deux matrices :

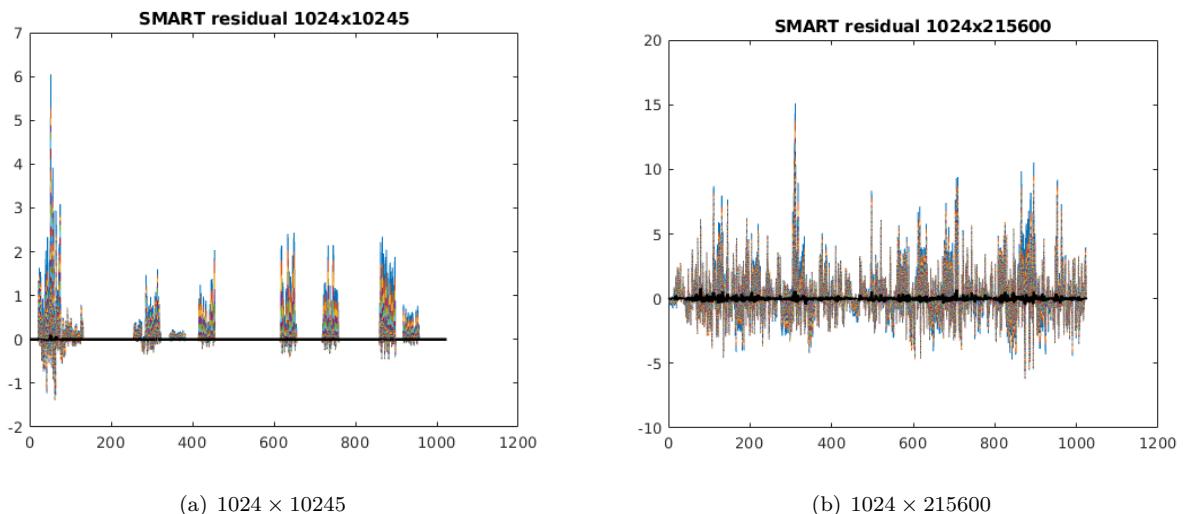


FIGURE 2.17 – Résidus de toutes les itérations (200) SMART - $sparsity \sim 4\%$

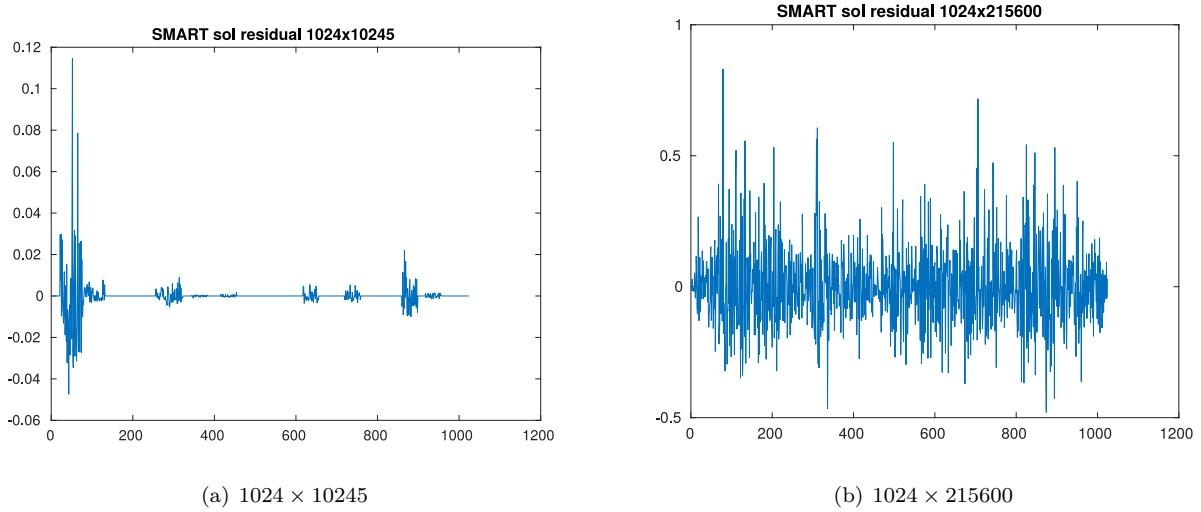


FIGURE 2.18 – Résidus de la solution à l’itération finale SMART - $sparsity \sim 4\%$

La forme du résidu pour la matrice 1024×10245 est étrangement similaire au résidu de LSQNONNEG sur la même matrice. Son amplitude est $\sim 10^{-13}$ plus élevée. Aussi, à noter que pour cette matrice, les amplitudes issues de SMART et de LSQNONNEG sont quasi identiques.

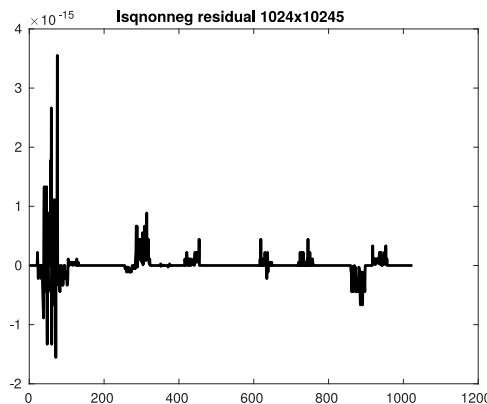


FIGURE 2.19 – Résidus de la solution à l’itération finale LSQNONNEG - 1024×10245 $sparsity \sim 4\%$

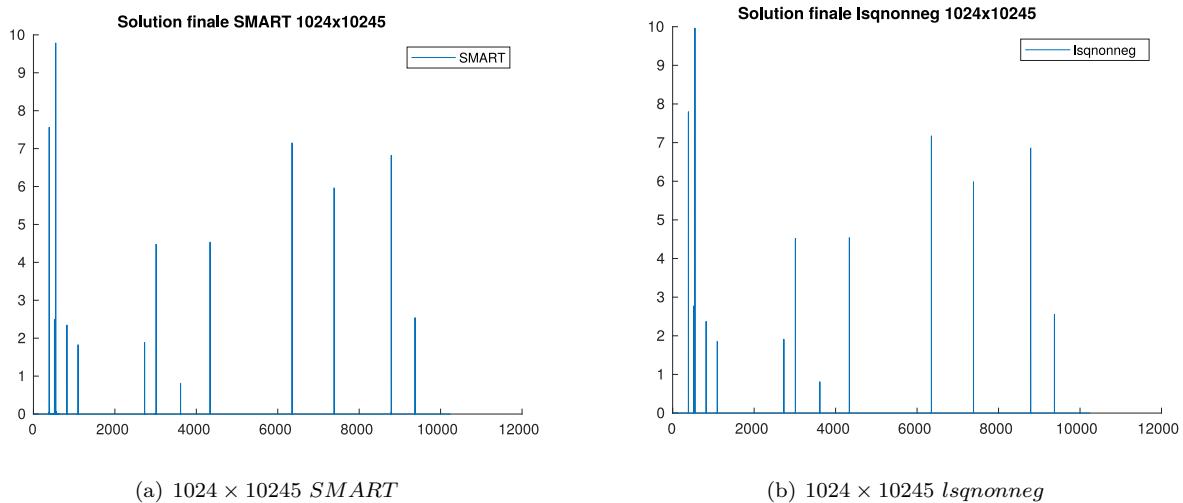


FIGURE 2.20 – Solution finale SMART et LSQNONNEG sur une matrice bande A 1024×10245 de $sparsity \sim 4\%$

En effet, comme on peut le voir sur la figure (2.20), les amplitudes sont identiques. Cependant, en calculant la norme L_1 , comme décrite en section (2.1.2), on se retrouve avec $\sim 54\%$ de différences. Cette différence est due aux valeurs d'amplitudes les plus faibles. Celles qui sont à 0 pour LSQNONNEG, peuvent aller de $\sim 10^{-10}$ à $\sim 10^{-2}$ pour SMART. On peut l'observer en superposant les deux solutions et en zoomant sur un petit intervalle.

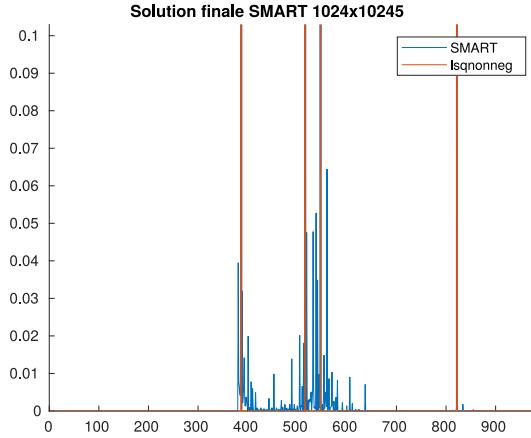


FIGURE 2.21 – Solution finale SMART et LSQNONNEG sur une matrice bande A 1024×10245 de *sparsity* $\sim 4\%$ - Zoom sur les différences

Le calcul sur la deuxième matrice 1024×215600 fournit des résultats dont on peut observer les mêmes comportements. Cette fois-ci, la norme est de 99%. Etant donné le nombre d'éléments 215600, et comme SMART semble fournir, en plus des amplitudes piquées, une sorte de bruit à de basses amplitudes, ce résultat est normal car quasiment toutes les positions sont remplies (voir figure 2.23) qui est un zoom sur la superposition des deux figures (2.22)

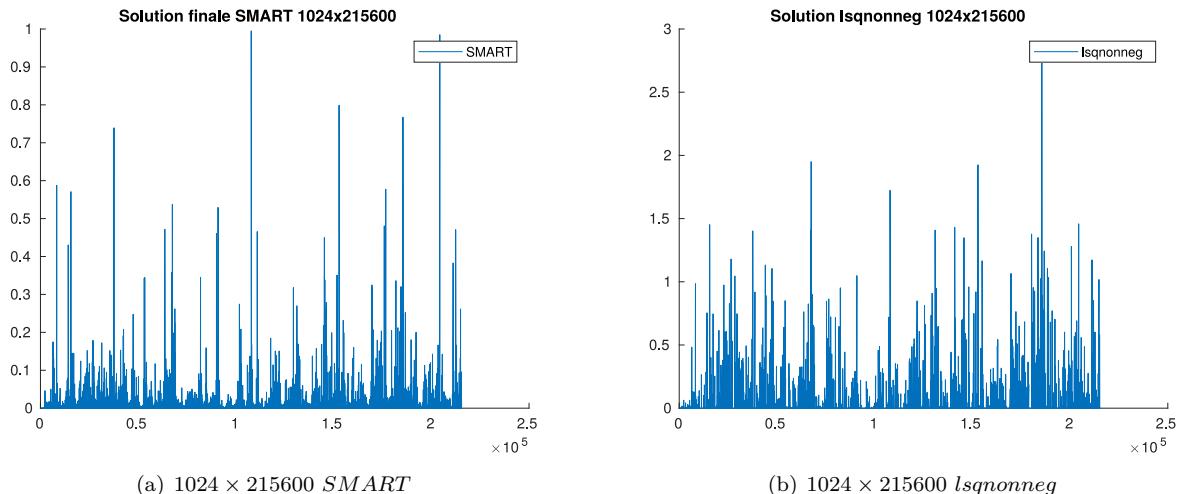


FIGURE 2.22 – Solution finale SMART et LSQNONNEG sur une matrice bande A 1024×215600 de *sparsity* $\sim 4\%$

Ci-dessus, on peut observer des amplitudes assez piquées pour SMART, environ le 1/3 de LSQNONNEG. Ce résultat est le meilleur (hormis LSQNONNEG) par rapport aux ART. Malgré, la norme L_1 très élevée (à cause du bruit à basse amplitude), SMART sera retenu pour les comparaisons avec des images réelles, puisque le processus de seuillage supprimera ce bruit et ne gardera que les amplitudes piquées.

