# Accélération algorithmique et matérielle des méthodes d'estimation de cartes d'abondances en imagerie hyperspectrale

Maxime Legendre

l'université nantes angers le mans
COMMUNAUTÉ D'UNIVERSITÉS ET ÉTABLISSEMENTS

Centrale Nantes

# Thèse de Doctorat

Maxime LEGENDRE

*Mémoire présenté en vue de l'obtention du*

***grade de Docteur de l'École centrale de Nantes***

*sous le label de l'Université de Nantes Angers Le Mans*

**École doctorale : Sciences et technologies de l'information et mathématiques**

**Discipline : Traitement du signal et des images, section CNU 61**

**Unité de recherche : Institut de Recherche en Communications et Cybernétique de Nantes (IRCCyN)**

**Soutenue le 22 avril 2015**

# Accélération algorithmique et matérielle des méthodes d'estimation de cartes d'abondances en imagerie hyperspectrale

## JURY

| | |
|---|---|
| Président : | **M. Christophe COLLET**, Professeur des universités, Télécom Physique Strasbourg |
| Rapporteurs : | **M. Jocelyn CHANUSSOT**, Professeur des universités, Institut National Polytechnique de Grenoble |
| | **M. José BERMUDEZ**, Professeur, Univesidade Federal de Santa Catarina, Florianopolis, Brésil |
| Examinateur : | **M. Nicolas GAC**, Maître de conférences, Université Paris Sud |
| Invité : | **M. Albrecht SCHMIDT**, Solar System Operation Division, ESAC, Agence Spatiale Européenne |
| Directeur de thèse : | **M. Jérôme IDIER**, Directeur de recherche CNRS, IRCCyN, Nantes |
| Co-encadrants : | **M. Saïd MOUSSAOUI**, Maître de conférences HDR, École Centrale de Nantes |
| | **M. Frédéric SCHMIDT**, Maître de conférences HDR, Université Paris Sud |

# Remerciements

En préambule, je remercie l'Agence Spatiale Européenne (ESA) et la région Pays de la Loire pour avoir financé cette thèse.

Je tiens à remercier tout particulièrement mes trois encadrants de thèse : Saïd Moussaoui avec qui j'ai eu le plaisir de travailler au quotidien et qui a su me transmettre ses connaissances autant que sa passion pour la recherche ; Frédéric Schmidt qui a suivi à distance mon travail et m'a apporté son savoir-faire dans le domaine de l'hyperspectral ; et Jérôme Idier qui a dirigé cette thèse avec le recul et l'expérience nécessaires pour me guider dans les moments importants. Toujours disponibles, ils ont su faire évoluer l'équilibre entre encadrement et autonomie pour me faire passer du rôle d'étudiant à celui de chercheur. Travailler à leurs côtés a été très enrichissant, tant du point de vue professionnel que relationnel.

Je remercie les membres du jury qui ont accepté de juger mon travail de thèse. Merci à Christophe Collet, professeur des universités à Télécom Physique Strasbourg, d'avoir présidé ce jury. Merci à Nicolas Gac, Maître de Conférences à l'université Paris-Sud. Je remercie vivement Jocelyn Chanussot, professeur des universités à l'INP Grenoble, et José Bermudez, professeur à l'UFSC de Florianopolis au Brésil, d'avoir accepté d'être les rapporteurs de mon manuscrit. Leurs remarques ont permis d'améliorer la qualité de celui-ci.

Mes remerciements vont également aux membres dynamiques de l'équipe *Analyse et Décision en Traitement du Signal et de l'Image* de l'IRCCyN, les permanents, Marie-Françoise Lucas, Sébastien Bourguignon, Éric Le Carpentier et Mathieu Lagrange, ainsi que les doctorants (cités plus loin).

Je remercie de surcroît les membres de l'IRCCyN que j'ai croisés pendant ces trois années. Je remercie particulièrement Emily Thureau pour son aide constante. Merci également au service administratif, Patricia Brière, Sylvie Julienne, Virginie Dupont, Michelle-Anne Audrain et Armelle Radigois. Je n'oublie pas le service informatique, Denis Creusot, Richard Randriatoamanana et Robert Legal. Je remercie enfin Michel Malabre, directeur de l'IRCCyN, de m'avoir accueilli dans le laboratoire où j'ai travaillé dans d'excellentes conditions.

Je voudrais également remercier tous ceux avec qui j'ai collaboré de près ou de loin pour mener à bien ce travail : Emilie Chouzenoux de l'Université Paris-Est pour sa rigueur et ses précieuses contributions théoriques ; Albrecht Schmidt de l'ESA pour son implication dans ce travail et pour son accueil à Madrid durant une semaine en 2012 ; Uwe Lammers de l'ESA pour son expertise sur la mission Gaia ; Luca Capriotti, stagiaire à l'ESA, pour le travail fourni même après la fin de son stage ; le laboratoire IDES de l'université Paris-Sud pour m'avoir accueilli une semaine en 2013 afin d'avancer sur les aspects pratiques de la thèse concernant la planétologie ; Patrick Launeau et Antoine Ba de l'Université de Nantes pour m'avoir ramené sur Terre en appliquant mon travail dans un contexte local.

Je remercie l'équipe pédagogique du département d'enseignement *Automatique et Robotique*

de l'Ecole Centrale de Nantes qui m'a permis de m'initier à l'enseignement dans un cadre privilégié et qui m'a donné le goût de transmettre mes connaissances.

J'adresse mes vifs remerciements aux doctorants et post-doctorants qui m'ont accompagné pendant ces trois années : Ewen, Vincent, Corentin, Grégoire, Philip, Emmanuel, Bogdan, Joan, Ablamvi, Hendry, Marija, Natalia, Oscar, Eduardo, Fang, Jonathan, Inès, Denis, Xavier, Céline, Ilja, ... Leur présence à mes côtés, au travail comme en dehors, a fait de cette thèse un moment riche en rencontres, en joie et en bonne humeur. Dans cette liste, une place toute particulière revient à Aleksandra pour l'amour que je lui porte et qu'elle me rend à merveille.

Je remercie enfin ma famille pour m'avoir toujours soutenu dans mes choix de carrière, pour me ramener parfois les pieds sur terre, et pour les efforts consentis pour tenter d'expliquer mon travail dans son entourage.

# Table des matières

# Liste des tableaux

# Table des figures

# Chapitre 1

# Introduction générale

## Sommaire



## 1.1 Imagerie hyperspectrale

### 1.1.1 De l'image classique à l'image hyperspectrale

Un appareil photographique classique échantillonne une scène spatialement sous forme de pixels et spectralement sous forme de trois bandes spectrales couvrant la partie visible du spectre lumineux : rouge, vert et bleu entre 0.39 $\mu m$ et 0.78 $\mu m$. Il réalise ainsi trois mesures de luminosité par pixel. Le résultat de cette procédure est *une image trichromatique*. Ce choix est adapté à la reproduction d'une scène destinée à être vue par un œil humain car celui-ci possède sur sa rétine trois types de cônes permettant de percevoir les trois couleurs primaires mesurées par l'appareil photographique. On peut ainsi reproduire fidèlement toutes les couleurs perceptibles par l'œil humain et seuls l'échantillonnage spatial et la quantification numérique limitent la qualité de l'image. Mais la lumière provenant de cette scène contient bien plus d'information que ces trois valeurs par pixel. En effet, l'énergie lumineuse réfléchie par une surface est une grandeur continue spatialement et spectralement. Tout échantillonnage de cette grandeur conduit à une perte d'information. Un bon échantillonnage est celui qui conserve un maximum d'information utile en vue d'une exploitation donnée. Si l'échantillonnage de l'image trichromatique est adapté à sa fonction, d'autres types d'échantillonnage, en particulier suivant la dimension spectrale, permettent d'autres types d'applications.

| Bande spectrale | Longueur d'onde | Résolution spatiale |
|---|---|---|
| Bande 1 | 0.433 - 0.453 $\mu$m | 30 m |
| Bande 2 | 0.450 - 0.515 $\mu$m | 30 m |
| Bande 3 | 0.525 - 0.600 $\mu$m | 30 m |
| Bande 4 | 0.630 - 0.680 $\mu$m | 30 m |
| Bande 5 | 0.845 - 0.885 $\mu$m | 30 m |
| Bande 6 | 1.560 - 1.660 $\mu$m | 30 m |
| Bande 7 | 2.100 - 2.300 $\mu$m | 30 m |
| Bande 8 | 0.500 - 0.680 $\mu$m | 15 m |
| Bande 9 | 1.360 - 1.390 $\mu$m | 30 m |
| Bande 10 | 10.30 - 11.30 $\mu$m | 100 m |
| Bande 11 | 11.50 - 12.50 $\mu$m | 100 m |

TABLE 1.1 – Caractéristiques des images multispectrales issus du satellite *Landsat 8*.

*L'imagerie multispectrale* [Schott, 2007; Richards, 1999] propose un échantillonnage différent de l'imagerie trichromatique dont le but est l'identification des constituants de la surface observée. En effet, en un point donné, la forme du spectre de la lumière reflétée est déterminée par la composition chimique de la surface car chaque liaison chimique entre les molécules constituant la surface absorbe une longueur d'onde spécifique de la lumière incidente. Ainsi, bien plus que la couleur, c'est la composition chimique de la surface qui est accessible à partir de la lumière reflétée, chaque matériau étant caractérisé par un spectre de réflectance spécifique appelé *signature spectrale*. C'est pour exploiter cette information qu'est apparue l'imagerie multispectrale dans les années 1970. Celle-ci consiste à mesurer la luminance dans un nombre donné de bandes spectrales (ou canaux spectraux) choisies de façon à discriminer les différents matériaux recherchés. Ces bandes peuvent être séparées, contiguës ou superposées, et éventuellement en dehors du domaine visible. Cette technique permet l'identification des matériaux à partir du profil global de leurs spectres lumineux. Le tableau 1.1 détaille les 11 bandes spectrales acquises par le satellite d'observation terrestre *Landsat 8* lancé le 11 février 2013 [Knight et Kvaran, 2014]. Dans cet exemple, l'échantillonnage spatial varie également en fonction des bandes spectrales.

*L'imagerie hyperspectrale* [Chang, 2007] est une évolution de l'imagerie multispectrale qui est caractérisée par l'acquisition de la lumière de la scène observée dans un grand nombre de bandes spectrales contiguës. Avec plusieurs dizaines ou centaines de bandes couvrant généralement le visible et l'infrarouge, la résolution spectrale est nettement supérieure à celle de l'imagerie multispectrale, souvent au détriment de la résolution spatiale. Cela permet une analyse plus fine de la composition de la surface, laissant apparaître les bandes d'absorption caractéristiques de chaque matériau. Une image hyperspectrale est donc composée d'un ensemble de mesures d'énergie lumineuse suivant trois dimensions (deux dimensions spatiales et une dimension spectrale). Elle est parfois représentée sous la forme d'un cube de données comme dans la figure 1.1. Une image hyperspectrale peut occuper un volume de plusieurs centaines de méga-octet (Mo) en mémoire. Tout traitement est rendu difficile par le volume important des données en jeu. **La mise en œuvre de solutions algorithmiques et matérielles permettant le contournement de cette difficulté est l'un des objectifs de ce travail de thèse.**

La figure 1.2 illustre les différents échantillonnages d'un spectre lumineux correspondant à un pixel d'une image. Avec l'acquisition trichromatique, il est seulement possible de donner une couleur au pixel correspondant à ce spectre : un vert foncé. Avec l'acquisition multispectrale, on pourra par exemple identifier qu'il s'agit de végétation. Avec l'acquisition hyperspectrale, on pourra identifier l'espèce végétale observée, ici un pin à pignons dont le spectre de réflec-

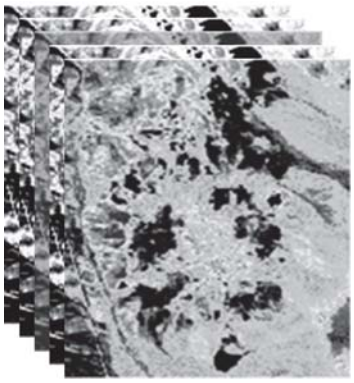


FIGURE 1.1 – Image hyperspectrale représentée par un cube de données.

tance est disponible à l'adresse http://speclab.cr.usgs.gov/spectral.lib06/ds231/datatable.html.