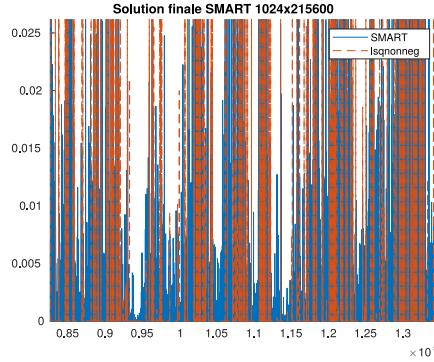


FIGURE 2.23 – Solution finale SMART et LSQNONNEG sur une matrice bande A 1024×215600 de *sparsity* $\sim 4\%$ - Zoom sur une partie des différences

2.3.2.6 Normes L_1 et temps de calculs de LSQNONNEG, ART k,rand,sym, et SMART

Les normes L_1 des solutions de chaque algorithme ont été calculées et reportées sur le tableau ci-dessous, ainsi que le temps de calcul.

Matrices / Algorithmes	Lsqnonneg	SMART	ART k	ART sym
1024×10245	t : 0.21 s <i>Norme</i> L_1 : 0 %	t : 2.99 s <i>Norme</i> L_1 : 54.91 %	t : 18.79 s <i>Norme</i> L_1 : 51.62 %	t : 28.56 s <i>Norme</i> L_1 : 52.10 %
1024×215600	t : 38.88 s <i>Norme</i> L_1 : 0.59 %	t : 73.53 s <i>Norme</i> L_1 : 99.68 %	t : 160.16 s <i>Norme</i> L_1 : 66.78 %	t : 385.06 s <i>Norme</i> L_1 : 66.19 %

TABLE 2.9 – Norme L_1 et temps de calcul des algorithmes LSQNONNEG, SMART, ART k, et sym pour les matrices bandes 1024×10245 et 1024×215600 de *sparsity* 4%

Matrices / Algorithmes	ART rand
1024×10245	t : 16.66 s <i>Norme</i> L_1 : 52.26 %
1024×215600	t : 296.91 s <i>Norme</i> L_1 : 66.27 %

TABLE 2.10 – Norme L_1 et temps de calcul de l'algorithme LSQNONNEG, SMART, ART rand pour les matrices bandes 1024×10245 et 1024×215600 de *sparsity* 4%

Pour la matrice proche de notre cas physique (1024×215600 très sous-déterminée et très *sparse*) ART k, rand et sym ne sont pas du tout adaptés. En effet leurs temps de calcul est trop élevé, et l'étude faite sur ces matrices bandes a montré leur inefficacité à produire des solutions piquées.

SMART, malgré sa norme L_1 proche du 100%, a le deuxième meilleur temps de calcul, et fournit des solutions assez piquée. Néanmoins, LSQNONNEG reste en tête de liste avec une norme quasi parfaite et le meilleur temps de calcul.

Ainsi, en procédant à une étude, d'abord sur des cas simples de matrices carrées et sous-déterminée en faisant varier les *sparsity*, ensuite sur des cas plus proches du problème physique en fixant cette fois-ci la *sparsity* de **A** et **x**, nous avons pu mettre en évidence l'efficacité de l'algorithme LSQNONNEG comparé aux autres algorithmes de résolution du système linéaire. Ceci dit, ces tests ont été faits sur des matrices synthétiques, en essayant de reproduire au mieux les conditions réelles. Il est donc logique de passer à des tests sur des matrices réelles issues d'images de particules. C'est l'objet du chapitre suivant. Nous commencerons par fournir certaines explications quant aux méthodes utilisées, puis nous testerons les algorithmes précédent. Dans un premier temps, les tests seront faits sur une certaine densité de particules (PPP5) puis sur une plus grande densité de particules (PPP10). Tous les algorithmes seront utilisés dans cette partie. Une seconde partie qui finalisera l'étude, consistera à reconstruire le champ de vitesse grâce à la détection de particules sur deux images successives, où seront utilisés et comparés LSQNONNEG et SMART.

Chapitre 3

Validation à partir de matrices réalistes issues d'images de particules

Ce chapitre a pour but de vérifier et de valider les résultats précédents sur des cas réels d'images de particules issues de [14]. Cette dernière étape est cruciale puisqu'on rappelle que les cas précédents étaient synthétiques (matrice \mathbf{A} générée aléatoire à structure intérieure aléatoire ou fixée en bande diagonale), or dans le cas réel cette matrice est fixée pour chaque image de particules, chaque taille Pixel \times Pixel, et chaque densité de particules.

Cette matrice est aussi structurée en bande latérale, d'une *sparsity* plus ou moins égale à celle du Chapitre précédent ($\sim 4\%$). Cependant cette structure en bande est différente. En effet elle présente des irrégularités formant un genre d'ondulation d'éléments nuls à l'intérieur de la bande diagonale.

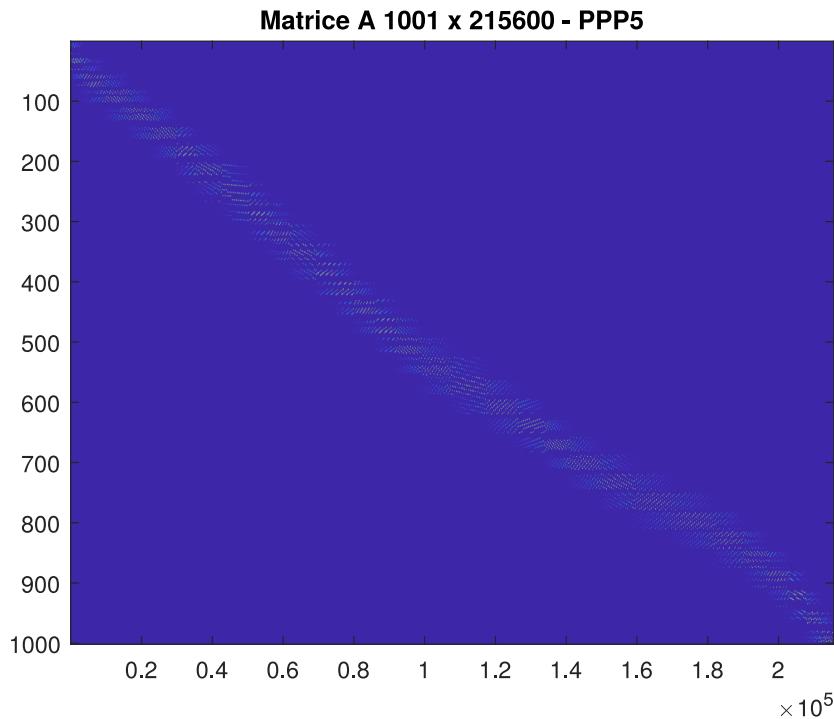


FIGURE 3.1 – Matrice \mathbf{A} 1001×215600 pour une image 32×32 pixels - PPP5

Ce détail n'aura a priori pas d'impact sur le fonctionnement des algorithmes, puisque la structure globale est proche de celle utilisée au Chapitre 2.

Dans un premier temps, l'idée est de tester tous les algorithmes utilisés depuis le début de l'étude sur deux images de 32×32 pixels [14]. L'une ayant une densité de particules PPP (Particules Par Pixel) de 0.05 et la deuxième 0.1 (on écrira respectivement "PPP5" et "PPP10").

La deuxième partie, après avoir fait le choix du meilleur algorithme et du second, consistera à procéder au tracking des particules après détection sur une paire d'images afin de tracer le champ de vitesse de deux vortex, ce qui servira d'exemple d'utilisation de ces algorithmes.

3.1 Tests sur des images de particules 32×32 pixels

Ci-dessous les deux images utilisées, de densités de particules différentes. La position exacte des particules est marquée d'une croix rouge.

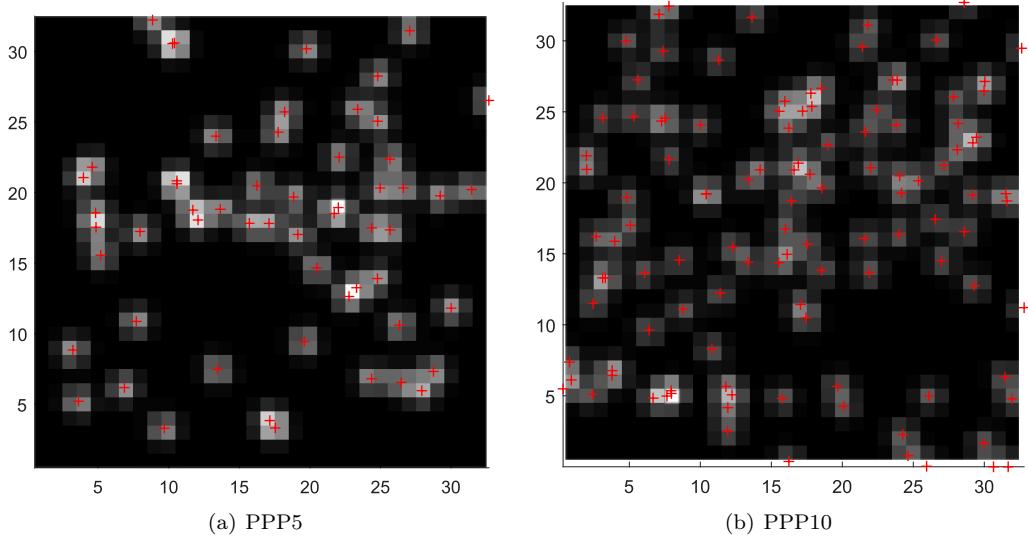


FIGURE 3.2 – Images de particules utilisées avec leurs positions exactes ("+" rouge)

On commencera par comparer en section (3.1.1) le vecteur position issu des algorithmes au vecteur position exacte. La détection des positions sur l'image en elle-même sera présentée en section (3.1.2). Les deux parties sont directement liées puisque le vecteur position est construit à partir des coordonnées des positions sur les images.

Le seuillage des images est obtenue grâce à une fonction Matlab qui analyse les différents niveaux de gris de tous les pixels et estime le seuil minimum qui nous donne le pixel susceptible de contenir au moins une particule

3.1.1 Vecteur position x issu des algorithmes comparé au vecteur position exacte - Détection des positions des particules

Le vecteur x des positions exactes est généré à partir des images. L'amplitude est arbitraire à cause du processus de vectorisation des coordonnées des positions qui ne tient pas compte de l'intensité lumineuse puisque les positions sont déjà marquées et identifiées. Cette amplitude est donc fixée à 150 (après tracé du vecteur position issu de l'algorithme de référence LSQNONNEG) afin de mieux voir l'efficacité des algorithmes en termes d'amplitudes piquées.