### 1.1.2 Acquisition d'une image hyperspectrale

Il convient ici de préciser ce que l'on appelle *réflectance* d'une surface. Pour une longueur d'onde donnée, la réflectance correspond à la proportion de lumière incidente réfléchie par la surface. Plus précisément, la réflectance en tant que BRDF (*Bidirectional Reflectance Distribution Function*) est calculée comme le rapport entre la radiance $I$ émergeant d'un élément de surface dans un certain angle solide, exprimée en $\mathtt{W} \cdot \mathtt{m}^{-2} \cdot \mathtt{sr}^{-1}$, et l'intensité $I'$ du flux incident collimaté (c'est à dire venant d'une seule direction), exprimée en $\mathtt{W} \cdot \mathtt{m}^{-2}$. La réflectance BRDF est donc exprimée en $\mathtt{sr}^{-1}$ et est obtenue par la formule [Schmidt, 2014]

$$\rho(\theta) = \frac{I}{\pi I' \cos(\theta)}, \tag{1.1}$$

$\theta$ étant l'angle d'incidence. Dans cette thèse, la BRDF est supposée constante quelque soit la direction émergente choisie (hypothèse lambertienne). Son intégration sur la demi-sphère des émergences donne une réfléctance globale sans unité et comprise entre 0 et 1. Durant toute cette thèse, le terme *réflectance* désignera cette réflectance globale.

Une image hyperspectrale est acquise à l'aide d'un spectro-imageur dont le schéma de fonctionnement est donné par la figure 1.3. Il est généralement embarqué sur un satellite ou aéroporté pour acquérir des images de zones étendues, mais il peut aussi être utilisé en laboratoire où les conditions de mesure sont maitrisées. Il effectue un balayage de la surface et mesure la quantité d'énergie lumineuse issue de chaque élément de surface couvert par un pixel pour chaque bande spectrale. Il ne mesure pas directement la réflectance d'un élément de surface, mais la quantité de radiation enregistrée par la surface du détecteur (en W). Après une calibration spectrale et radiométrique, les mesures sont converties en *radiance*, exprimée en $\mathtt{W} \cdot \mathtt{m}^{-2} \cdot \mathtt{sr}^{-1}$. La réflectance est caractéristique d'une surface, alors que la radiance dépend des conditions de mesure telles que :

- Le spectre de la lumière incidente.
- L'interaction de la lumière avec l'atmosphère.
- L'angle d'incidence et l'angle d'émergence.

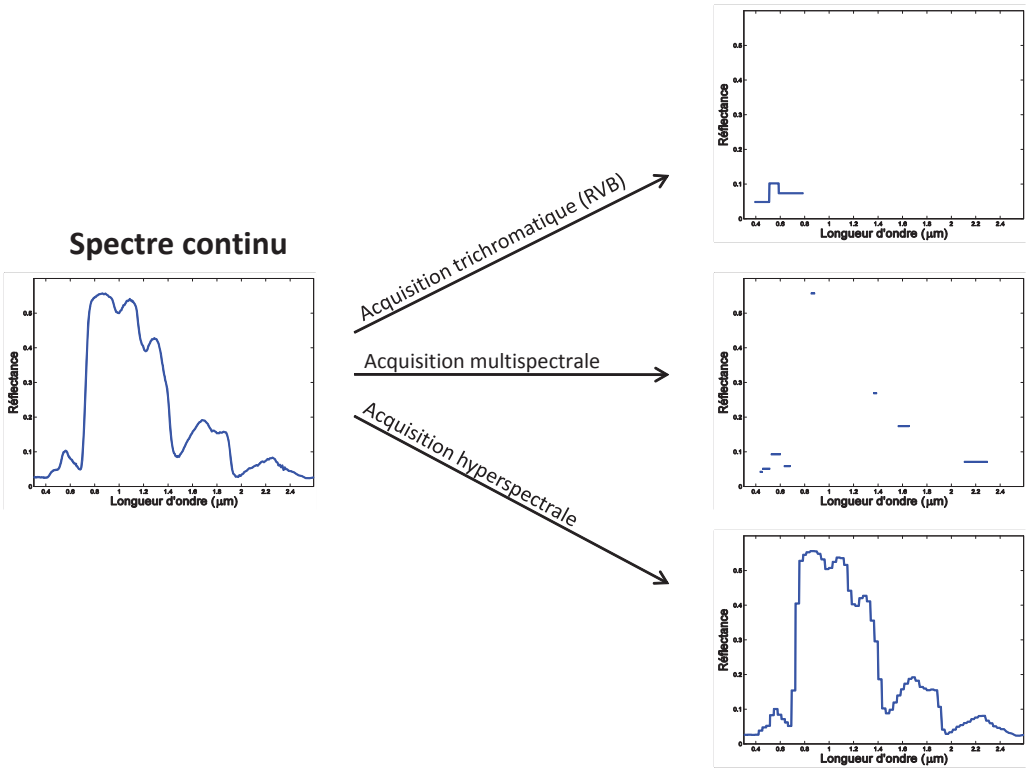



FIGURE 1.2 – Différents modes d'acquisition d'un spectre lumineux : trichromatique, multispectral et hyperspectral.

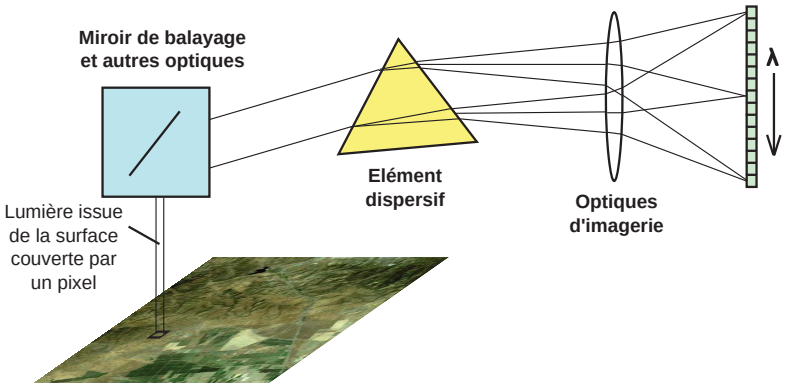



FIGURE 1.3 – Diagramme simplifié des éléments composant un spectro-imageur. (figure extraite de [Microimages, 2012])

| Nom | AVIRIS | Hyperion | OMEGA |
|---|---|---|---|
| Embarquation | Aéroporté | Satellite EO-1 | Sonde Mars Express |
| Organisme | JPL, NASA | NASA | ESA |
| Gamme spectrale | 0.4 - 2.5 $\mu$m | 0.4 - 2.5 $\mu$m | 0.35 - 5.1 $\mu$m |
| Nombre de bandes | 224 | 242 | 352 |
| Résolution spectrale | 10 nm | 10 nm | 7 à 20 nm |
| Résolution spatiale | 20 m | 30 m | 300 m à 4.8 km |
| | (à 20 km d'altitude) | | (orbite élliptique) |

TABLE 1.2 – Caractéristiques de quelques spectro-imageurs actuels.

Toutes ces influences doivent être prises en compte afin de convertir chaque spectre de radiance mesuré en un spectre de réflectance permettant de caractériser la composition de la surface. Toutes les images hyperspectrales utilisées dans cette thèse sont exprimées en réflectance. Cela suppose que la radiance mesurée a été préalablement convertie en réflectance en prenant en compte un maximum de paramètres [Green *et al.*, 1993]. Le tableau 1.2 donne les spécifications de quelques spectro-imageurs actuellement utilisés pour des mission d'observation de la Terre et de Mars.

### 1.1.3 Applications

Les domaines pouvant utiliser l'imagerie hyperspectrale sont nombreux. On peut citer :

- L'agriculture : suivi du développement et de la croissance des cultures, détection de maladies [Lacar *et al.*, 2001; Tilling *et al.*, 2006].
- La géologie : identification et classification des roches en laboratoire ou sur des surfaces étendues [Van der Meer *et al.*, 2012].
- L'activité militaire : cartographie, détection et suivi de cibles [Ardouin *et al.*, 2007].
- L'astrophysique et la planétologie : caractérisation chimique d'astres lointains, cartographie géologique d'astres plus proches [Starck et Murtagh, 2007; Schmidt *et al.*, 2011].

### 1.1.4 Traitements possibles

Une image hyperspectrale peut être exploitée de différentes façons en fonction des hypothèses faites sur la composition de la surface observée et de l'information que l'on souhaite obtenir. On peut distinguer 4 types de traitement usuels :

- **Détection de cible** [Manolakis *et al.*, 2003; Nasrabadi, 2014] : Il s'agit d'identifier les pixels dont la surface est constituée entièrement ou en partie d'un matériau cible dont le spectre de réflectance est connu. Ce type de traitment fait appel à des outils de mesure de distance entre deux spectres comme la *distance euclidienne* ou l'*angle spectral* ainsi qu'à des méthodes de seuillage pour discriminer les pixels contenant le matériau cible. Des méthodes statistiques peuvent être employées lorsque le spectre cible est connu à travers une description probabiliste.

- **Classification** [Schmidt, 2007; Fauvel *et al.*, 2008] : Il s'agit d'associer à chaque pixel un matériau. Ceux-ci peuvent être issus d'une liste préétablie de matériaux dont les spectres sont connus (*classification supervisée*) ou bien issus des spectres mesurés dans l'image (*classification non-supervisée*). Ces méthodes de traitement font également appel aux outils de mesure de distance spectrale ainsi qu'aux méthodes classiques de classification telles que les SVM (*Support Vector Machine*, en français *Machines à Vecteurs de Support*) ou les réseaux de neurones. Certaines méthodes dites de classification *spatiale-spectrale* exploitent un *a priori* d'homogénéité spatiale de l'image pour améliorer la qualité de leurs résultats.

- **Déconvolution** [Bourguignon *et al.*, 2012; Henrot, 2013] : Comme toute image issue d'un instrument de mesure optique, une image hyperspectrale est altérée par les défauts de l'appareil et certaines conditions extérieurs comme les perturbations atmosphériques. Ces phénomènes se traduisent dans l'image par un flou à la fois spatial et spectral caractérisé par sa PSF (*Point Spread Function*, en français *Fonction d'Étalement du Point*). La déconvolution est une procédure de restauration d'image permettant de compenser les effets de la PSF. Il s'agit d'une étape de prétraitement effectuée en amont des autres traitements décrits dans cette section lorsque ces effets ne peuvent pas être négligés.

- **Démixage spectral** [Keshava et Mustard, 2002; Bioucas-Dias *et al.*, 2012] : Ce type de traitement repose sur l'hypothèse que chaque élément de surface couvert par un pixel peut être constitué de plusieurs matériaux. Le démixage spectral consiste à identifier ces matériaux ainsi que leurs proportions dans chaque pixel à partir des spectres mesurés qui sont des mélanges des spectres des matériaux purs. Il s'agit donc d'un problème de séparation de sources. Cette thèse se place dans le cadre du démixage spectral, et plus particulièrement de l'estimation de cartes d'abondances. Ce problème sera détaillé dans le chapitre 2.

L'ensemble de ces traitements font partie d'une classe plus large de problèmes appelés *problèmes inverses*.

### 1.1.5 Problèmes inverses

On parle de problème inverse [Idier, 2013; Tarantola, 2005] lorsque l'on cherche à déterminer les causes d'un phénomène à partir des observations expérimentales de ses effets. En sismologie, la localisation de l'origine d'un tremblement de terre à partir de mesures faites par plusieurs stations sismiques est un exemple classique de problème inverse. En traitement d'image, un

exemple est celui de la restauration d'image lorsqu'il s'agit de retrouver une image originelle à partir d'une version dégradée par un bruit, un flou, ou un sous-échantillonnage [Bertero et Boccacci, 1998; Jansson, 2014]. On parle de reconstruction d'image lorsqu'il s'agit d'estimer une image originelle à partir de données liées à cette image, par exemple par une opération de projection [Gordon et Herman, 1971; Zeng, 2001].

La résolution d'un problème inverse passe par une étape de modélisation du phénomène physique qui décrit comment les grandeurs recherchées se traduisent en effets observables expérimentalement. Il s'agit du modèle direct. La démarche consiste alors à exploiter ce modèle physique afin d'estimer au mieux les inconnues recherchées. La résolution mathématique est rendue difficile par le fait que les problèmes inverses sont en général des problèmes mal posés, c'est-à-dire que les mêmes effets observés peuvent s'expliquer par des causes différentes. Un problème inverse peut donc avoir plusieurs solutions. Il est alors nécessaire d'ajouter des contraintes ou des *a priori* permettant l'obtention d'une solution unique.

Les travaux présentés dans cette thèse rentrent dans le cadre des problèmes inverses linéaires, c'est-à-dire que l'information d'intérêt est liée aux données mesurées par une relation linéaire. Ces développements sont motivés par un problème de resonstruction rencontré en imagerie hyperspectrale : le démixage spectral, et plus partculièrement l'estimation de cartes d'abondances. Le modèle direct retenu et plusieurs méthodes d'inversions sont présentés dans le chapitre 2.

## 1.2 Contributions de la thèse et publications

Cette thèse apporte au domaine de l'imagerie hyperspectrale des contributions à la fois pratiques et théoriques.

Tout d'abord d'un point de vue pratique, le problème de l'estimation de cartes d'abondances est étudié en détail. Quatre méthodes de résolutions existantes sont décrites : NNLS (*Non-Negative Least Squares*), FCLS (*Fully-Constrained Least Squares*), ADMM (*Alternating Direction Method of Multipliers*) et IPLS (*Interior-Points Least Squares*). Leurs performances sont évaluées de la façon la plus exhaustive possible dans une étude comparative. Leurs comportements, et en particulier leurs temps de calcul, sont analysés en fonction de chaque paramètre du modèle afin de transmettre au lecteur la vision la plus juste et complète possible de chaque méthode. L'une d'entre elles, la méthode primale-duale de points intérieurs (IPLS), est analysée plus en détail et des améliorations sont proposées, tant au niveau de l'algorithme que de son implémentation. Tout au long de ce travail, un effort particulier est porté sur la facilité de parallélisation de l'algorithme proposé afin de traiter de grands volumes de données. Il en résulte une nette accélération de cette méthode sur CPU et des calculs adaptés à une implémentation sur GPU. Cette implémentation est réalisée puis utilisée pour traiter un grand nombre d'images de grande taille. La méthode développée est testée en conditions réelles à travers deux collaborations : l'une dans le domaine de la planétologie et l'autre dans le domaine de l'environnement.

Ensuite d'un point de vue théorique, les améliorations apportées à la méthode IPLS dépassent le cadre de l'imagerie hyperspectrale. Plusieurs méthodes d'optimisation sont combinées pour former une proposition nouvelle dans le domaine de l'optimisation convexe sous contraintes. Ainsi une approche par majoration-minimisation quadratique et une approche par gradient conjugué préconditionné sont incluses dans la méthode de points intérieurs pour en accélérer la vitesse de convergence. La preuve de convergence de la méthode développée est établie.

Les travaux réalisés au cours de cette thèse ont fait l'objet de publications parues dans des revues nationales et internationales et de communications dans des congrès nationaux et internationaux.

**Articles dans des revues nationales ou internationales**

[A.1] E. Chouzenoux, S. Moussaoui, **M. Legendre** et J. Idier, « Algorithme primal-dual de points intérieurs pour l'estimation pénalisée des cartes d'abondances en imagerie hyperspectrale », *Traitement du Signal*, vol. 30, no.1-2, pp.35-59, 2013.

[A.2] **M. Legendre**, A. Schmidt, S. Moussaoui et U. Lammers, « Solving Systems of Linear Equations by GPU-based Matrix Factorization in a Science Ground Segment », *Astronomy and Computing*, vol. 3-4, pp. 58-64, 2013.

[A.3] E. Chouzenoux, **M. Legendre**, S. Moussaoui et J. Idier, « Fast constrained least squares spectral unmixing using primal-dual interior point optimization », *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. 7, no. 1, pp. 59-69, 2014

[A.4] F. Schmidt, **M. Legendre** et S. Le Mouëlic, « Minerals detection for hyperspectral images using adapted linear unmixing : Linmin », *Icarus*, vol.237, pp. 61-74, 2014

**Communications avec actes dans un congrès national ou international**

[C.1] **M. Legendre**, L. Capriotti, F. Schmidt, S. Moussaoui et A. Schmidt, « GPU Implementation issues for fast unmixing of hyperspectral images », dans *EGU General Assembly Conference Abstracts*, Vienne, Autriche, Avril 2013

[C.2] **M. Legendre**, S. Moussaoui, F. Schmidt et J. Idier, « Parallel implementation of a primal-dual interior-point optimization method for fast abundance maps estimation », dans *Proc. of IEEE International Workshop on Hyperspectral Image and Signal Processing (WHISPERS)*, Gainesville, États-Unis, Juin 2013.

[C.3] **M. Legendre**, S. Moussaoui, F. Schmidt et J. Idier, « Implémentation parallèle d'une méthode d'estimation des cartes d'abondances en imagerie hyperspectrale », dans *Actes du 24ème Colloque GRETSI*, Brest, France, Septembre 2013.

[C.4] **M. Legendre**, S. Moussaoui, E . Chouzenoux et J. Idier, « Primal-dual interior-point optimization based on majorization-minimization for edge-preserving spectral unmixing », à paraître dans *Proc. of IEEE International Conference on Image Processing (ICIP)*, Paris, France, Octobre 2014.

## 1.3 Organisation du document

Le chapitre 2 est consacré à la présentation du problème de l'estimation de cartes d'abondances. Toutes les hypothèses sont données pour transformer ce problème physique en un problème d'optimisation. Quatre méthodes de résolution de ce problème sont détaillées : NNLS, FCLS, ADMM et IPLS.

Le chapitre 3 reprend les quatre méthodes décrites au chapitre 2 et propose une analyse comparative à l'aide de données simulées numériquement. L'influence de chaque paramètre du modèle sur les performances des différentes méthodes est étudiée.

Le chapitre 4 se focalise sur la méthode IPLS et propose plusieurs améliorations visant à réduire le temps de calcul tout en produisant un algorithme facilement parallélisable. Pour cela l'étape de l'algorithme la plus coûteuse en temps de calcul est remplacée par une méthode itérative basée sur une approximation majorante séparable permettant à la fois une accélération et l'apparition d'une structure adaptée au calcul parallèle.

Le chapitre 5 donne les détails de l'implémentation GPU de la méthode développée au chapitre 4. L'architecture matérielle du GPU est étudiée et exploitée afin d'utiliser le maximum des capacités de cet outil de calcul intensif.

Le chapitre 6 développe deux applications réalisées à travers deux collaborations. La première application concerne le traitement sur GPU d'un grand nombre d'images hyperspectrales de grande taille issues de la mission d'exploration planétaire Mars Express contribuant à la connaissance de la géologie de la surface de Mars. La deuxième concerne l'identification des espèces végétales présentes sur la côte atlantique française à partir d'une campagne de mesure aéroportée afin de surveiller et de préserver les dunes côtières grâce à la végétation.

# Chapitre 2

# Estimation des cartes d'abondances en imagerie hyperspectrale

## Sommaire



Ce chapitre présente le problème d'estimation des cartes d'abondances en imagerie hyperspectrale. La formulation du modèle de mélange conduit à poser l'estimation des cartes sous la forme d'un problème d'optimisation convexe sous contraintes. Plusieurs méthodes de résolution sont détaillées.

## 2.1 Séparation de sources en imagerie hyperspectrale

Comme nous l'avons vu dans le chapitre 1, l'imagerie hyperspectrale consiste en la collecte et l'exploitation du spectre de réflectance de la lumière renvoyée par chaque zone d'une surface associée à un pixel, mesuré dans plusieurs bandes spectrales contiguës. Cette technique permet d'accéder à des informations qualitatives et quantitatives liées à la composition de la surface observée. On suppose maintenant que chaque pixel d'une image hyperspectrale correspond à une surface composée de plusieurs matériaux caractérisés par leurs spectres de réflectance, le

spectre observé est donc issu du mélange des spectres purs de chaque constituant. Le démélange spectral consiste à identifier ces constituants ainsi qu'à déterminer leurs proportions, appelées *abondances*, dans chaque pixel.

Certains travaux proposent d'estimer conjointement ces deux entités, soit par des méthodes de séparation de sources non-négatives [Moussaoui *et al.*, 2008; Dobigeon *et al.*, 2009], soit par factorisation de matrices non-négatives [Jia et Qian, 2009; Huck *et al.*, 2010]. D'autres préconisent une estimation séquentielle des spectres purs et de leurs abondances [Keshava et Mustard, 2002; Chang, 2007], c'est le cadre dans lequel se place cette thèse.

La première étape de ces méthodes consiste à déterminer les pôles du mélange, appelés *endmembers*. Ces spectres peuvent être des pixels de l'image considérés comme purs, sélectionnés par une méthode méthode d'extraction des endmembers telle que VCA [Nascimento et Dias, 2005] ou NFINDR [Winter, 1999; Plaza et Chang, 2005]. Une étude sur les méthodes d'extraction des endmembers est fournie dans [Bioucas-Dias et Plaza, 2011]. Le principe de ces méthodes est de trouver les pixels purs, c'est-à-dire les pixels dont les spectres ne peuvent pas être exprimés comme des combinaisons linéaires des autres. Les endmembers peuvent aussi être sélectionnés dans une bibliothèque spectrale, c'est-à-dire une liste de spectres de matériaux connus mesurés en laboratoire. Dans cette thèse, nous utilisons à titre d'illustration la bibliothèque spectrale USGS (United States Geological Survey) [Clark *et al.*, 1993].

La deuxième étape consiste à estimer les proportions de chacun des spectres purs dans chacun des pixels de l'image. Cette étape, aussi appelée estimation des cartes d'abondances, fait l'objet de cette thèse. Elle peut être vue comme un problème de séparation de sources supervisé, chaque spectre pur étant une source connue à l'avance, le spectre de chaque pixel est un mélange de ces sources selon des proportions que l'on cherche à connaître.