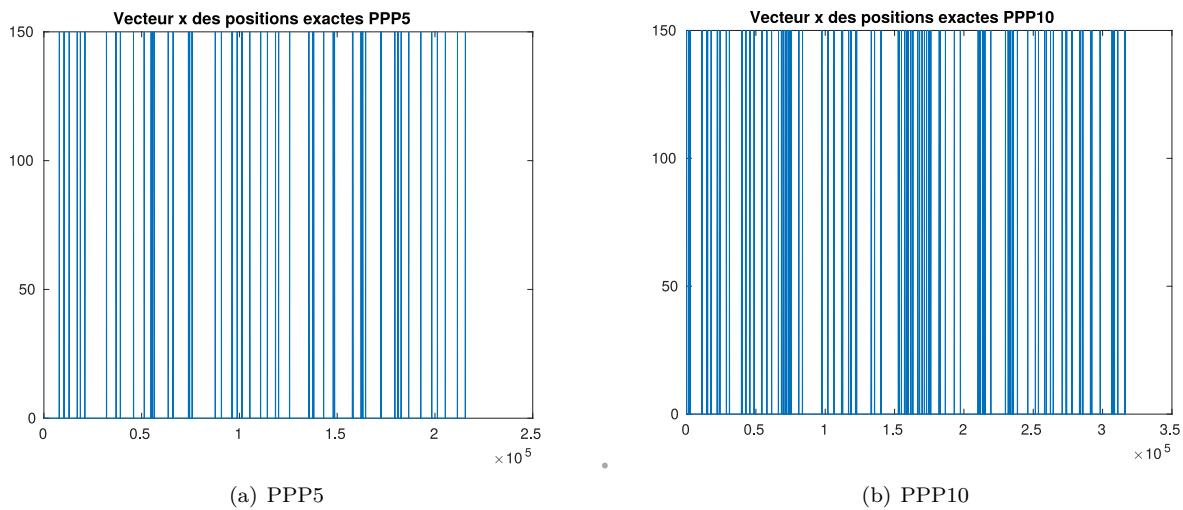
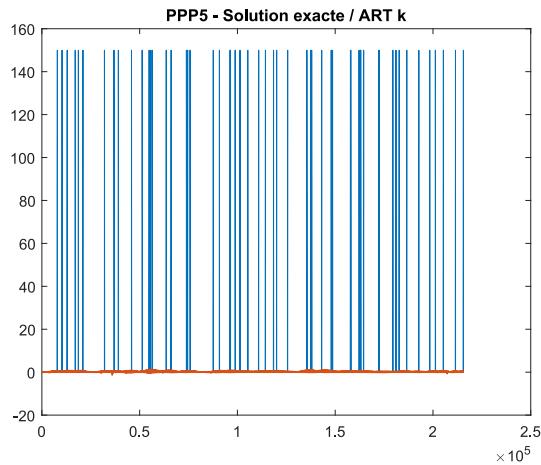


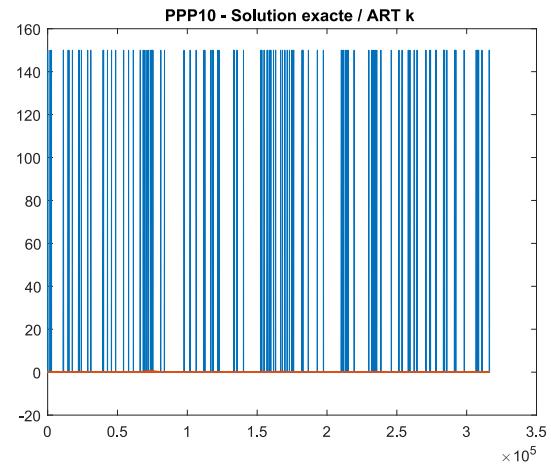
FIGURE 3.3 – Vecteur position exacte des particules pour les deux images

Tous les algorithmes ont été testés sur ces images, pour 200 itérations.

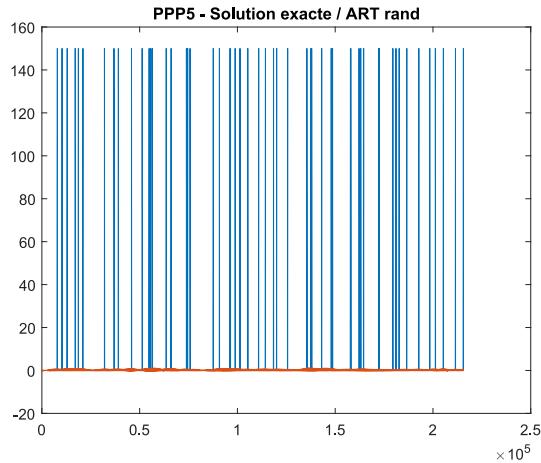
. ART kaczmarz, random et symetric



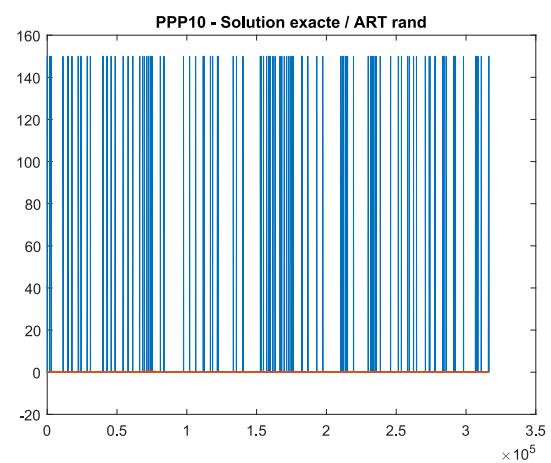
(a) PPP5 - ART k



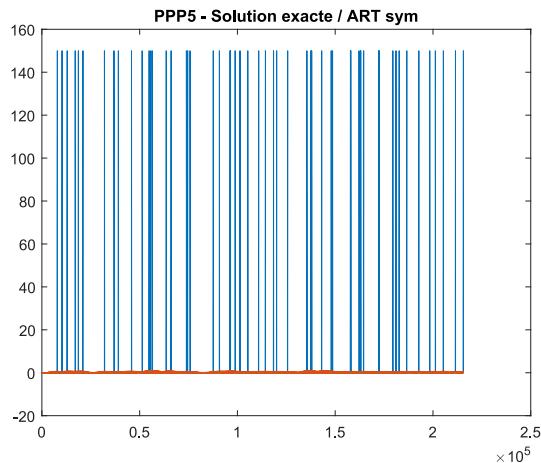
(b) PPP10 - ART k



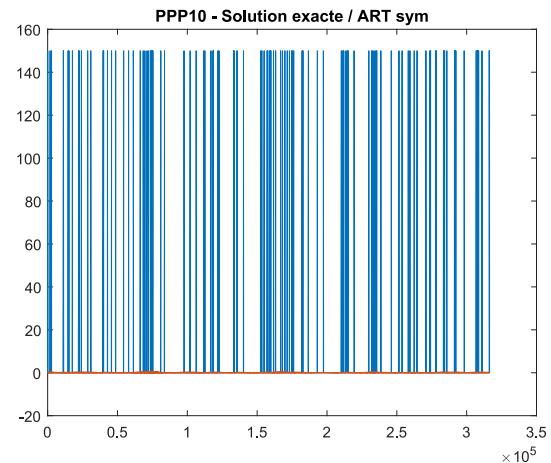
(c) PPP5 - ART rand



(d) PPP10 - ART rand



(e) PPP5 - ART sym



(f) PPP10 - ART sym

FIGURE 3.4 – Vecteur position \mathbf{x} (orange) des algorithmes ART sur les deux images PPP5 et PPP10, comparé au vecteur position exacte (bleu)

On observe une très faible amplitude qui peut s'assimiler à du bruit et la *sparsity* de ces vecteurs est exactement de 100%.

D'après le tableau (3.1), les normes L_1 des trois balayages sont similaires ($\sim 55\%$ pour PPP10 et autour de 53% pour PPP5). Cela donne l'information d'une indépendance de ART par rapport au changement de structure de la bande diagonale de la matrice \mathbf{A} . D'autre part, puisque ART produit du bruit, cette différence sera toujours présente et proportionnelle à la densité de particules. On peut l'observer sur les figures (3.5), dont sont issus les vecteurs positions de la figure (3.4).

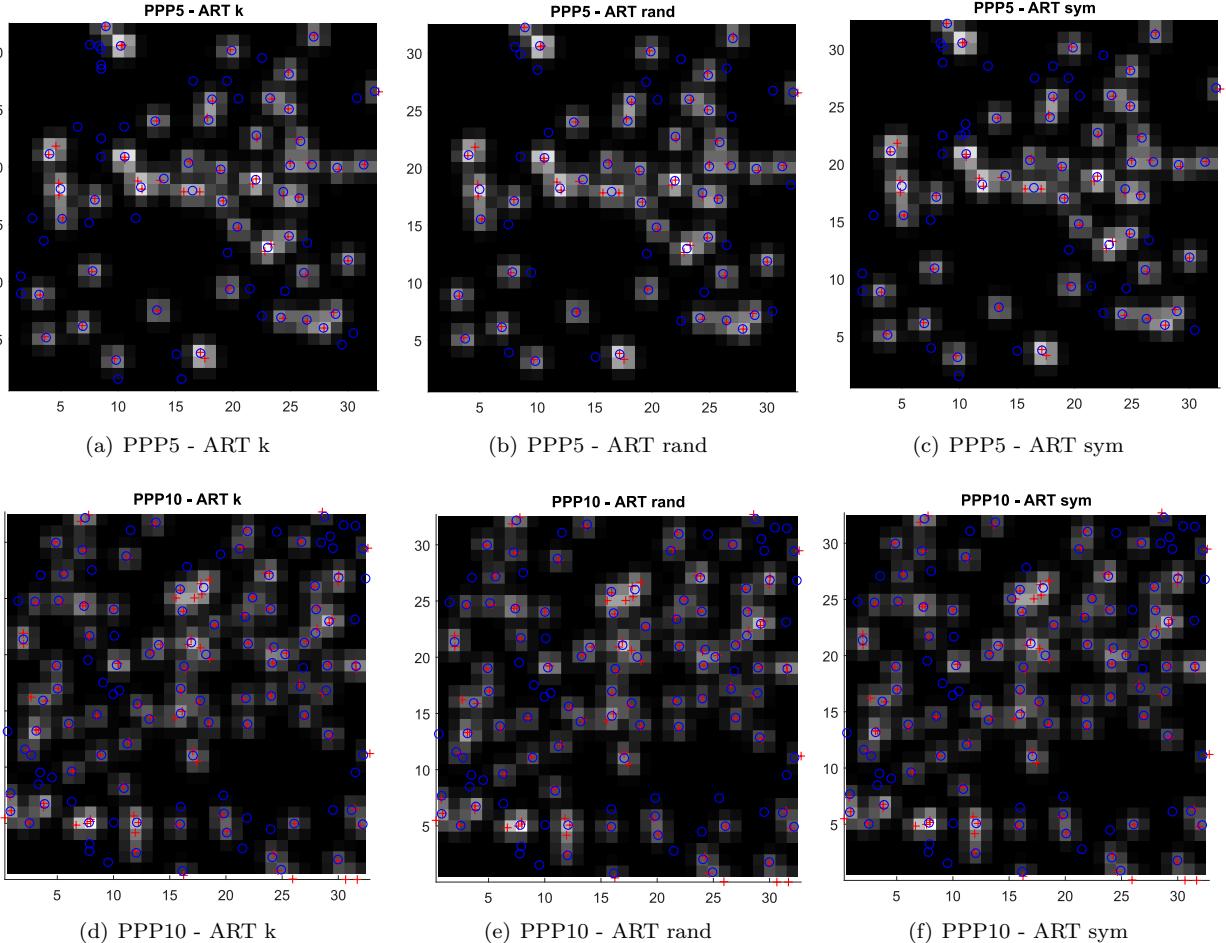


FIGURE 3.5 – Position des particules ("o" bleu) détectée par les algorithmes ART sur les deux images PPP5 et PPP10, comparé aux positions exactes ("+" rouge)

Les trois méthodes ART détectent les bonnes positions certes, mais détectent aussi des particules inexistantes.

Le système étant très sous-déterminé, soit possédant beaucoup plus de variables que d'équations (une infinité de solutions), les algorithmes ART se retrouvent à converger vers des solutions inexactes par rapport à la solution unique, ce qui est la conséquence de détections de particules inexistantes.

De tels résultats sur des images faiblement peuplées en particules (à noter que 32×32 n'est que le 32 ème d'une image utilisée par exemple dans l'article [14] (1024×1024 pixels)) font de ART, encore une fois, une méthode qui n'est pas du tout adaptée à notre problème, malgré son utilisation fréquente en reconstruction d'image.