## 2.2 Hypothèses de travail

### 2.2.1 Modèle de mélange linéaire

L'observation de plusieurs matériaux au sein d'un même pixel peut avoir deux origines qui sont illustrées sur la figure 2.1. Premièrement, les matériaux peuvent être séparés spatialement mais avoir une surface inférieure à celle couverte par un pixel. Dans ce cas, chaque rayon lumineux provenant de la source n'interagit qu'avec un seul matériau de la surface observée. Le mélange est seulement dû à une trop faible résolution spatiale de l'instrument de mesure. Le spectre mesuré est alors un mélange linéaire des spectres purs des matériaux de surface, les coefficients de mélange représentant les proportions surfaciques de chaque constituant. Deuxièmement, la surface observée peut être composée d'une seule entité qui est un mélange intime de plusieurs matériaux, comme par exemple un sable dont les grains ont diverses compositions chimiques. Les rayons lumineux peuvent alors interagir avec plusieurs matériaux avant d'être renvoyés vers l'instrument de mesure. Ces réflexions multiples conduisent à l'observation d'un spectre qui est un mélange non linéaire des spectres purs des constituants en présence.

Dans cette thèse, nous utilisons le modèle de mélange linéaire, qui correspond à l'observation de matériaux spatialement séparés, mais qui peut être vu comme une approximation au premier ordre du modèle de transfert radiatif dans le cas d'un mélange intime [Keshava et Mustard, 2002]. Ainsi, nous supposons que le spectre observé en un pixel donné est une combinaison

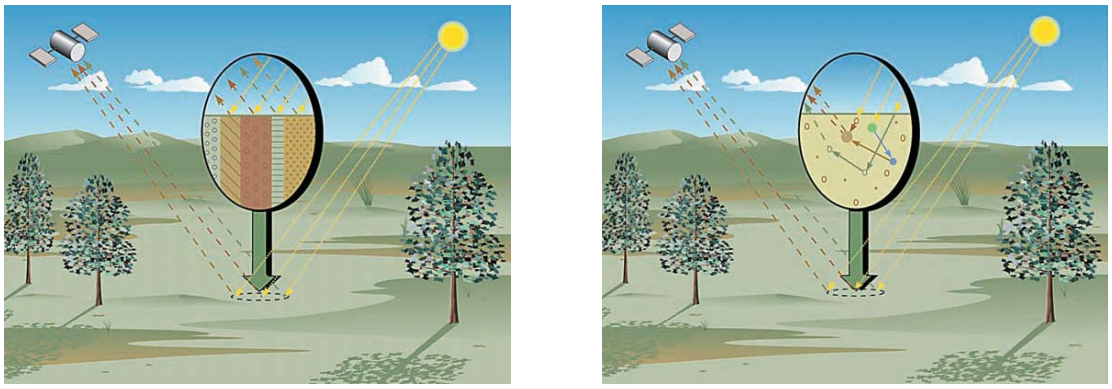


FIGURE 2.1 – Illustration d'un mélange linéaire (à gauche) où chaque rayon incident n'interagit qu'avec un matériau de surface, et d'un mélange non-linéaire (à droite) où un rayon incident peut intéragir avec plusieurs matériaux. (figure extraite de [Keshava et Mustard, 2002])

linéaire des spectres purs, pondérés par des coefficients de mélange, appelés abondances, correspondant aux proportions recherchées.

En représentant le spectre du n-ième pixel d'une image sous la forme d'un vecteur $\boldsymbol{y}_n \in \mathbb{R}^L$, $L$ étant le nombre de bandes spectrales acquises, le modèle de mélange linéaire permet d'exprimer le spectre observé en fonction des spectres purs des composants et de leurs abondances à l'aide d'une simple équation linéaire :

$$\boldsymbol{y}_n = \boldsymbol{S}\boldsymbol{a}_n + \boldsymbol{e}_n, \tag{2.1}$$

où $\boldsymbol{S} \in \mathbb{R}^{L\times P}$ est la matrice des spectres purs contenant les valeurs de réflectance de $P$ composants dans $L$ bandes spectrales, $\boldsymbol{a}_n \in \mathbb{R}^P$ est le vecteur des abondances contenant les proportions de chacun des $P$ composants, et $\boldsymbol{e}_n \in \mathbb{R}^L$ représente le bruit de mesure supposé gaussien, centré, et i.i.d. (indépendant et identiquement distribué).

Une image complète contenant $N$ pixels peut être représentée par une matrice $\boldsymbol{Y} = [\boldsymbol{y}_1 \dots \boldsymbol{y}_N] \in \mathbb{R}^{L\times N}$. Le modèle de mélange linéaire s'écrit alors

$$\boldsymbol{Y} = \boldsymbol{S}\boldsymbol{A} + \boldsymbol{E}, \tag{2.2}$$

avec $\boldsymbol{A} = [\boldsymbol{a}_1 \dots \boldsymbol{a}_N] \in \mathbb{R}^{P\times N}$ et $\boldsymbol{E} = [\boldsymbol{e}_1 \dots \boldsymbol{e}_N] \in \mathbb{R}^{L\times N}$.

Le problème consiste alors à estimer la matrice $\boldsymbol{A}$ qui minimise l'erreur entre l'observation et le modèle, au sens des moindres carrés :

$$\underset{\boldsymbol{A}\in\mathbb{R}^{P\times N}}{\text{minimiser}} \ \|\boldsymbol{Y} - \boldsymbol{S}\boldsymbol{A}\|_F^2 = \sum_{n=1}^{N} \|\boldsymbol{y}_n - \boldsymbol{S}\boldsymbol{a}_n\|_2^2 , \tag{2.3}$$

où $\|\cdot\|_F$ représente la norme de Frobenius.

### 2.2.2 Contraintes

Le problème de moindres carrés non contraint (2.3) possède une solution analytique simple :

$$\hat{\boldsymbol{A}} = \boldsymbol{S}^{+}\boldsymbol{Y}, \tag{2.4}$$

où $\boldsymbol{S}^{+}$ est la pseudo-inverse à gauche de $\boldsymbol{S}$ définie par $\boldsymbol{S}^{+} = (\boldsymbol{S}^{\mathrm{t}}\boldsymbol{S})^{-1}\boldsymbol{S}^{\mathrm{t}}$. Cependant cette solution n'est pas satisfaisante car elle ne prend pas en compte les contraintes liées à la physique

du problème : les valeurs de $\boldsymbol{A}$ étant des coefficients de mélange, elles doivent satisfaire les contraintes de non-négativité :

$$A_{p,n} \geqslant 0 \ (\forall n = 1, \ldots, N, \ \forall p = 1, \ldots, P), \tag{2.5}$$

et éventuellement d'additivité :

$$\sum_{p=1}^{P} A_{p,n} = 1 \ (\forall n = 1, ..., N). \tag{2.6}$$

Nous noterons la contrainte de non-négativité *NN* (*Non-Negativity*), et la contrainte d'additivité *STO* (*Sum To One*). Il existe une alternative à la contrainte STO, que nous noterons *SLO* (*Sum Less than One*) :

$$\sum_{p=1}^{P} A_{p,n} \leqslant 1 \ (\forall n = 1, ..., N). \tag{2.7}$$

L'utilisation de cette contrainte, que nous avons rarement rencontré dans la littérature [Keshava et Mustard, 2002], est motivée par le fait qu'une image hyperspectrale peut contenir en réalité plus de constituants que le nombre de spectres purs contenus dans la matrice $\boldsymbol{S}$. Dans une telle situation, l'expérience rapportée dans l'annexe A montre que la contrainte SLO conduit à de meilleurs résultats que les contraintes NN+STO ou NN seule.

Ainsi, le problème (2.3) sera toujours soumis aux contraintes linéaires d'égalité ou d'inégalité qui pourront prendre 3 formes :

- NN : $\boldsymbol{A} \geqslant 0$,
- NN et STO : $\boldsymbol{A} \geqslant 0$ et $\mathbf{1}_P^{\mathrm{t}} \boldsymbol{A} = \mathbf{1}_N^{\mathrm{t}}$,
- NN et SLO : $\boldsymbol{A} \geqslant 0$ et $\mathbf{1}_P^{\mathrm{t}} \boldsymbol{A} \leqslant \mathbf{1}_N^{\mathrm{t}}$.

La solution du problème contraint ne peut pas être exprimée sous forme analytique. En revanche elle peut être obtenue par des méthodes itératives présentées dans la section 2.4.

### 2.2.3 Régularisation spatiale

Quand bien même la solution donnée par l'équation (2.4) respecterait les contraintes NN et STO, ou NN et SLO, un autre problème survient si un bruit de mesure trop important est présent. Il s'agit de l'instabilité de la solution par rapport aux données.

En effet, une faible perturbation additive sur l'observation d'une image $\delta\boldsymbol{Y}$ entraîne une perturbation sur la solution $\delta\hat{\boldsymbol{A}} = \boldsymbol{S}^{+}\delta\boldsymbol{Y}$. Une mesure de l'instabilité de la solution peut être donnée par l'erreur relative de la solution, rapportée à l'erreur relative des données :

$$\frac{\left\|\delta\hat{\boldsymbol{A}}\right\|_F / \left\|\hat{\boldsymbol{A}}\right\|_F}{\|\delta\boldsymbol{Y}\|_F / \|\boldsymbol{Y}\|_F}. \tag{2.8}$$

Or, cette erreur relative est bornée par

$$\frac{\left\|\delta\hat{\boldsymbol{A}}\right\|_F / \left\|\hat{\boldsymbol{A}}\right\|_F}{\|\delta\boldsymbol{Y}\|_F / \|\boldsymbol{Y}\|_F} \leqslant \kappa(\boldsymbol{S}), \tag{2.9}$$

$\kappa(\boldsymbol{S})$ étant le conditionnement de la matrice $\boldsymbol{S}$ défini par

$$\kappa(\boldsymbol{S}) = \frac{\sigma_{max}}{\sigma_{min}}, \tag{2.10}$$

où $\sigma_{max}$ et $\sigma_{min}$ sont respectivement la valeur singulière maximale et minimale de $\boldsymbol{S}$.

La valeur de $\kappa(\boldsymbol{S})$ détermine ainsi la sensibilité de la solution du problème, calculée par l'équation (2.4), par rapport aux faibles variations des données. Ceci peut être interprété comme la robustesse de la solution vis-à-vis du bruit de mesure. Si $\kappa(\boldsymbol{S})$ a une valeur proche de 1, le niveau de bruit de la solution est équivalent à celui des données. Si $\kappa(\boldsymbol{S})$ est grand devant l'unité, alors le bruit de mesure est amplifié dans la solution obtenu. On parle alors de problème mal conditionné. C'est le cas lorsque plusieurs spectres purs de la matrice $\boldsymbol{S}$ sont fortement corrélés mutuellement. Le cas limite, lorsque $\sigma_{min} = 0$, correspond à une matrice $\boldsymbol{S}^{\mathrm{t}}\boldsymbol{S}$ singulière, et donc la solution ne peut plus être calculée par l'équation (2.4). En pratique, cela arrive lorsque l'un des spectres purs est une combinaison linéaire des autres. Nous rencontrerons un tel cas dans le chapitre 6. On parle alors de problème mal posé au sens de Hadamard [Hadamard, 1902] car l'unicité de la solution n'est pas assurée.

Que le problème soit mal conditionné ou mal posé, une solution consiste à le régulariser, c'est-à-dire utiliser des connaissances *a priori* sur la solution pour la contraindre d'avantage et éviter ce phénomène d'instabilité.

Ici, nous souhaitons utiliser une information *a priori* sur la régularité spatiale des abondances. En effet, sur une image hyperspectrale on s'attend à ce que deux pixels voisins aient des compositions proches. Un *a priori* plus réaliste consiste à supposer que l'image est homogène par morceaux, c'est-à-dire composée de régions au sein desquelles la composition varie peu entre pixels voisins séparées par des frontières nettes. Pour introduire cet *a priori* dans le problème, nous ajoutons un terme de pénalité au critère des moindres carrés. Le problème prend alors la forme

$$\begin{gathered}\underset{\boldsymbol{A}\in\mathbb{R}^{P\times N}}{\text{minimiser}} \ \|\boldsymbol{Y} - \boldsymbol{S}\boldsymbol{A}\|_F^2 + \beta R(\boldsymbol{A}), \\ \text{sous les contraintes } NN \text{ ou } NN + STO \text{ ou } NN + SLO,\end{gathered} \tag{2.11}$$

où $\beta \in \mathbb{R}$ le paramètre de régularisation et $R(\cdot) : \mathbb{R}^{P\times N} \to \mathbb{R}$ est la fonctionnelle de régularisation.

Cette fonctionnelle doit représenter les irrégularités spatiales de la solution, que l'on souhaite minimiser. La solution retenue consiste à utiliser l'approximation des dérivées spatiales de l'image :

$$R(\boldsymbol{A}) = \sum_{i=1}^{2PN} \varphi([(\boldsymbol{\nabla}_s \otimes \boldsymbol{I}_P)\boldsymbol{a}]_i), \tag{2.12}$$

où $\boldsymbol{a} = \text{vect}(\boldsymbol{A})$, $\otimes$ représente le produit de Kronecker, $\boldsymbol{I}_P$ est la matrice identité de taille $P \times P$. $\boldsymbol{\nabla}_s \in \mathbb{R}^{2N\times N}$ est l'opérateur gradient spatial, construit comme la concaténation des opérateurs de différences de premier ordre verticaux et horizontaux $\boldsymbol{\nabla}_h$ et $\boldsymbol{\nabla}_v$. Ainsi formé, le terme $(\boldsymbol{\nabla}_s \otimes \boldsymbol{I}_P)\boldsymbol{a} \in \mathbb{R}^{2PN}$ est un vecteur contenant l'ensemble des différences entre deux pixels voisins pour chacune des $P$ cartes d'abondances. La fonction $\varphi(\cdot)$ est appelée fonction

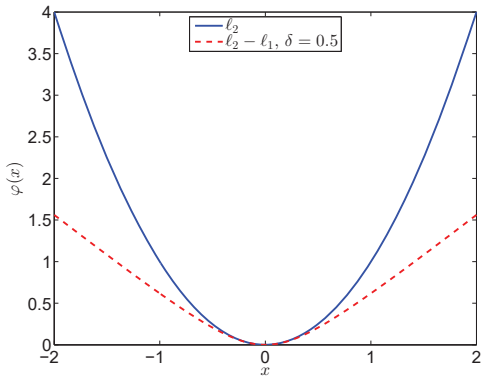



FIGURE 2.2 – Exemples de fonctions de pondération de type $\ell_2$ et $\ell_2 - \ell_1$.

de pondération. De nombreux exemples pour cette fonction sont étudiés dans [Idier, 2013]. La première hypothèse évoquée concernant la régularité de l'ensemble de l'image conduit naturellement à une pénalisation de type quadratique, aussi notée $\ell_2$ :

$$\varphi(x) = \frac{1}{2}x^2. \tag{2.13}$$

La seconde hypothèse, celle d'une image homogène par morceaux, est plus délicate à mettre en œuvre. Idéalement, on aimerait utiliser dans ce cas une fonction de pondération de type $\ell_2 - \ell_0$, qui a un comportement quadratique autour de $0$, et est asymptotiquement constante. Ceci est le meilleur moyen de pénaliser les faibles différences inter-pixel, considérées comme faisant partie d'une même région homogène, tout en laissant la possibilité à des frontières nettes de se former en ne pénalisant pas trop les fortes différences inter-pixel. Ce type de fonction est nécessairement non convexe et, selon l'expression choisie, possiblement non différentiable. Or, dans la suite de cette thèse, la convexité et la différentiabilité seront des propriétés importantes pour la mise en œuvre des algorithmes. C'est pourquoi nous choisissons d'utiliser une fonction de pondération de type $\ell_2 - \ell_1$, qui a un comportement quadratique autour de 0, et est asymptotiquement linéaire :

$$\varphi(x) = \sqrt{\delta^2 + x^2} - \delta, \ \delta > 0. \tag{2.14}$$

Dans cette fonction, le paramètre $\delta$ détermine la position de la zone de transition entre les deux comportements. La figure 2.2 illustre les deux fonctions de pondération choisies.

## 2.3 Formulation du problème

Le problème d'estimation des cartes d'abondances peut donc s'exprimer sous la forme du problème d'optimisation suivant :

$$\underset{\boldsymbol{A}\in\mathbb{R}^{P\times N}}{\text{minimiser}}\ F(\boldsymbol{A}), \ \text{sous les contraintes } \mathcal{C}, \tag{2.15}$$

avec

$$F(\boldsymbol{A}) = \|\boldsymbol{Y} - \boldsymbol{S}\boldsymbol{A}\|_F^2 + \beta R(\boldsymbol{A}), \tag{2.16}$$

$R(\cdot)$ étant définit par les équations (2.12) et (2.13), ou (2.12) et (2.14), selon l'hypothèse de régularité choisie, et $\mathcal{C}$ pouvant être l'un des trois jeux de contraintes présentés en section 2.2.2 : NN, NN+SLO, ou NN+STO.

Il s'agit d'un problème convexe, avec contraintes linéaires, ayant la particularité d'être de grande taille ($P \times N$ variables).

Dans la suite de ce chapitre sont détaillées plusieurs méthodes permettant de résoudre un tel problème, notamment la méthode de points intérieurs qui nous intéresse plus particulièrement dans cette thèse.

## 2.4 Méthodes de résolution existantes

Pour l'estimation des cartes d'abondances en imagerie hyperspectrale, les premiers travaux apportant une évolution par rapport au problème des moindres carrés non contraints et non pénalisés ont conduit à l'algorithme NNLS (*Non-Negative Least Squares*) [Lawson et Hanson, 1974; Bro et De Jong, 1997]. Comme son nom l'indique, cette méthode ajoute la contrainte NN au problème des moindres carrés. Elle est détaillée dans la suite de cette section. Plus tard est apparue la méthode SCLS (*Sum-to-one Constrained Least Squares*) [Settle et Drake, 1993] qui prend en charge la contrainte STO. Ces deux contraintes ont ensuite été réunies dans l'algorithme FCLS (*Fully Constrained Least Squares*) [Heinz et Chang, 2001], dont la description est également donnée ci-après car il s'agit d'un algorithme de référence qui reste largement utilisé et auquel seront comparées les méthodes développées dans cette thèse. Plus récemment, un algorithme d'optimisation convexe basé sur la méthode ADMM (*Alternating Direction Method of Multipliers*) a été développé pour résoudre le problème d'estimation des cartes d'abondances [Bioucas-Dias et Figueiredo, 2010]. De plus, elle a été déclinée en une version incluant un terme de régularisation spatiale [Iordache *et al.*, 2012]. Cette méthode suscitant un intérêt grandissant, elle est également décrite dans ce chapitre et utilisée par la suite à des fins de comparaison. L'approche IPLS (*Interior-Points Least Squares*) qui sera développée durant cette thèse se fonde sur les techniques de points intérieurs [Karmarkar, 1984], plus particulièrement les méthodes primales-duales [Nocedal et Wright, 1999]. Comme nous le verrons dans les chapitres suivants, cette méthode offre un potentiel d'amélioration pour les problèmes de grande taille. Elle est décrite dans la section 2.4.5. Enfin, il existe d'autres méthodes qui ne sont pas traitées dans cette thèse. On peut citer les méthodes de gradient projeté [Lin, 2007; Zymnis *et al.*, 2007] ou encore les méthodes géométriques [Parente et Plaza, 2010].

### 2.4.1 Quelques notions d'optimisation convexe

Avant de présenter les quatres approches, cette section regroupe les prérequis théoriques nécessaires à la compréhension des méthodes [Boyd et Vandenberghe, 2004]. Les notations utilisées sont indépendantes du reste du document.

#### 2.4.1.1 Forme générale d'un problème d'optimisation convexe

Soient $(m, n, p)$ trois entiers naturels. On appelle problème d'optimisation convexe un problème de la forme

$$\begin{aligned} \underset{\boldsymbol{x}\in\mathbb{R}^n}{\text{minimiser}} \quad & f_0(\boldsymbol{x}), \\ \text{sous les contraintes} \quad & f_i(\boldsymbol{x}) \leqslant 0, \quad \forall i = 1 \dots m, \\ & h_i(\boldsymbol{x}) = 0, \quad \forall i = 1 \dots p, \end{aligned} \tag{2.17}$$

où $f_0(\cdot), f_1(\cdot), \dots, f_m(\cdot)$ sont des fonctions convexes de $\mathbb{R}^n$ à valeurs dans $\mathbb{R}$, $h_1(\cdot), \dots, h_p(\cdot)$ sont des fonctions affines de $\mathbb{R}^n$ dans $\mathbb{R}$, de la forme $h_i(x) = \boldsymbol{a}_i^{\mathrm{t}}\boldsymbol{x} - b_i$. Il s'agit de la minimisation d'une fonction convexe, sous des contraintes d'inégalités convexes, et d'égalités affines. Supposons de plus que $f_0(\cdot)$ est strictement convexe, alors le problème est dit strictement convexe.

Deux propriétés importantes de ce type de problème peuvent être relevées :

- Le domaine admissible défini par les contraintes du problème (2.17) est un ensemble convexe de $\mathbb{R}^n$.
- Si $\boldsymbol{x}^\star$ est un optimum local pour le problème (2.17), alors $\boldsymbol{x}^\star$ est l'optimum global pour ce problème.

Notons $p^\star$ la valeur optimale du critère $f_0$, atteinte en $\boldsymbol{x}^\star$.

#### 2.4.1.2 Dualité

**Lagrangien**

Le Lagrangien associé au problème (2.17) est la fonction $L(\cdot,\cdot,\cdot) : \mathbb{R}^n \times \mathbb{R}^m \times \mathbb{R}^p \to \mathbb{R}$ définie par

$$L(\boldsymbol{x}, \boldsymbol{\lambda}, \boldsymbol{\nu}) = f_0(\boldsymbol{x}) + \sum_{i=1}^{m} \lambda_i f_i(\boldsymbol{x}) + \sum_{i=1}^{p} \nu_i h_i(\boldsymbol{x}). \tag{2.18}$$

Il s'agit d'une somme pondérée de la fonction objectif et des fonctions définissant les contraintes. Les variables $\boldsymbol{\lambda} \in \mathbb{R}^m$ et $\boldsymbol{\nu} \in \mathbb{R}^p$ sont appelées multiplicateurs de Lagrange, ou variables duales.

**Fonction duale**

La fonction duale associée au problème (2.17) est la fonction $g(\cdot,\cdot) : \mathbb{R}^m \times \mathbb{R}^p \to \mathbb{R}$ définie par

$$g(\boldsymbol{\lambda}, \boldsymbol{\nu}) = \inf_{\boldsymbol{x}\in\mathbb{R}^n} L(\boldsymbol{x}, \boldsymbol{\lambda}, \boldsymbol{\nu}). \tag{2.19}$$

Notons que par construction, la fonction duale est concave.