. BBNLNS

BBNLNS, comme on peut le voir sur la figure (3.7), détecte bien les particules. En effet, l'amplitude des éléments du vecteur position (figure (3.6)) est piquée aux bons endroits. Néanmoins ce calcul subit un certain seuillage, qui sera appliqué à l'instant suivant jusqu'à ne plus produire de solutions piquées et ainsi ne plus détecter les bonnes positions.

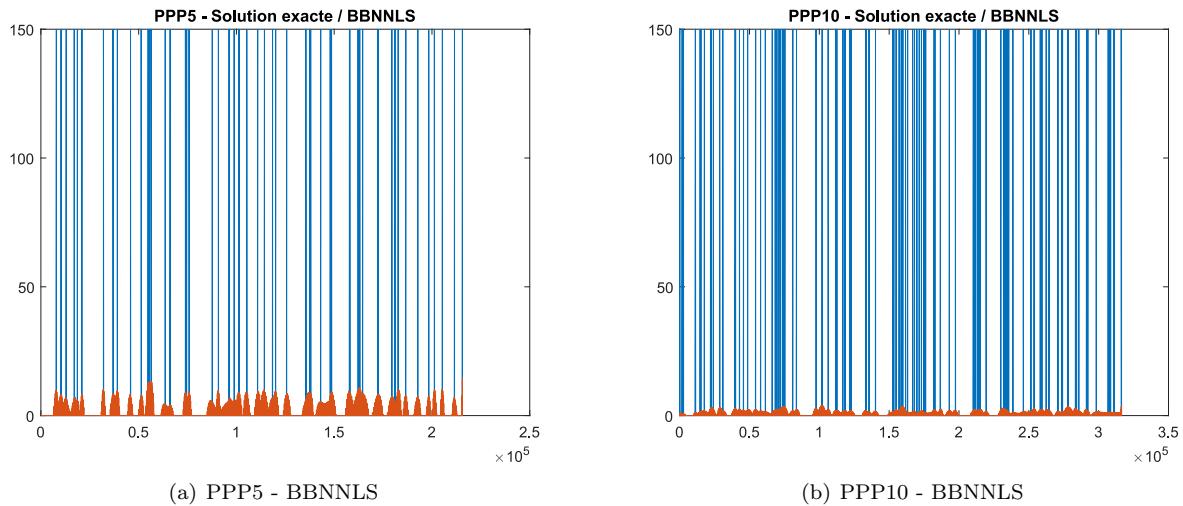


FIGURE 3.6 – Vecteur position \mathbf{x} (orange) de l'algorithme BBNLNS sur les deux images PPP5 et PPP10, comparé au vecteur position exacte (bleu)

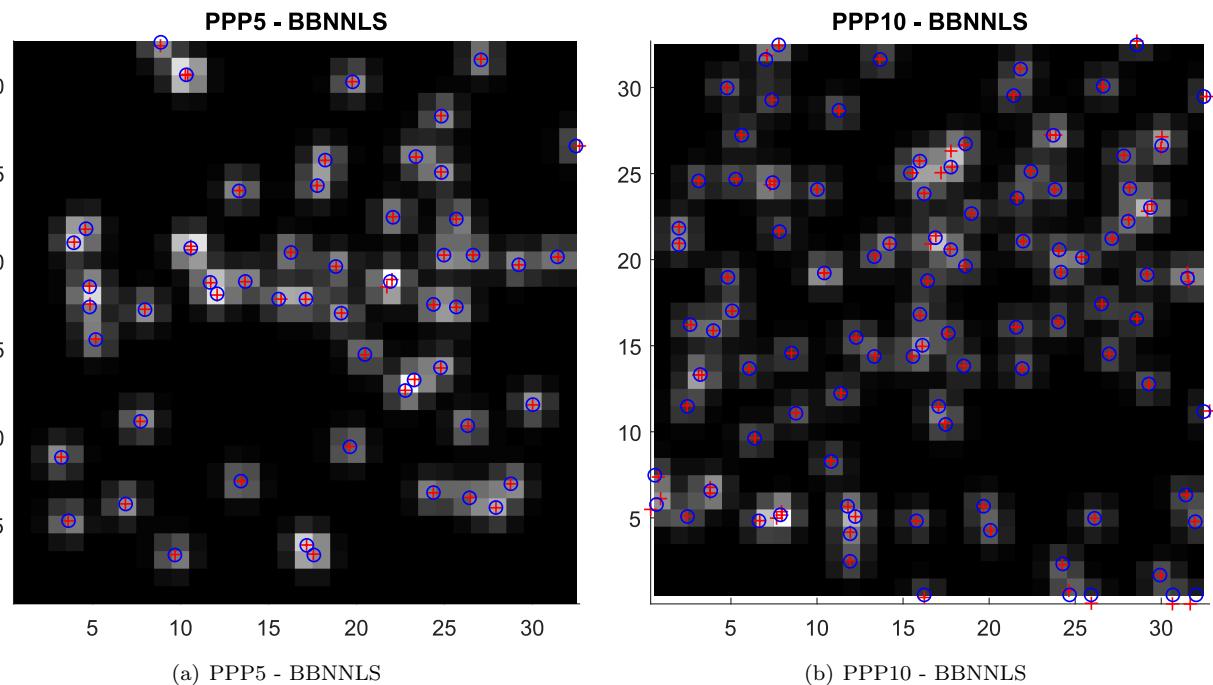


FIGURE 3.7 – Position des particules ("o" bleu) détectée par l'algorithme BBNLNS sur les deux images PPP5 et PPP10, comparé aux positions exactes ("+" rouge)

Les normes L_1 sont acceptables (4.61% pour PPP10 et 2.15% pour PPP5), et le temps de calcul est plus bas que pour ART.

Malgré sa bonne détection, BBNLNS sera rapidement inefficace lorsque la densité de particules augmentera.

. L-BFGS-B

L-BFGS-B produit les solutions les piquées des tests jusqu'à présent, ce qui est lié aux normes L_1 acceptable (voir tableau (3.2)). Cependant son temps de calcul très élevé (1633.25s pour PPP10 et 5930.06s pour PPP5) confirme les résultats des tests précédents. A noter que cette fois-ci, la légère modification de la structure de la bande diagonale permet à L-BFGS-B de produire de bonnes solutions contrairement à ART.

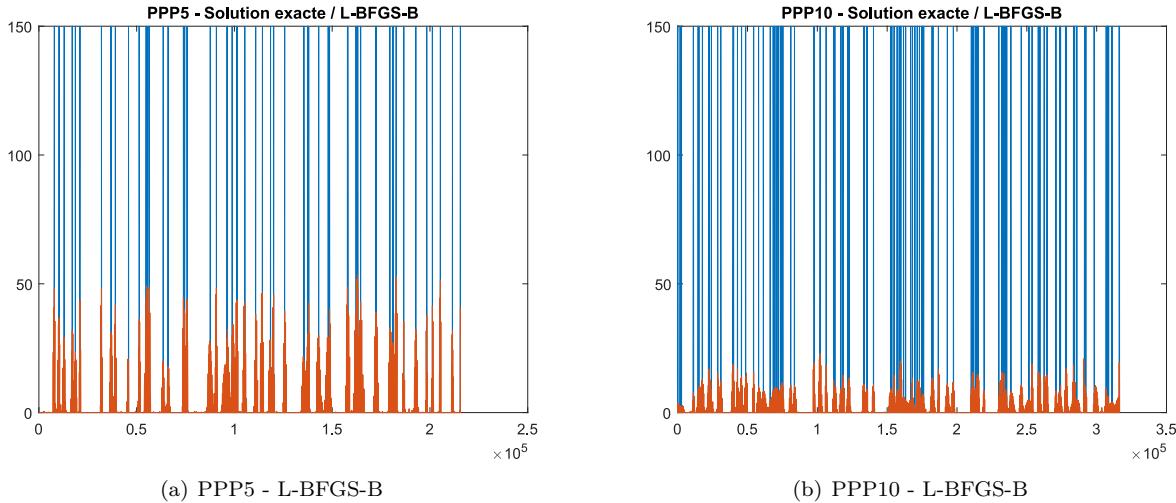


FIGURE 3.8 – Vecteur position \mathbf{x} (orange) de l'algorithme L-BFGS-B sur les deux images PPP5 et PPP10, comparé au vecteur position exacte (bleu)

. SMART

Comme expliqué en section (2.3.2.5), SMART est susceptible de produire une bonne solution, mais accompagné d'un bruit de faible amplitude, ce qui a pour conséquence de fournir des normes L_1 élevées (96% sur une matrice similaire à la matrice PPP5).

Les normes L_1 de ces tests sont 35.92% pour PPP10 et 25.89% pour PPP5. De manière analogue aux résultats de L-BFGS-B, la structure de la bande diagonale permet de retrouver une meilleure solution, ce qui permet à SMART de retrouver quand même les positions exactes des particules (figure (3.10)).

Le seuillage de cet algorithme est plus élevé que celui des algorithmes précédents. Une valeur élevée est synonyme d'efficacité de l'algorithme sur plusieurs images successives.

Aussi, SMART possède le temps de calculs le plus bas de tous les algorithmes.

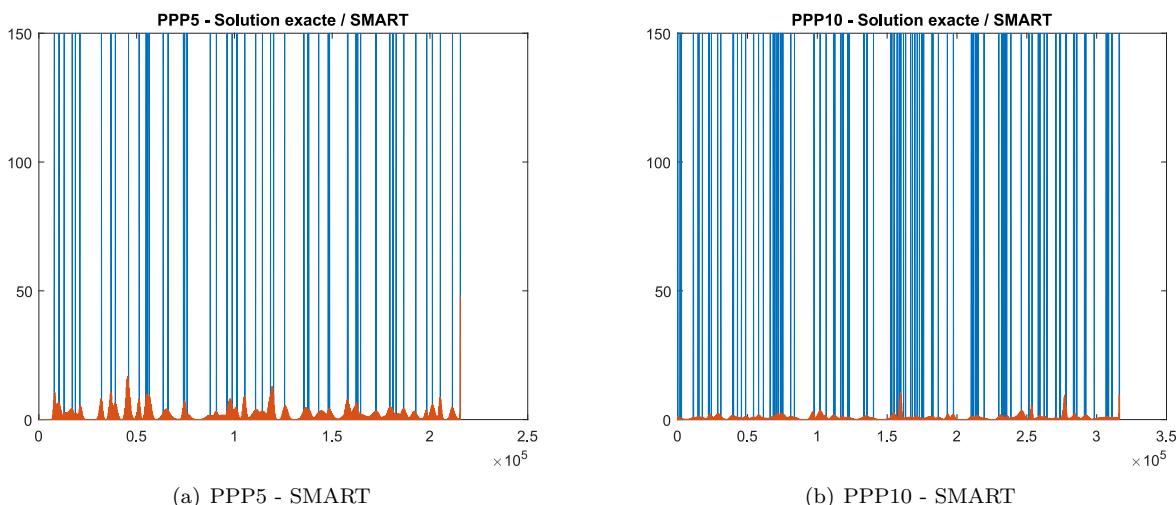


FIGURE 3.9 – Vecteur position \mathbf{x} (orange) de l'algorithme SMART sur les deux images PPP5 et PPP10, comparé au vecteur position exacte (bleu)