**Problème dual**

Le problème dual associé au problème (2.17) est défini par

$$\begin{aligned} \underset{(\boldsymbol{\lambda},\boldsymbol{\nu})\in\mathbb{R}^m\times\mathbb{R}^p}{\text{maximiser}} \quad & g(\boldsymbol{\lambda}, \boldsymbol{\nu}), \\ \text{sous les contraintes} \quad & \lambda_i(\boldsymbol{x}) \geqslant 0, \quad \forall i = 1 \dots m. \end{aligned} \tag{2.20}$$

Le problème dual est concave. Notons sa valeur optimale $d^\star$, atteinte en $(\boldsymbol{\lambda}^\star, \boldsymbol{\nu}^\star)$.

Les propriétés suivantes permettent de faire un lien entre le problème primal et le problème dual :

- On a toujours $d^\star \leqslant p^\star$. On appelle saut de dualité la quantité $p^\star - d^\star$.
- S'il existe un point $\boldsymbol{x} \in \mathbb{R}^n$ strictement admissible pour le problème (2.17) (condition suffisante de qualification de Slater), alors $d^\star = p^\star$. On parle dans ce cas de dualité forte.
- En cas de dualité forte, on a :

$$\boldsymbol{x}^\star = \arg\min_{\boldsymbol{x}} L(\boldsymbol{x}, \boldsymbol{\lambda}^\star, \boldsymbol{\nu}^\star),$$

$$(\boldsymbol{\lambda}^\star, \boldsymbol{\nu}^\star) = \arg\max_{\boldsymbol{\lambda}, \boldsymbol{\nu}} L(\boldsymbol{x}^\star, \boldsymbol{\lambda}, \boldsymbol{\nu}).$$

#### 2.4.1.3 Conditions d'optimalité de Karush-Kuhn-Tucker

En cas de dualité forte, et si les fonctions $f_0(\cdot), f_1(\cdot), \ldots, f_m(\cdot), h_1(\cdot), \ldots, h_p(\cdot)$ sont différentiables, alors la propriété fondamentale suivante est vérifiée :

**Propriété**

$\boldsymbol{x}$ est l'optimum pour le problème (2.17) *si et seulement si* il existe $(\boldsymbol{\lambda}, \boldsymbol{\nu}) \in \mathbb{R}^m \times \mathbb{R}^p$ tels que :

$$\nabla_{\boldsymbol{x}} L(\boldsymbol{x}, \boldsymbol{\lambda}, \boldsymbol{\nu}) = 0, \tag{2.21}$$
$$\lambda_i f_i(\boldsymbol{x}) = 0 \quad \forall i = 1 \ldots m, \tag{2.22}$$
$$f_i(\boldsymbol{x}) \leqslant 0 \quad \forall i = 1 \ldots m, \tag{2.23}$$
$$h_i(\boldsymbol{x}) = 0 \quad \forall i = 1 \ldots p, \tag{2.24}$$
$$\lambda_i \geqslant 0 \quad \forall i = 1 \ldots m. \tag{2.25}$$

Ces conditions nécessaires et suffisantes d'optimalité sont appelées conditions de Karush-Kuhn-Tucker, ou conditions KKT. Elles permettent de remplacer la résolution du problème (2.17) par la vérification des conditions (2.21) à (2.25).

L'objectif des méthodes présentées dans ce chapitre est donc de trouver les variables primales et duales $(\boldsymbol{x}^\star, \boldsymbol{\lambda}^\star, \boldsymbol{\nu}^\star)$ vérifiant les conditions KKT correspondant à la résolution du problème (2.15). Les variables primales $\boldsymbol{x}^\star$ seront alors solutions du problème (2.15).

### 2.4.2 NNLS : *Non-Negative Least Squares*

#### 2.4.2.1 Restriction à un problème sur un seul pixel

La méthode NNLS [Bro et De Jong, 1997] permet la résolution du problème (2.15) sans régularisation ($\beta = 0$), sous les contraintes NN uniquement :

$$\begin{aligned} \underset{\boldsymbol{A}\in\mathbb{R}^{P\times N}}{\text{minimiser}} \quad & F(\boldsymbol{A}) = \frac{1}{2}\left\|\boldsymbol{Y}-\boldsymbol{S}\boldsymbol{A}\right\|_F^2, \\ \text{sous les contraintes} \quad & A_{p,n} \geqslant 0, \quad \forall p = 1\dots P, \forall n = 1\dots N. \end{aligned} \tag{2.26}$$

Remarquons que le critère peut aussi s'écrire

$$F(\boldsymbol{A}) = \sum_{n=1}^{N} \frac{1}{2}\left\|\boldsymbol{y}_n - \boldsymbol{S}\boldsymbol{a}_n\right\|_2^2. \tag{2.27}$$

Ainsi, il s'agit de minimiser une somme de $N$ termes. Les variables $\boldsymbol{a}_n$ intervenant dans un de ces termes sont indépendantes de celles des autres termes. Le problème (2.26) est donc équivalent à $N$ minimisations indépendantes : pour tout $n = 1\dots N$,

$$\begin{aligned} \underset{\boldsymbol{a}_n\in\mathbb{R}^{P}}{\text{minimiser}} \quad & F(\boldsymbol{a}_n) = \frac{1}{2}\left\|\boldsymbol{y}_n-\boldsymbol{S}\boldsymbol{a}_n\right\|_2^2, \\ \text{sous les contraintes} \quad & a_{np} \geqslant 0, \quad \forall p = 1\dots P. \end{aligned} \tag{2.28}$$

La suite de cette section vise à estimer les coefficients d'abondance dans un seul pixel, l'indice $n$ est par conséquent omis.

#### 2.4.2.2 Principe de la méthode

La méthode NNLS fait partie de la famille des méthodes de *contraintes actives* [Nocedal et Wright, 1999]. Elle se base sur l'idée que la solution d'un problème avec contraintes NN correspond à la solution d'un problème non contraint, dont les variables sont un sous ensemble des variables du problème contraint, à savoir les variables non-nulles à la solution.

Par exemple, si le problème contraint (2.28) comporte 5 variables et a pour solution $\hat{\boldsymbol{a}} = [a_1^\star\ a_2^\star\ 0\ a_4^\star\ 0]^{\mathrm{t}}$ avec $a_i^\star \neq 0$ pour $i = \{1, 2, 4\}$, alors les valeurs non-nulles de cette solution correspondent à la solution du problème non contraint comportant 3 variables

$$\underset{\boldsymbol{x}\in\mathbb{R}^3}{\text{minimiser}} \quad F_2(\boldsymbol{x}) = \frac{1}{2}\left\|\ \boldsymbol{y} - \boldsymbol{S}[x_1\ x_2\ 0\ x_3\ 0]^{\mathrm{t}}\right\|_2^2. \tag{2.29}$$

La solution de ce problème peut être calculée de façon analytique :

$$\hat{\boldsymbol{x}} = (\boldsymbol{S}_{\mathcal{P}}^{\mathrm{t}}\boldsymbol{S}_{\mathcal{P}})^{-1} - \boldsymbol{S}_{\mathcal{P}}\boldsymbol{y}_{\mathcal{P}}, \tag{2.30}$$

où $\boldsymbol{S}_{\mathcal{P}}$ et $\boldsymbol{y}_{\mathcal{P}}$ correspondent aux lignes et colonnes d'indices $\{1, 2, 4\}$ de $\boldsymbol{S}$ et $\boldsymbol{y}$.

Ainsi, le problème est résolu si l'on parvient à déterminer les indices pour lesquelles la solution du problème contraint est non-nulle.

Dans l'algorithme NNLS estimant $P$ abondances dans un pixel, l'ensemble des indices $i = \{1\dots P\}$ est séparé en deux sous ensembles complémentaires : l'ensemble actif $\mathcal{A}$ regroupant les indices auxquels la solution est nulle, et l'ensemble passif $\mathcal{P}$ regroupant les indices auxquels la solution est non-nulle.

Initialiser $\mathcal{P} = \{1, \ldots, P\}$ et $\mathcal{A} = \emptyset$
Calculer de la solution du problème non contraint : $\boldsymbol{a}_{LS} = (\boldsymbol{S}^{\mathrm{t}}\boldsymbol{S})^{-1}\boldsymbol{S}^{\mathrm{t}}\boldsymbol{y}$
**Tant que** (La condition (2.33) n'est pas respectée) **faire**
  Mettre à jour les sous ensemble actif $\mathcal{A} = \{i, \boldsymbol{a}_i < 0\}$ et passif $\mathcal{P} = \{i, a_i \geqslant 0\}$
  Projection de la solution non contrainte : $\forall i = 1 \ldots P$, $a_i = \begin{cases} a_{LS\,i} & \text{si } i \in \mathcal{P} \\ 0 & \text{si } i \in \mathcal{A} \end{cases}$
  Calculer les variables duales d'après (2.31) : $\boldsymbol{\lambda} = \boldsymbol{S}^{\mathrm{t}}\boldsymbol{S}\boldsymbol{a}_n - \boldsymbol{S}^{\mathrm{t}}\boldsymbol{y}$
  **Tant que** (La condition (2.34) n'est pas respectée) **faire**
    Transférer l'indice $i = \arg\min_j(\lambda_j)$ de $\mathcal{A}$ vers $\mathcal{P}$
    Projection de la solution non contrainte : $\forall i = 1 \ldots P$, $a_i = \begin{cases} a_{LS\,i} & \text{si } i \in \mathcal{P} \\ 0 & \text{si } i \in \mathcal{A} \end{cases}$
    Calculer les variables duales d'après (2.31) : $\boldsymbol{\lambda} = \boldsymbol{S}^{\mathrm{t}}\boldsymbol{S}\boldsymbol{a} - \boldsymbol{S}^{\mathrm{t}}\boldsymbol{y}$
  **Fait**
  Calculer la solution non contrainte sur les indices passifs : $\boldsymbol{a}_{\mathcal{P}} = (\boldsymbol{S}_{\mathcal{P}}^{\mathrm{t}}\boldsymbol{S}_{\mathcal{P}})^{-1}\boldsymbol{S}_{\mathcal{P}}^{\mathrm{t}}\boldsymbol{y}_{\mathcal{P}}$
  Déduire la solution du problème contraint $\forall i = 1 \ldots P, a_i = \begin{cases} a_{\mathcal{P}\,i} & \text{si } i \in \mathcal{P} \\ 0 & \text{si } i \in \mathcal{A} \end{cases}$
**Fait**

Algorithme 1: Algorithme NNLS

Tous les indices sont initialement placés dans l'ensemble passif $\mathcal{P}$, puis les ensembles $\mathcal{A}$ et $\mathcal{P}$ sont mis à jour itérativement suivant l'Algorithme 1 jusqu'à satisfaction des conditions KKT correspondant au problème (2.28) :

$$\nabla_{\boldsymbol{a}}(F(\boldsymbol{a}) - \sum_{p=1}^{P} \lambda_i a_i) = 0, \tag{2.31}$$

$$\lambda_i a_i = 0 \quad \forall i = 1 \ldots P, \tag{2.32}$$

$$a_i \geqslant 0 \quad \forall i = 1 \ldots P, \tag{2.33}$$

$$\lambda_i \geqslant 0 \quad \forall i = 1 \ldots P. \tag{2.34}$$

#### 2.4.2.3 Algorithme

La méthode NNLS est mise en œuvre par l'Algorithme 1 dans lequel $\boldsymbol{S}_{\mathcal{P}}$ et $\boldsymbol{y}_{\mathcal{P}}$ correspondent aux lignes et colonnes d'indices dans $\mathcal{P}$ de $\boldsymbol{S}$ et $\boldsymbol{y}$.

#### 2.4.2.4 Convergence

Il est important de noter que, d'après [Chang et Heinz, 2000], la convergence de cet algorithme n'est pas garantie car il peut osciller indéfiniment entre deux états sous-optimaux des ensembles $\mathcal{A}$ et $\mathcal{P}$.

### 2.4.3 FCLS : *Fully Constrained Least Squares*

Comme la méthode NNLS, la méthode FCLS [Heinz et Chang, 2001] permet la résolution du problème (2.15) restreint à un seul pixel sans régularisation ($\beta = 0$), mais cette fois sous les contraintes NN et STO :

$$\begin{aligned} \underset{\boldsymbol{a}\in\mathbb{R}^P}{\text{minimiser}} \quad & F(\boldsymbol{a}) = \frac{1}{2}\left\|\boldsymbol{y} - \boldsymbol{S}\boldsymbol{a}_n\right\|_2^2, \\ \text{sous les contraintes} \quad & a_p \geqslant 0, \quad \forall p = 1 \dots P, \\ & \sum_{p=1}^{P} a_p = 1. \end{aligned} \tag{2.35}$$

Elle consiste à utiliser la méthode NNLS en effectuant un prétraitement sur les données d'entrées de l'algorithme afin d'assurer le respect de la contrainte d'égalité sur les données de sortie. La contrainte d'égalité est prise en compte par le rajout d'une équation de mesure. Le vecteur d'observation $\boldsymbol{y}$ devient

$$\boldsymbol{y}' = \begin{bmatrix} 1 \\ \delta\boldsymbol{y} \end{bmatrix}, \tag{2.36}$$

et la matrice des endmembers $\boldsymbol{S}$ devient

$$\boldsymbol{S}' = \begin{bmatrix} \mathbf{1}^{\mathrm{t}} \\ \delta\boldsymbol{S} \end{bmatrix}, \tag{2.37}$$

avec $\delta > 0$ et $\mathbf{1} = [\underbrace{1 \dots 1}_{P}]^{\mathrm{t}}$.

Le problème à résoudre pour un pixel devient alors

$$\begin{aligned} \underset{\boldsymbol{a}\in\mathbb{R}^P}{\text{minimiser}} \quad & F_\delta(\boldsymbol{a}) = \frac{1}{2}\left\|\boldsymbol{y}' - \boldsymbol{S}'\boldsymbol{a}\right\|_2^2, \\ \text{sous les contraintes} \quad & a_p \geqslant 0, \quad \forall p = 1 \dots P. \end{aligned} \tag{2.38}$$

Remarquons que cette manipulation équivaut à l'ajout d'un terme de pénalisation incitant la solution à satisfaire la contrainte STO :

$$
\begin{aligned}
F_\delta(\boldsymbol{a}) &= \frac{1}{2}(\boldsymbol{y}' - \boldsymbol{S}'\boldsymbol{a})^{\mathrm{t}}(\boldsymbol{y}' - \boldsymbol{S}'\boldsymbol{a}) \\
&= \frac{1}{2}\left(\begin{bmatrix} 1 \\ \delta\boldsymbol{y} \end{bmatrix} - \begin{bmatrix} \mathbf{1}^{\mathrm{t}} \\ \delta\boldsymbol{S} \end{bmatrix}\boldsymbol{a}\right)^{\mathrm{t}}\left(\begin{bmatrix} 1 \\ \delta\boldsymbol{y} \end{bmatrix} - \begin{bmatrix} \mathbf{1}^{\mathrm{t}} \\ \delta\boldsymbol{S} \end{bmatrix}\boldsymbol{a}\right) \\
&= \frac{1}{2}\begin{bmatrix} 1 - \mathbf{1}^{\mathrm{t}}\boldsymbol{a} \\ \delta\boldsymbol{y} - \delta\boldsymbol{S}\boldsymbol{a} \end{bmatrix}^{\mathrm{t}}\begin{bmatrix} 1 - \mathbf{1}^{\mathrm{t}}\boldsymbol{a} \\ \delta\boldsymbol{y} - \delta\boldsymbol{S}\boldsymbol{a} \end{bmatrix} \\
&= \frac{1}{2}(\delta\boldsymbol{y} - \delta\boldsymbol{S}\boldsymbol{a})^{\mathrm{t}}(\delta\boldsymbol{y} - \delta\boldsymbol{S}\boldsymbol{a}) + \frac{1}{2}(1 - \mathbf{1}^{\mathrm{t}}\boldsymbol{a})(1 - \mathbf{1}^{\mathrm{t}}\boldsymbol{a}) \\
&= \delta^2 F(\boldsymbol{a}) + \frac{1}{2}\left(1 - \sum_{p=1}^{P}\boldsymbol{a}_p\right)^2 .
\end{aligned} \tag{2.39}
$$

Par cette manipulation, la méthode FCLS se ramène donc à l'utilisation de la méthode NNLS pour résoudre le problème (2.38).

Il reste le choix du paramètre $\delta$ qui assure le compromis entre respect de la contrainte d'égalité et adéquation aux données. En pratique, ce paramètre est calculé par

$$
\delta = \frac{\max\limits_{l=1\ldots L} y_l}{10}. \tag{2.40}
$$

### 2.4.4 ADMM : *Alternating Direction Method of Multipliers*

#### 2.4.4.1 Principe général

La méthode ADMM a été proposée pour la première fois par [Glowinski et Marroco, 1975] et [Gabay et Mercier, 1976], puis analysée dans de nombreuses publications. Un résumé des développements théoriques liés à ADMM est présenté dans [Boyd *et al.*, 2011].

Soient $(m, n, p)$ trois entiers naturels. L'algorithme ADMM permet de résoudre un problème de la forme

$$
\begin{aligned}
\underset{\boldsymbol{x}\in\mathbb{R}^n\ \boldsymbol{z}\in\mathbb{R}^m}{\text{minimiser}} \quad & f(\boldsymbol{x}) + g(\boldsymbol{z}), \\
\text{sous les contraintes} \quad & \boldsymbol{A}\boldsymbol{x} + \boldsymbol{B}\boldsymbol{z} = \boldsymbol{c}.
\end{aligned} \tag{2.41}
$$

où $f : \mathbb{R}^n \to \mathbb{R} \cup \{\infty\}$ et $g : \mathbb{R}^m \to \mathbb{R} \cup \{\infty\}$ sont des fonctions convexes, $\boldsymbol{A} \in \mathbb{R}^{p\times n}$, $\boldsymbol{B} \in \mathbb{R}^{p\times m}$, $\boldsymbol{c} \in \mathbb{R}^p$. La notion de convexité est ici étendue aux fonctions à valeurs éventuellement infinies.

Le Lagrangien associé à ce problème s'écrit

$$
L(\boldsymbol{x}, \boldsymbol{z}, \boldsymbol{y}) = f(\boldsymbol{x}) + g(\boldsymbol{z}) + \boldsymbol{y}^{\mathrm{t}}(\boldsymbol{A}\boldsymbol{x} + \boldsymbol{B}\boldsymbol{z} - \boldsymbol{c}), \tag{2.42}
$$

et les conditions KKT :

$$\nabla_{\boldsymbol{x}} L(\boldsymbol{x}, \boldsymbol{z}, \boldsymbol{y}) = 0, \tag{2.43}$$
$$\nabla_{\boldsymbol{z}} L(\boldsymbol{x}, \boldsymbol{z}, \boldsymbol{y}) = 0, \tag{2.44}$$
$$\boldsymbol{A}\boldsymbol{x} + \boldsymbol{B}\boldsymbol{z} - \boldsymbol{c} = 0. \tag{2.45}$$

La méthode ADMM fait partie des méthodes de Lagrangien augmenté [Nocedal et Wright, 1999]. Ces méthodes incluent dans le Lagrangien un terme lié à la contrainte d'égalité qui peut être assimilé à un terme de pénalisation, bien que la contrainte soit maintenue. Cette modification permet généralement d'obtenir plus rapidement la solution du problème. Le Lagrangien augmenté utilisé dans la méthode ADMM est

$$L_\rho(\boldsymbol{x}, \boldsymbol{z}, \boldsymbol{y}) = f(\boldsymbol{x}) + g(\boldsymbol{z}) + \boldsymbol{y}^{\mathrm{t}}(\boldsymbol{A}\boldsymbol{x} + \boldsymbol{B}\boldsymbol{z} - \boldsymbol{c}) + \frac{\rho}{2} \left\| \boldsymbol{A}\boldsymbol{x} + \boldsymbol{B}\boldsymbol{z} - \boldsymbol{c} \right\|_2^2, \tag{2.46}$$

avec un poids $\rho > 0$.

Les conditions KKT à vérifier deviennent alors

$$\nabla_{\boldsymbol{x}} L_\rho(\boldsymbol{x}, \boldsymbol{z}, \boldsymbol{y}) = 0, \tag{2.47}$$
$$\nabla_{\boldsymbol{z}} L_\rho(\boldsymbol{x}, \boldsymbol{z}, \boldsymbol{y}) = 0, \tag{2.48}$$
$$\boldsymbol{A}\boldsymbol{x} + \boldsymbol{B}\boldsymbol{z} - \boldsymbol{c} = 0. \tag{2.49}$$

Afin de trouver les variables primales $(\boldsymbol{x}, \boldsymbol{z})$ et duales $\boldsymbol{y}$ vérifiant ces conditions, une méthode itérative est employée. Une méthode classique consiste à passer de l'itération $k$ à l'itération $k+1$ grâce à une minimisation du Lagrangien suivant les variables primales suivie d'une maximisation suivant les variables duales :

$$(\boldsymbol{x}^{k+1}, \boldsymbol{z}^{k+1}) = \underset{\boldsymbol{x}\in\mathbb{R}^n}{\arg\min} \underset{\boldsymbol{z}\in\mathbb{R}^m}{} L_\rho(\boldsymbol{x}, \boldsymbol{z}, \boldsymbol{y}^k), \tag{2.50}$$
$$\boldsymbol{y}^{k+1} = \boldsymbol{y}^k + \rho(\boldsymbol{A}\boldsymbol{x}^{k+1} + \boldsymbol{B}\boldsymbol{z}^{k+1} - \boldsymbol{c}). \tag{2.51}$$

La mise à jour des variables primales à l'équation (2.50) est effectuée en minimisant le Lagrangien de façon exacte ou approchée. Dans cette étape, les variables $\boldsymbol{x}$ et $\boldsymbol{z}$ ne sont pas séparables à cause du terme ajouté au Lagrangien. La mise à jour des variables duales à l'équation (2.51) est effectuée en maximisant le Lagrangien en utilisant la méthode du gradient avec un pas fixe choisi comme étant le poids du Lagrangien augmenté $\rho$.