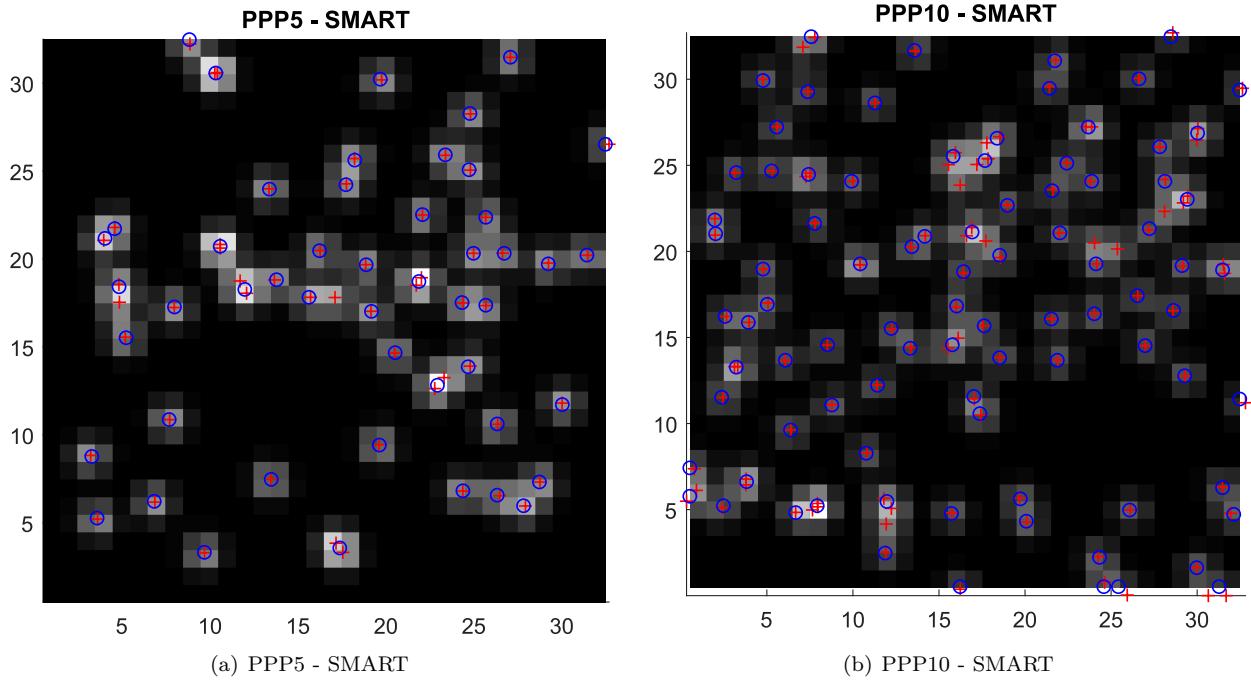


FIGURE 3.10 – Position des particules ("o" bleu) détectée par l'algorithme SMART sur les deux images PPP5 et PPP10, comparé aux positions exactes ("+" rouge)

. LSQNONNEG

LSQNONNEG fournit des solutions très piquées, d'une normes L_1 quasi-parfaite (0.14% pour PPP10 et 0.13% pour PPP5).

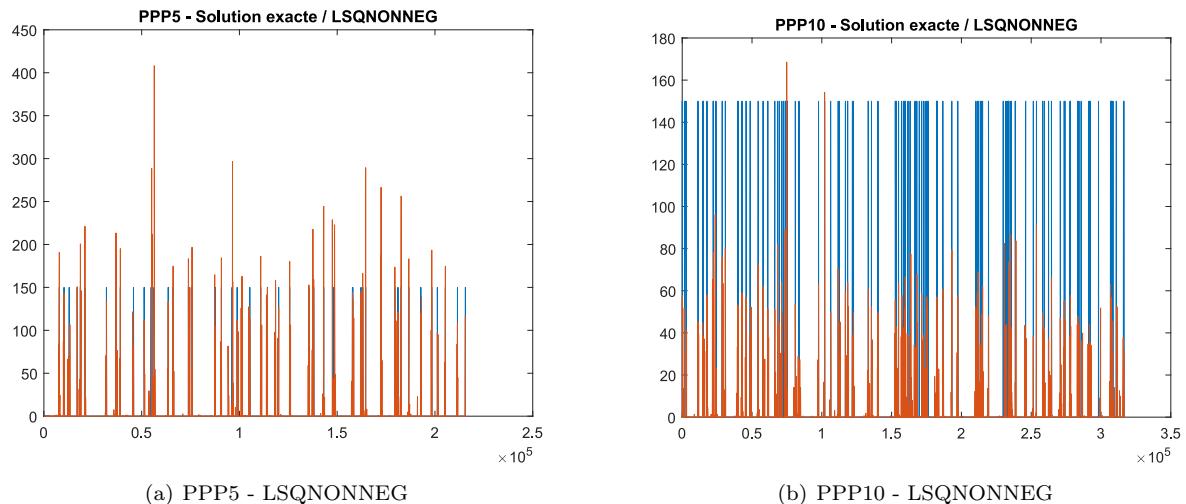


FIGURE 3.11 – Vecteur position \mathbf{x} (orange) de l'algorithme LSQNONNEG sur les deux images PPP5 et PPP10, comparé au vecteur position exacte (bleu)

Possédant un seuillage 36 fois plus élevé que SMART, LSQNONNEG peut détecter des particules avec efficacité.

Ces résultats confirment son statut de référence, et valident les résultats sur les matrices générées aléatoirement et en bande diagonale.

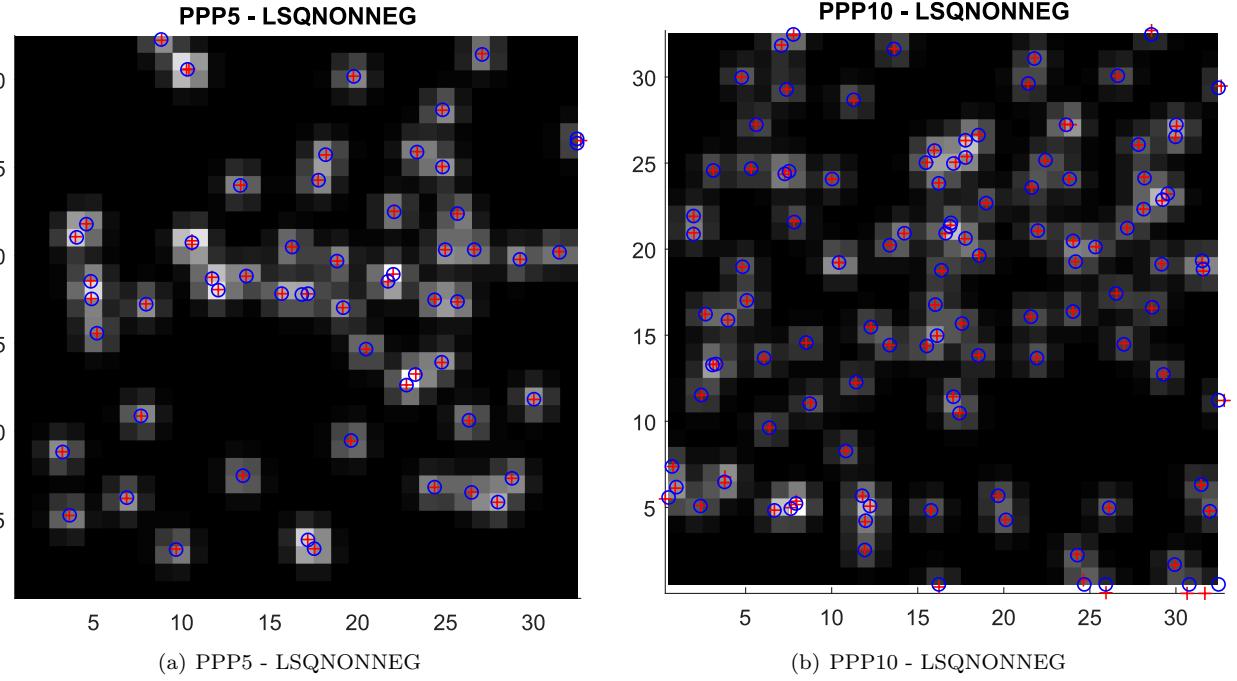


FIGURE 3.12 – Position des particules ("o" bleu) détectée par l'algorithme LSQNONNEG sur les deux images PPP5 et PPP10, comparé aux positions exactes ("+" rouge)

3.1.2 Temps de calcul et normes L_1

Images / Algorithmes	ART k	ART rand	ART sym	BBNNLS
PPP10	t : 114.99 s <i>Norme</i> _{L_1} : 55.45 %	t : 118.44 s <i>Norme</i> _{L_1} : 55.7 %	t : 216.43 s <i>Norme</i> _{L_1} : 55.45 %	t : 89.98 s <i>Norme</i> _{L_1} : 4.61 %
	t : 79.63 s <i>Norme</i> _{L_1} : 54.07 %	t : 88.36 s <i>Norme</i> _{L_1} : 52.8 %	t : 157.9 s <i>Norme</i> _{L_1} : 52.52 %	t : 53.43 s <i>Norme</i> _{L_1} : 2.15 %

TABLE 3.1 – Temps des calcul et norme L_1 des algorithmes ART et BBNNLS sur les deux images de densité de particules respective PPP5 et PPP10

Images / Algorithmes	L-BFGS-B	LSQNONNEG	SMART
PPP10	t : 1633.25 s <i>Norme</i> _{L_1} : 3.26 %	t : 193.05 s <i>Norme</i> _{L_1} : 0.1425 %	t : 50.82 s <i>Norme</i> _{L_1} : 35.92 %
	t : 5930.06 s <i>Norme</i> _{L_1} : 2.41 %	t : 67.08 s <i>Norme</i> _{L_1} : 0.1299 %	t : 36.81 s <i>Norme</i> _{L_1} : 25.89 %

TABLE 3.2 – Temps des calcul et norme L_1 des algorithmes L-BFGS-B, LSQNONNEG et SMART sur les deux images de densité de particules respective PPP5 et PPP10

Les tableaux ci-dessus nous permettent de conclure quant au choix des algorithmes pour la suite. Les algorithmes ART ne sont pas adaptés à notre problème. L-BFGS-B est prometteur au niveau de la solution fournie mais son temps de calcul est trop élevé pour des images aussi simples. BBNNLS et SMART fournissent de bonnes solutions, mais SMART possède le temps de calcul le plus bas. Et LSQNONNEG est toujours l'algorithme de référence.

C'est donc LSQNONNEG et SMART qui seront testés dans la suite sur des images plus denses en pixels et en particules dans le but de reconstruire le champ de vitesse.

3.2 Reconstruction du champ de vecteurs vitesses

Pour finir, dans cette section LSQNONNEG et SMART seront utilisés pour reconstruire le champ de vecteur vitesses d'une paire d'images (256×256 pixels) successives de particules.

L'écoulement testé est le double vortex de Lamb-Oseen [21]. Voici-ci dessous les deux images avec les particules détectées.

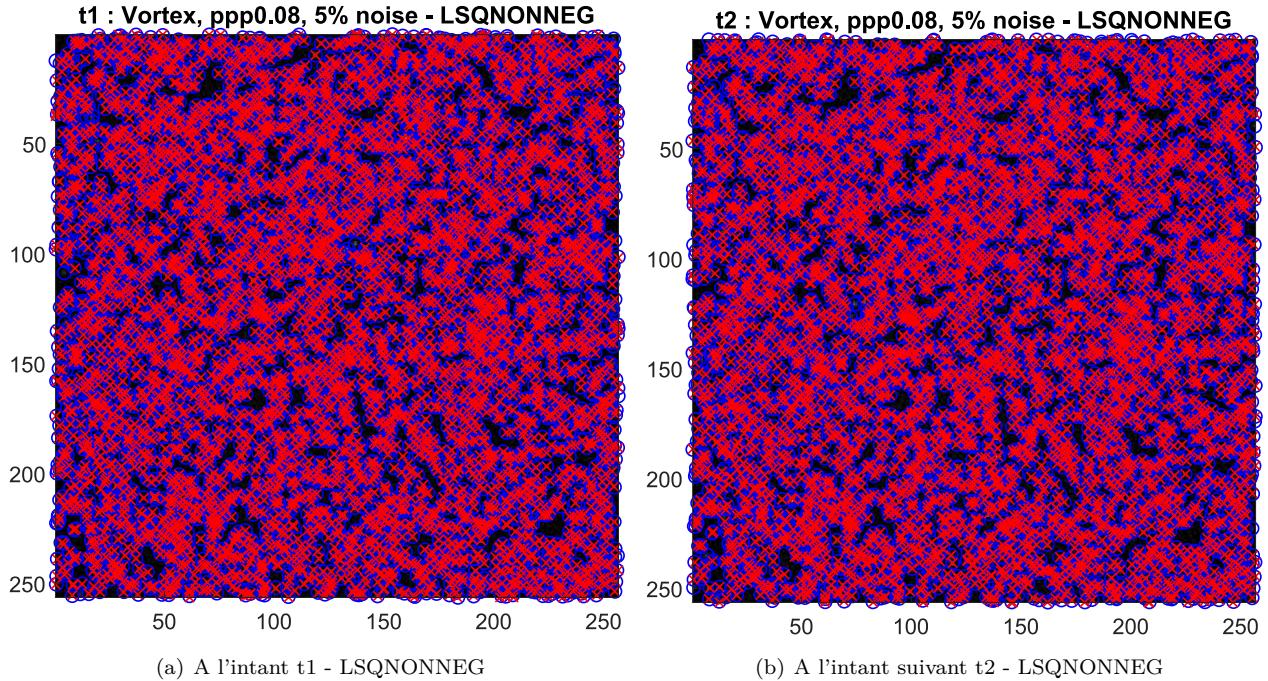


FIGURE 3.13 – Position des particules ("o" bleu) détectée par l'algorithme LSQNONNEG sur les deux images successives, comparé aux positions exactes ("+" rouge)

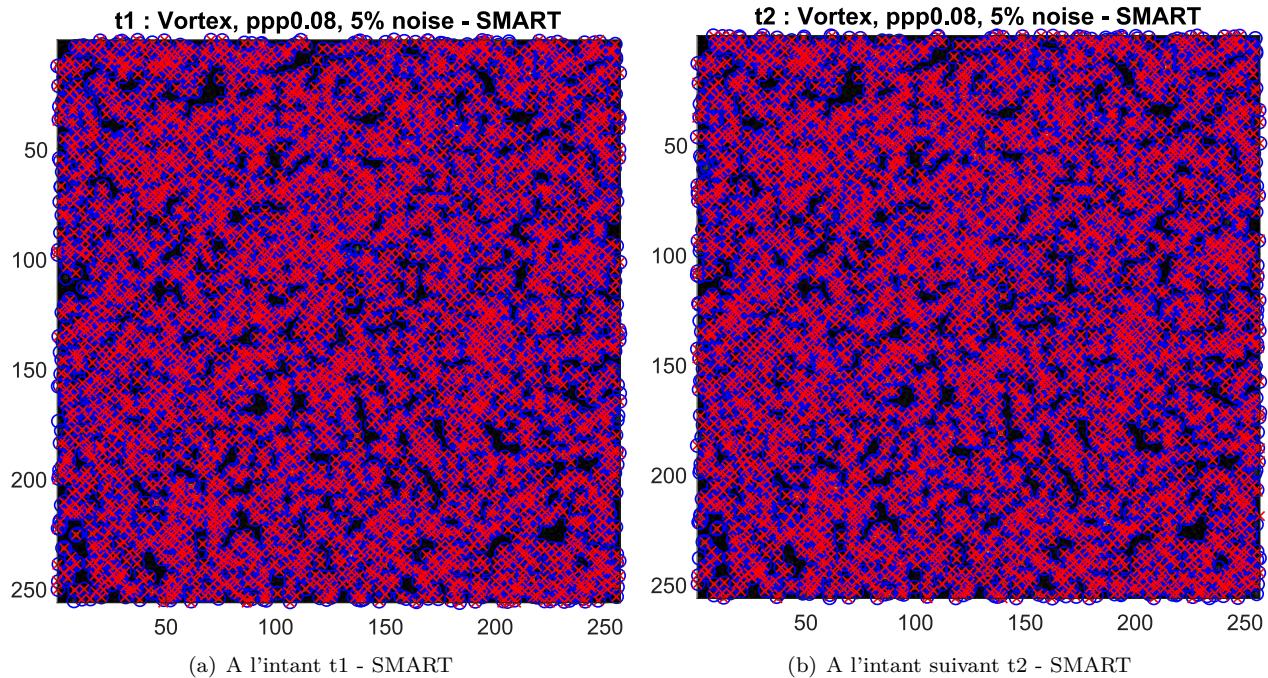


FIGURE 3.14 – Position des particules ("o" bleu) détectée par l'algorithme SMART sur les deux images successives, comparé aux positions exactes ("+" rouge) - Densité de 0.08 particules par pixel.

Après détection, une méthode de tracking est employée, notamment à travers l'algorithme ICCRM : Inte-

grated Cross Corelation Relaxation Methods (voir [6] pour les détails).

Ci-dessous l'image réaliste des vortex de Lamb-Oseen :

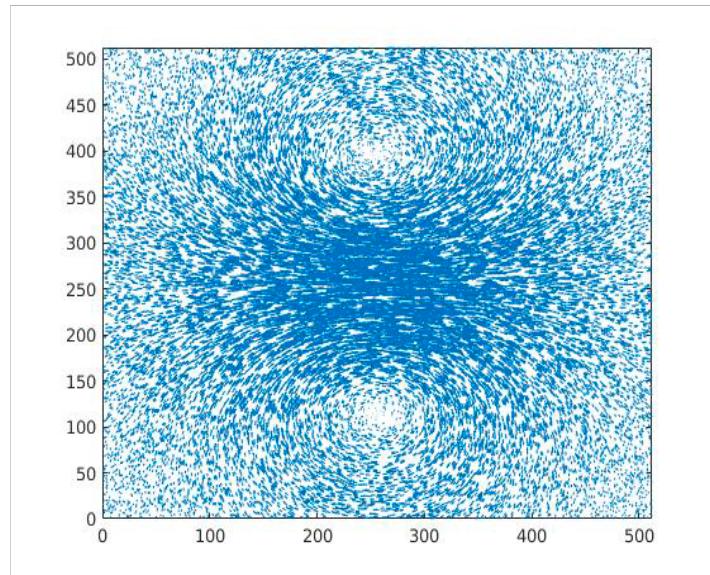


FIGURE 3.15 – Double vortex de Lamb-Oseen

Une fois calculés, les vecteurs vitesses montrent clairement les vortex :

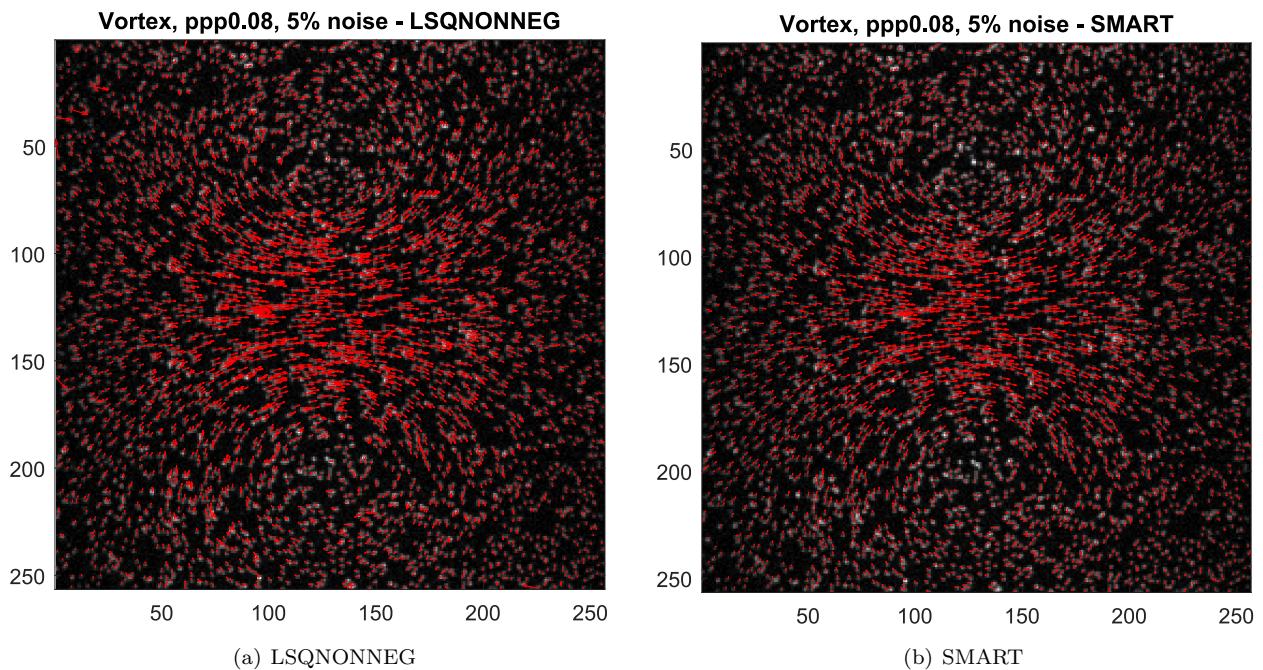


FIGURE 3.16 – Champs de vitesses reconstruit à partir des détections des algorithmes LSQNONNEG et SMART

Sur la même image, on peut observer quelques différences. LSQNONNEG semble détecter plus de particules que SMART, notamment au milieu de l'image. Cela est du au processus de seuillage, comme dit plus haut, qui tranche la solution fournie par SMART. Celui-ci finira par ne plus détecter les bonnes particules à mesure qu'on avance dans le tracking du champs de vitesses.

Bilan et perspectives

L'algorithme LSQNONNEG [32] [37] a donc été confronté à plusieurs algorithmes traitant le problème d'inversion linéaire faisant intervenir une matrice dite *short & fat* de grande dimension (problème très sous-déterminé) et dont le but est de détecter la position des particules à travers le vecteur inconnu du système \mathbf{x} soumis à des conditions de non-négativité et de *sparsity*.

Tout d'abord, une recherche bibliographique a été entreprise. De nombreux algorithmes existent dans la littérature, dans plusieurs domaines différents notamment en imagerie médicale, traitement de l'image et du signal, ou encore en machine learning [44]. Certains conçus pour ce type de problème (ART : tomographie, reconstruction d'image), d'autres à but plus mathématique et dont l'application pour varier (BBNNLSS, LBFGS-B) ou encore des extensions ou versions modifiées de LSQNONNEG (FNNLS) ont été choisis pour l'étude. De nombreux autres algorithmes ont été mis de côté car ne remplissant pas les conditions de bases du problème physique.

Ainsi, les algorithmes implémentés, des tests ont été entrepris. L'idée était de commencer par des cas simples de matrices et se rapprocher petit à petit du problème très sous-déterminé et *sparse* dont le but est l'étude de l'influence de paramètre tels que la *sparsity* de la matrice ou du vecteur recherché et le rapport $\frac{N}{M}$ sur la robustesse et le temps de calcul de ces algorithmes. Les premiers tests, sur des matrices dont la structure est générée aléatoirement, ont montré l'efficacité de LSQNONNEG sur les autres algorithmes, en termes de temps en calcul et de détection des bonnes positions des particules. Ce résultat s'est confirmé sur les tests suivants, où cette fois-ci la matrice est à bande diagonale (similaire à la matrice poids du problème physique) et où l'on a augmenté la dimension de celle ci. En effet, LSQNONNEG a fourni des résultats très satisfaisants. Les algorithmes de tomographie ont été testé aussi, notamment ART et SMART. Ce dernier a aussi fourni de bons résultats comparé à son cousin ART qui ne produisait que du bruit. SMART a aussi un temps de calcul plus bas que LSQNONNEG. Cependant il ne fournit pas de solutions assez piquées, ce qui le différencie de son concurrent pour ce problème. Ce dernier résultat a pu être validé lors de l'utilisation de ces algorithmes sur des images réelles de différentes tailles et de densité de particules différentes.

Cette étude a donc confirmé l'efficacité de LSQNONNEG face à ce problème de détection de particules. Le but étant de reconstruire un champ de vitesse et pouvoir étudier la turbulence. Néanmoins, malgré son efficacité et sa robustesse, lorsqu'il s'agit d'images très grandes (typiquement 1024×1024 pixels, voir [14]) LSQNONNEG présente des problèmes de temps de calculs. Ceci dit, il était crucial de procéder à ce genre de tests et comparaison avant de passer à l'étape suivante. Une thèse est en préparation concernant la caractérisation expérimentale par PTV d'une turbulence issue de l'interaction de jets turbulents, et l'algorithme LSQNONNEG reste est un bon candidat à la détection des particules pour ce sujet.

Annexes

Annexe 1

1 Algorithmes

Les algorithmes utilisés lors de l'étude sont détaillés ci-dessous sous forme de pseudo-code ou d'étapes à suivre (LSQNONNEG est déjà détaillé au chapitre 1)

1.1 FNNLS

FNNLS est très proche de LSQNONNEG. Le seul changement se fait au calcul de ω à l'étape de l'initialisation et à la fin, et s_P à l'étape 10 et 15.

Algorithm 2 FNNLS

```
1 : Input :  $\mathbf{A} \in \mathbb{R}^{N \times M}$ ,  $\mathbf{b} \in \mathbb{R}^N$ 
2 : Output :  $x^* \geq 0$  tel que  $x^* = \operatorname{argmin} \| \mathbf{Ax} - \mathbf{b} \|^2$ 
3 : Initialisation :  $\mathbf{x} = \mathbf{0}$ ,  $\mathbf{w} = \nabla f(x) = \mathbf{A}^T \mathbf{b} - (\mathbf{A}^T \mathbf{A})\mathbf{x}$ ,  $P = \emptyset$ ,  $R = \{1, 2, \dots, M\}$ 
4 :  $\triangleright P$  et  $R$  sont respectivement l'ensemble passif et actif
5 : Répéter
6 :   procedure FNNLS( $A, b$ )
7 :     Si  $R \neq \emptyset \wedge [\max_{i \in R}(w_i) > \text{tolerance}]$ 
8 :        $j = \operatorname{argmax}_{i \in R}(w_i)$ 
9 :       Inclure les indices  $j$  dans  $P$  et les retirer de  $R$ 
10 :       $s_P = [(\mathbf{A}_P)^T \mathbf{A}_P]^{-1} (\mathbf{A}_P^T \mathbf{b})_P$   $\triangleright \mathbf{A}_P$  est la matrice associée à l'ensemble passif  $P$ 
11 :      Si  $\min(s_P) \leq 0$ 
12 :         $\alpha = -\min_{i \in P}[x_i / (x_i - s_i)]$ 
13 :         $\mathbf{x} = \mathbf{x} + \alpha(\mathbf{s} - \mathbf{x})$ 
14 :        Mise à jour de  $R$  et  $P$ 
15 :         $s_P = [(\mathbf{A}_P)^T \mathbf{A}_P]^{-1} (\mathbf{A}_P^T \mathbf{b})_P$ 
16 :       $s_R = \mathbf{0}$ 
17 :       $\mathbf{x} = \mathbf{s}$ 
18 :       $\mathbf{w} = \mathbf{A}^T(\mathbf{b} - \mathbf{Ax})$ 
```

1.2 BB-NNLS

Cet algorithme utilise un paramètre β que les auteurs de [ref] appellent un DS (Diminishing Scalar), permettant d'optimiser la recherche de solutions.

Algorithm 3 BB-NNLS

```
1 : Input :  $\mathbf{x}^0$  et  $\mathbf{x}^1$ 
2 : Pour :  $i = 1, \dots$  jusqu'à ce que :  $\max |\nabla f(\mathbf{x}^k)| < \epsilon$  (KKT)
3 : Répéter
4 :   procedure BBNNLS( $A, b$ )
5 :      $\hat{\mathbf{x}}^0 \leftarrow \mathbf{x}^{i-1}$  et  $\hat{\mathbf{x}}^1 \leftarrow \mathbf{x}^i$ 
6 :     Pour  $j = 1, \dots, N$ 
7 :       Calcul du BB-step  $\alpha^j$  en utilisant (1.8)
8 :        $\hat{\mathbf{x}}^{j+1} \leftarrow \hat{\mathbf{x}}^j - \beta^i \cdot \alpha^j \nabla f(\hat{\mathbf{x}}^j)_+$ 
9 :     Si  $\hat{\mathbf{x}}^N$  satisfait (10) de l'article [ref] alors :
10 :       $\mathbf{x}^{i+1} \rightarrow \hat{\mathbf{x}}^N$  et  $\beta^{i+1} \leftarrow \beta^i$ 
11 :    Sinon  $\beta^{i+1} \leftarrow \eta \beta^i$ , avec  $\eta \in (0, 1)$ .
```

1.3 L-BFGS-B

Cette version de BFGS, utilise de nombreux autres algorithmes nécessaire à son fonctionnement, tel que l'algorithme du calcul du point de Cauchy (section 4 [ref]), ou encore les méthodes de minimisation d'un modèle ((2.1) de [ref] équivalent à la fonction objective à minimiser) 'direct primal method', 'conjugate gradient' et 'dual method' (section 5 [ref]). Les détails sous-jacents peuvent se trouver dans l'article [ref], dont les différentes étapes de L-BFGS-B sont reprises ci-dessous (les équations utilisées seront notés tel que dans l'article) :

Choisir \mathbf{x}_0 et un nombre m nécessaire au stockage des corrections de mémoires. Après avoir définis les bornes, à $k = 0$:

- 1. Si le test de convergence (6.1) n'est pas satisfait, passer à l'étape 2.
- 2. Calculer le point de Cauchy avec l'algorithme CP.
- 3. Procéder à une au calcul de d_k (voir (4.7), l'équivalent de l'étape 12 de LSQNONNEG) avec l'une des trois méthodes.
- 4. Procéder à la recherche de directions optimale à la minimisation locale du modèle sur les bords, puis calculer le step λ_k et fixer : $\mathbf{x}_{k+1} = \mathbf{x}_k + \lambda_k$.
- 5. Calculer $\nabla f(\mathbf{x}_{k+1})$.
- 6. Calculer les paires de corrections y_k et s_k (tout en satisfaisant la condition (3.9) de [ref]) et les rajouter à leur matrice respectives Y_k et S_k .
- 7. Mettre à jour les matrices de transition L_k (3.5 de [ref]) et R_k (3.8 de [ref]), et fixer $\theta = y_k^T y_k / y_k^T s_k$.
- 8. Passer à l'itération suivante → étape 1.

1.4 ART k,rand et sym

De manière basique ART suit les étapes suivantes :

- 1. Initialisation : x^0
- 2. Choisir une méthode de balayage (kaczmarz, random ou symmetric) du tableau 1.1.
- 3. Pour k correspondant à la méthode choisie, calculer x^{k+1} par (1.9).
- 4. Si $r^k = b - Ax^k < \tau\delta$ (δ est une estimation de l'erreur relative L_2 et τ et un facteur de sécurité légèrement au dessus de 1) la solution est retrouvée, sinon → étape 1.

1.5 SMART

De la même manière SMART suit les mêmes étapes que ART mais avec une méthode de balayage classique :

- 1. Initialisation : x^0
- 2. Pour $k = 1, \dots, kMax$, calculer x^{k+1} par (1.12)
- 3. L'algorithme s'arrête à $kMax$ mais on peut y implémenter un critère d'arrêt tel qu'une faible valeur du résidu.

Annexe 2

1 Résultats des tests sur matrice générée aléatoirement

En complément de l'étude sur une *sparsity* de **A** de 5% présente à la section 2.2.3, et comme mentionné nous avons procédé aux mêmes tests sur une matrice de *sparsity* de 10% et 20%. Les résultats étant similaires hormis quelques temps de calculs plus élevés, et une différence par rapport à la solution exacte augmenta avec la *sparsity* de **A**.

Nous affichons ci-dessous, les tableaux des résultats la matrice T-4 des deux *sparsity*.

1.1 10%

T-4 / <i>sparsity</i>	2%	1%	0.5%	0.15%
Lsqnonneg	<i>Err</i> : 1.08e+00 <i>sparsitysol</i> : 5.e-03 t : 0.204 s <i>Nbit</i> : 100 <i>Norm_{L1}</i> : 2.480 %	<i>Err</i> : 1.14e+00, <i>sparsitysol</i> : 5.e-03 t : 0.213 s <i>Nbit</i> : 100 <i>Norm_{L1}</i> : 1.490 %	<i>Err</i> : 1.24e+00 <i>sparsitysol</i> : 5.e-03 t : 0.205 s <i>Nbit</i> : 101 <i>Norm_{L1}</i> : 0.980 %	<i>Err</i> : 1.22e+00 <i>sparsitysol</i> : 4.8e-03 t : 0.456 s <i>Nbit</i> : 157 <i>Norm_{L1}</i> : 0.610 %
Fnnls	<i>Err</i> : 1.08e+00 <i>sparsitysol</i> : 5.e-03 t : 0.149 s <i>Nbit</i> : 100 <i>Norm_{L1}</i> : 2.470 %	<i>Err</i> : 1.19e+00 <i>sparsitysol</i> : 5.e-03 t : 0.159 s <i>Nbit</i> : 100 <i>Norm_{L1}</i> : 1.490 %	<i>Err</i> : 1.29e+00 <i>sparsitysol</i> : 5.e-03 t : 0.136 s <i>Nbit</i> : 100 <i>Norm_{L1}</i> : 0.98 %	<i>Err</i> : 1.15e+00 <i>sparsitysol</i> : 5.35e-03 t : 50.567 s <i>Nbit</i> : 107 <i>Norm_{L1}</i> : 0.495 %
BBnls	<i>Err</i> : 7.63e+00 <i>sparsitysol</i> : 9.99e-01 t : 0.036 s <i>Nbit</i> : 17 <i>Norm_{L1}</i> : 97.995 %	<i>Err</i> : 8.82e+00 <i>sparsitysol</i> : 9.59e-01 t : 0.048 s <i>Nbit</i> : 24 <i>Norm_{L1}</i> : 94.99%	<i>Err</i> : 1.08e+01 <i>sparsitysol</i> : 9.48e-01 t : 0.097 s <i>Nbit</i> : 42 <i>Norm_{L1}</i> : 94.365 %	<i>Err</i> : 5.03e+00 <i>sparsitysol</i> : 7.92e-02 t : 0.554 s <i>Nbit</i> : 226 <i>Norm_{L1}</i> : 7.920 %
L-bfgs-B	<i>Err</i> : 7.63e+00 <i>sparsitysol</i> : 9.90e-01 t : 0.295 s <i>Nbit</i> : 12 <i>Norm_{L1}</i> : 97.010 %	<i>Err</i> : 9.18e+00 <i>sparsitysol</i> : 9.61e-01 t : 0.440 s <i>Nbit</i> : 17 <i>Norm_{L1}</i> : 95.160 %	<i>Err</i> : 9.75e+00 <i>sparsitysol</i> : 9.37e-01 t : 1.093 s <i>Nbit</i> : 29 <i>Norm_{L1}</i> : 93.29 %	<i>Err</i> : 5.91e+00 <i>sparsitysol</i> : 7.04e-01 t : 29.639 s <i>Nbit</i> : 366 <i>Norm_{L1}</i> : 70.265 %

TABLE 3.3 – Résultats des algorithmes sur la matrice T-4 (10%) *sparsity* de **A**

1.2 20%

T-4 / sparsity	2%	1%	0.5%	0.15%
Lsqnonneg	<i>Err</i> : 1.06e+00 <i>sparsitysol</i> : 5.e-03 t : 0.215 s <i>Nbit</i> : 100 <i>Norm_{L1}</i> : 2.480 %	<i>Err</i> : 1.15e+00, <i>sparsitysol</i> : 5.e-03 t : 0.168 s <i>Nbit</i> : 100 <i>Norm_{L1}</i> : 1.490 %	<i>Err</i> : 1.24e+00 <i>sparsitysol</i> : 5.e-03 t : 0.172 s <i>Nbit</i> : 101 <i>Norm_{L1}</i> : 0.990 %	<i>Err</i> : 1.22e+00 <i>sparsitysol</i> : 5.e-03 t : 0.245 s <i>Nbit</i> : 117 <i>Norm_{L1}</i> : 0.630 %
Fnlnls	<i>Err</i> : 1.05e+00 <i>sparsitysol</i> : 5.e-03 t : 0.132 s <i>Nbit</i> : 100 <i>Norm_{L1}</i> : 2.480 %	<i>Err</i> : 1.12e+00 <i>sparsitysol</i> : 5.e-03 t : 0.129 s <i>Nbit</i> : 100 <i>Norm_{L1}</i> : 1.480 %	<i>Err</i> : 1.23e+00 <i>sparsitysol</i> : 5e-03 t : 0.156 s <i>Nbit</i> : 100 <i>Norm_{L1}</i> : 0.99 %	<i>Err</i> : 1.62e+00 <i>sarsitysol</i> : 5.25e-03 t : 40.656 s <i>Nbit</i> : 107 <i>Norm_{L1}</i> : 0.59 %
BBnnls	<i>Err</i> : 7.66e+00 <i>sparsitysol</i> : 9.99e-01 t : 0.026 s <i>Nbit</i> : 12 <i>Norm_{L1}</i> : 97.905 %	<i>Err</i> : 8.58e+00 <i>sparsitysol</i> : 9.01e-01 t : 0.083 s <i>Nbit</i> : 24 <i>Norm_{L1}</i> : 89.25%	<i>Err</i> : 9.69e+01 <i>sparsitysol</i> : 9.17e-01 t : 0.124 s <i>Nbit</i> : 58 <i>Norm_{L1}</i> : 91.2 %	<i>Err</i> : 5.47e+00 <i>sparsitysol</i> : 2.42e-01 t : 0.465 s <i>Nbit</i> : 212 <i>Norm_{L1}</i> : 24.195 %
L-bfgs-B	<i>Err</i> : 7.69e+00 <i>sparsitysol</i> : 9.94e-01 t : 0.451 s <i>Nbit</i> : 14 <i>Norm_{L1}</i> : 97.41 %	<i>Err</i> : 9.78e+00 <i>sparsitysol</i> : 9.72e-01 t : 0.577 s <i>Nbit</i> : 18 <i>Norm_{L1}</i> : 96.230 %	<i>Err</i> : 1.02e+01 <i>sparsitysol</i> : 9.24e-01 t : 0.660 s <i>Nbit</i> : 20 <i>Norm_{L1}</i> : 92 %	<i>Err</i> : 5.78e+00 <i>sparsitysol</i> : 8.25e-01 t : 5.175 s <i>Nbit</i> : 87 <i>Norm_{L1}</i> : 82.455 %

TABLE 3.4 – Résultats des algorithmes sur la matrice T-4 (20%) *sparsity* de **A**

Bibliographie

- [1] *Minima of Functions of Several Variables with Inequalities as Side Constraints.* Dept. of Mathematics, Univ. of Chicago, 1939.
- [2] *Principles of Computerized Tomographic Imaging*, chapter Chapter 7 : Algebraic Reconstruction Algorithms. SIAM, 2001.
- [3] C. H. Atkinson and J. Soria. Algebraic reconstruction techniques for tomographic particle image velocimetry. *Proceedings of the 16th Australasian Fluid Mechanics Conference, 16AFMC*, (December) :191–198, 2007.
- [4] A. Bjorck and Elfving. Accelerated projection methods for computing pseudoinverse solutions of systems of linear equations. *BIT*, 1979.
- [5] J Brazilai and J. M. Borwein. Two-Point Step Size Gradient Methods. *IMA Journal of Numerical Analysis*, 8(1) :141–148, 01 1988.
- [6] W Brevis, Yarko Niño, and G H. Jirka. Integrating cross-correlation and relaxation algorithms for particle tracking velocimetry. *Experiments in Fluids*, 50 :135–147, 03 2011.
- [7] Rasmus Bro and Sijmen De Jong. A fast non-negativity-constrained least squares algorithm. *Journal of Chemometrics*, 11(5) :393–401, 1997.
- [8] Alfred M. Bruckstein, David L. Donoho, and Michael Elad. From sparse solutions of systems of equations to sparse modeling of signals and images. *SIAM Rev.*, 51(1) :34–81, February 2009.
- [9] Alfred M. Bruckstein, Michael Elad, and Michael Zibulevsky. On the uniqueness of nonnegative sparse solutions to underdetermined systems of equations. *IEEE Transactions on Information Theory*, 54(11) :4813–4820, 2008.
- [10] Richard H. Byrd, Peihuang Lu, Jorge Nocedal, and Ciyou Zhu. A Limited Memory Algorithm for Bound Constrained Optimization. *SIAM Journal on Scientific Computing*, 16(5) :1190–1208, 1995.
- [11] C. Byrne. The Multiplicative Algebraic Reconstruction Technique Solves the Geometric Programming Problem. *October*, pages 1–11, 2007.
- [12] Charles L Byrne. Iterative Algorithms in Inverse Problems. *UMass Library*, 2006.
- [13] Charles L. Byrne. A first course in optimization. *A First Course in Optimization*, pages 1–289, 2014.
- [14] Adam Chomet, Jean François Krawczynski, and Philippe Druault. Particle image reconstruction for particle detection in particle tracking velocimetry. *Measurement Science and Technology*, 29(12), 2018.
- [15] Donghui Chen and Robert J. Plemmons. Nonnegativity constraints in numerical analysis. *The Birth of Numerical Analysis*, pages 109–140, 2009.
- [16] Jon Dattoro. *Optimization euclidean*.
- [17] Keith Dillon and Yu Ping Wang. Imposing uniqueness to achieve sparsity. *Signal Processing*, 123 :1–8, 2016.
- [18] G E. Elsinga, Fulvio Scarano, Bernhard Wieneke, and Bas Oudheusden. Tomographic particle image velocimetry. *Experiments in Fluids*, 41 :933–947, 12 2006.
- [19] D Ching-Lung F. and Saunders M. LSMR : An iterative algorithm for sparse least-squares problems. *SIAM Journal on Scientific Computing*, Vol. 33, N :pp. 2950–2971, 2012.
- [20] Vojtěch Franc, Václav Hlaváč, and Mirko Navara. Sequential coordinate-wise algorithm for the non-negative least squares problem. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 3691 LNCS(i) :407–414, 2005.
- [21] J. Förste. Saffman, p. g., vortex dynamics, cambridge etc., cambridge university press 1992. xi, 311 pp., č 35.00 h/b. isbn 0-521-42058-x (cambridge monographs on mechanics and applied mathematics). *ZAMM - Journal of Applied Mathematics and Mechanics / Zeitschrift für Angewandte Mathematik und Mechanik*, 74(8) :332–332, 1994.

- [22] R Gordon. A tutorial on ART (algebraic reconstruction algorithms. *IEEE Trans. Nucl. Sci.*, 21(May) :78, 1974.
- [23] Richard Gordon, Robert Bender, and Gabor T. Herman. Algebraic Reconstruction Techniques (ART) for three-dimensional electron microscopy and X-ray photography. *Journal of Theoretical Biology*, 29(3) :471–481, 1970.
- [24] James Gregson. Applications of Inverse Problems in Fluids and Imaging by. (July), 2015.
- [25] Martin Hanke, James G. Nagy, and Curtis Vogel. Quasi-Newton approach to nonnegative image restorations. *Linear Algebra and Its Applications*, 316(1-3) :223–236, 2000.
- [26] Christian Hansen. Algebraic Methods for Computed Tomography. page 50, 2016.
- [27] Per Christian Hansen and Jakob Sauer Jørgensen. AIR Tools II : algebraic iterative reconstruction methods, improved implementation. *Numerical Algorithms*, 79(1) :107–137, 2018.
- [28] S. Kaczmarz. Angenaherte auflösung von systemen linearer gleichungen. *Acad. Pol. Sci. Lett. A*, 1937.
- [29] Dongmin Kim, Suvrit Sra, and Inderjit S. Dhillon. A non-monotonic method for large-scale non-negative least squares. *Optimization Methods and Software*, 28(5) :1012–1039, 2013.
- [30] Dongmin Kim, Suvrit Sra, and IS S Dhillon. A new projected quasi-newton approach for the nonnegative least squares problem. 2006.
- [31] S. Kullback and R. A. Leibler. On information and sufficiency. *Ann. Math. Statist.*, 22(1) :79–86, 03 1951.
- [32] Charles L. Lawson and Richard J. Hanson. *Solving least squares problems*, volume 15 of *Classics in Applied Mathematics*. Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 1995. Revised reprint of the 1974 original.
- [33] H. G. Maas, A. Gruen, and D. Papantoniou. Particle tracking velocimetry in three-dimensional flows. *Experiments in Fluids*, 15(2) :133–146, Jul 1993.
- [34] M. Moroni, J. Nogueira, M. Miozzi, G. P. Romano, A. Cenedese, P. A. Rodriguez, and A. Lecuona. Ptv for the characterization of turbulent channel flow : Comparison of experimental and simulation approaches. pages 279–293, 2004.
- [35] V. Paul Pauca, J. Piper, and Robert J. Plemmons. Nonnegative matrix factorization for spectral data analysis. *Linear Algebra and its Applications*, 416(1) :29 – 47, 2006. Special Issue devoted to the Haifa 2005 conference on matrix theory.
- [36] D. Raparia, J. Alessi, and A. Kponou. The Algebraic Reconstruction Technique (ART). pages 2023–2025, 2002.
- [37] L. Shure. Brief history of nonnegative least squares in matlab. <http://blogs.mathworks.com/loren/2006/> , .
- [38] Karsten Steinhaeuser, Nitesh V Chawla, and Auroop R Ganguly. Improving Inference of Gaussian Mixtures using Auxiliary Variables. *Statistical Analysis and Data Mining*, 8(5) :497–511, 2015.
- [39] T. Strohmer and R. Vershynin. A randomized kaczmarz algorithm for linear systems with exponential convergence. *J. Fourier Anal. Appl.*, 2009.
- [40] Conghui Tan, Shiqian Ma, Yu Hong Dai, and Yuqiu Qian. Barzilai-borwein step size for stochastic gradient descent. *Advances in Neural Information Processing Systems*, (2) :685–693, 2016.
- [41] Joel A. Tropp and Anna C. Gilbert. Signal recovery from random measurements via orthogonal matching pursuit. *IEEE Transactions on Information Theory*, 53(12) :4655–4666, 2007.
- [42] Mark H. Van Benthem and Michael R. Keenan. Fast algorithm for the solution of large-scale non-negativity-constrained least squares problems. *Journal of Chemometrics*, 18(10) :441–450, 2004.
- [43] Gary L. Viviani. *Practical Optimization.*, volume PAS-104. 1985.
- [44] Zheng Zhang, Yong Xu, Jian Yang, Xuelong Li, and David Zhang. A Survey of Sparse Representation : Algorithms and Applications. *IEEE Access*, 3 :490–530, 2015.