La méthode ADMM propose de séparer l'étape de minimisation (2.50) en deux minimisations, respectivement suivant $\boldsymbol{x}$ et suivant $\boldsymbol{z}$. Cette modification peut être interprétée comme le remplacement de l'équation (2.50) par une itération de la méthode de Gauss-Seidel permettant de minimiser suivant $\boldsymbol{x}$ et $\boldsymbol{z}$ conjointement :

$$\boldsymbol{x}^{k+1} = \underset{\boldsymbol{x}\in\mathbb{R}^n}{\arg\min}\, L_\rho(\boldsymbol{x}, \boldsymbol{z}^k, \boldsymbol{y}^k), \tag{2.52}$$
$$\boldsymbol{z}^{k+1} = \underset{\boldsymbol{z}\in\mathbb{R}^m}{\arg\min}\, L_\rho(\boldsymbol{x}^{k+1}, \boldsymbol{z}, \boldsymbol{y}^k), \tag{2.53}$$
$$\boldsymbol{y}^{k+1} = \boldsymbol{y}^k + \rho(\boldsymbol{A}\boldsymbol{x}^{k+1} + \boldsymbol{B}\boldsymbol{z}^{k+1} - \boldsymbol{c}). \tag{2.54}$$

#### 2.4.4.2 Application à l'estimation des cartes d'abondances

L'algorithme d'estimation des cartes d'abondances basé sur ADMM est développé dans [Bioucas-Dias et Figueiredo, 2010] sous le nom SUnSAL (*Sparse Unmixing by variable Splitting and Augmented Lagrangian*). Tout comme FCLS, cette méthode permet la résolution du problème (2.15) pour un pixel sans régularisation et sous les contraintes NN et STO :

$$\begin{aligned} \underset{\boldsymbol{a}\in\mathbb{R}^P}{\text{minimiser}} \quad & F(\boldsymbol{a}) = \frac{1}{2}\left\|\boldsymbol{y}-\boldsymbol{S}\boldsymbol{a}\right\|_2^2, \\ \text{sous les contraintes} \quad & a_p \geqslant 0, \quad \forall p = 1\dots P, \\ & \sum_{p=1}^{P} a_p = 1. \end{aligned} \tag{2.55}$$

Ce problème peut être réécrit sous la forme d'une minimisation de deux fonctions convexes sous contrainte d'égalité afin d'être résolu par la méthode ADMM :

$$\begin{aligned} \underset{\boldsymbol{a}\in\mathbb{R}^P\ \boldsymbol{z}\in\mathbb{R}^P}{\text{minimiser}} \quad & f(\boldsymbol{a}) + g(\boldsymbol{z}), \\ \text{sous les contraintes} \quad & \boldsymbol{a}-\boldsymbol{z} = 0, \end{aligned} \tag{2.56}$$

avec

$$f(\boldsymbol{a}) = \frac{1}{2}\left\|\boldsymbol{y}-\boldsymbol{S}\boldsymbol{a}\right\|_2^2 + \mathcal{I}_{\{1\}}(\mathbf{1}^{\mathrm{t}}\boldsymbol{x}), \tag{2.57}$$

$$g(\boldsymbol{z}) = \mathcal{I}_{\mathbb{R}_+^P}(\boldsymbol{z}). \tag{2.58}$$

$\mathcal{I}_S$ est la fonction indicatrice de l'ensemble $S$ (i.e., $\mathcal{I}_S(\boldsymbol{x}) = 0$ si $\boldsymbol{x} \in S$ et $\mathcal{I}_S(\boldsymbol{x}) = +\infty$ sinon).

Les équation de mise à jour des variables (2.52) à (2.54) deviennent

$$\boldsymbol{a}^{k+1} = \boldsymbol{B}^{-1}\boldsymbol{w}^k - \boldsymbol{c}(\mathbf{1}^{\mathrm{t}}\boldsymbol{B}^{-1}\boldsymbol{w}^k - 1), \tag{2.59}$$

$$\boldsymbol{z}^{k+1} = \Pi_{\mathbb{R}_+^P}(\boldsymbol{a}^{k+1} - \boldsymbol{\lambda}^k), \tag{2.60}$$

$$\boldsymbol{\lambda}^{k+1} = \boldsymbol{y}^k - (\boldsymbol{a}^{k+1} - \boldsymbol{z}^{k+1}). \tag{2.61}$$

avec $\boldsymbol{B} = \boldsymbol{S}^{\mathrm{t}}\boldsymbol{S} + \rho I_P$, $\boldsymbol{c} = \boldsymbol{B}^{-1}\mathbf{1}(\mathbf{1}^{\mathrm{t}}\boldsymbol{B}^{-1}\mathbf{1})^{-1}$, $\boldsymbol{w}^k = \boldsymbol{S}^{\mathrm{t}}\boldsymbol{y} + \rho(\boldsymbol{z}^k + \boldsymbol{\lambda}^k)$. $\Pi_S$ est l'opérateur de projection sur l'ensemble $S$.

L'équation (2.59) est obtenue à partir de l'équation (2.52) et de la définition de la fonction $f$ par la méthode des multiplicateurs de Lagrange. Les équations (2.60) et (2.61) sont la traduction directe des équations (2.53) et (2.54) appliquées au problème (2.56).

L'algorithme est arrêté lorsque les critères d'arrêt suivants sont vérifiés :

$$res_p^k < tol_1, \tag{2.62}$$

$$res_d^k < tol_2, \tag{2.63}$$

avec $res_p^k = \left\|\boldsymbol{a}^k - \boldsymbol{z}^k\right\|_2$, $res_d^k = \left\|\boldsymbol{z}^k - \boldsymbol{z}^{k-1}\right\|_2$, $tol_1 > 0$ et $tol_2 > 0$. Il est montré que les termes $res_p^k$ et $res_d^k$ ainsi calculés correspondent respectivement au résidu primal et au résidu dual.

#### 2.4.4.3 Algorithme

Dans le cadre de l'estimation des cartes d'abondances, la méthode ADMM est mise en œuvre par l'Algorithme 2 dont le code Matlab est disponible à l'adresse

`http://www.lx.it.pt/~bioucas`.

Les valeurs choisies par défaut pour les paramètres de l'algorithmes sont $\rho = 10^{-2}$ et $tol_1 = tol_2 = P \cdot 10^{-8}$.

Initialiser $k = 0$, $\rho > 0$, $tol_1 > 0$, $tol_2 > 0$, $\boldsymbol{z}^0$ et $\boldsymbol{\lambda}^0$
**Tant que** ((2.62) et (2.63) ne sont pas vérifiées) **faire**
  Calculer $\boldsymbol{a}^{k+1}$ d'après (2.59)
  Calculer $\boldsymbol{z}^{k+1}$ d'après (2.60)
  Calculer $\boldsymbol{\lambda}^{k+1}$ d'après (2.61)
  $k = k + 1$
**Fait**

Algorithme 2: Algorithme ADMM

#### 2.4.4.4 Convergence

La convergence de l'algorithme 2 est démontrée dans [Bioucas-Dias et Figueiredo, 2010] et se base sur les résultats de [Eckstein et Bertsekas, 1992] :
**Théorème 2.1.** *La suite* $\boldsymbol{a}^k$ *définie par les équations* (2.59), (2.60) *et* (2.61) *converge vers la solution du problème* (2.56) *lorsque* $k \to +\infty$.

#### 2.4.4.5 Prise en compte de la pénalisation spatiale

Dans [Iordache *et al.*, 2012], l'algorithme ADMM est employé pour résoudre le problème avec une pénalisation spatiale de type $\ell_1$. Le problème s'écrit :

$$\begin{aligned} \underset{\boldsymbol{A}\in\mathbb{R}^{P\times N}}{\text{minimiser}} \quad & F(\boldsymbol{A}) = \frac{1}{2}\left\|\boldsymbol{Y} - \boldsymbol{S}\boldsymbol{A}\right\|_F^2 + \beta \left\|\boldsymbol{\nabla}_S \boldsymbol{A}^{\mathrm{t}}\right\|_{1,1}, \\ \text{sous les contraintes} \quad & \text{NN et STO}, \end{aligned} \tag{2.64}$$

avec $\|\boldsymbol{X}\|_{1,1} = \sum_i \sum_j X_{i,j}$. L'idée est de réécrire ce prolème sous la forme d'une minimisation de plusieurs fonctions convexes dont les variables sont liées par des contraintes d'égalité :

$$\begin{aligned} \underset{\boldsymbol{A},\boldsymbol{V}_1,\boldsymbol{V}_2,\boldsymbol{V}_3,\boldsymbol{V}_4,\boldsymbol{V}_5}{\text{minimiser}} \quad & \frac{1}{2}\left\|\boldsymbol{Y}-\boldsymbol{V}_1\right\|_F^2 + \mathcal{I}_{\{1\}}(\mathbf{1}^{\mathrm{t}}\boldsymbol{V}_2) + \mathcal{I}_{\mathbb{R}_+^P}(\boldsymbol{V}_3) + \beta\left\|\boldsymbol{V}_5\right\|_{1,1}, \qquad (2.65) \\ \text{sous les contraintes} \quad & \boldsymbol{S}\boldsymbol{A} = \boldsymbol{V}_1, \\ & \boldsymbol{A} = \boldsymbol{V}_2, \\ & \boldsymbol{A} = \boldsymbol{V}_3, \\ & \boldsymbol{A} = \boldsymbol{V}_4, \\ & \boldsymbol{\nabla}_S \boldsymbol{V}_4^{\mathrm{t}} = \boldsymbol{V}_5, \end{aligned} \qquad (2.66)$$

Ce problème peut être résolu par un algorithme de type ADMM dans lequel les deux étapes de minimisation du lagrangien augmenté sont remplacées par 6 minimisations respectivement suivant les variables $\boldsymbol{A}$, $\boldsymbol{V}_1$, $\boldsymbol{V}_2$, $\boldsymbol{V}_3$, $\boldsymbol{V}_4$ et $\boldsymbol{V}_5$.

### 2.4.5 IPLS : *Interior-Points Least Squares*

Les méthodes de *points intérieurs* ont la spécificité de garantir le respect des contraintes d'inégalité strictement, d'où le qualificatif *intérieurs*. Le principe d'optimisation par points intérieurs est apparu dans les années 1950 grâce à la définition de la fonction barrière logarithmique [Frisch, 1955]. Le terme de points intérieurs est introduit pour la première fois dans [Fiacco et McCormick, 1968], puis un algorithme à convergence polynomiale est proposé dans [Karmarkar, 1984]. Depuis, plusieurs variantes ont été développées telles que le *suivi de chemin central*, la *barrière logarithmique*, et la méthode *primale-duale* [Nocedal et Wright, 1999]. La méthode IPLS met en œuvre un algorithme itératif de type points intérieurs avec une approche primale-duale [Mehrotra, 1992; Armand *et al.*, 2000].

#### 2.4.5.1 Principe général

La méthode primale-duale de points intérieurs permet de résoudre un problème de la forme

$$\begin{aligned} \underset{\boldsymbol{x}\in\mathbb{R}^n}{\text{minimiser}} \quad & f(\boldsymbol{x}), \qquad (2.67) \\ \text{sous les contraintes} \quad & \boldsymbol{c}(\boldsymbol{x}) \geqslant 0, \end{aligned}$$

où $f : \mathbb{R}^n \to \mathbb{R}$ et $\boldsymbol{c} : \mathbb{R}^n \to \mathbb{R}^m$. L'inégalité $\boldsymbol{c}(\boldsymbol{x}) \geqslant 0$ signifie que chaque composante $c_i : \mathbb{R}^n \to \mathbb{R}$ de $\boldsymbol{c}$ doit respecter l'inégalité $c_i(\boldsymbol{x}^\star) \geqslant 0$ à la solution pour tout $i = 1, \ldots, m$. Les fonctions $f$ et $-c_i$ sont supposées convexes et deux fois différentiables. Ainsi le problème (2.67) est convexe.

Le Lagrangien de ce problème s'écrit

$$L(\boldsymbol{x}, \boldsymbol{\lambda}) = f(\boldsymbol{x}) + \boldsymbol{\lambda}^{\mathrm{t}}\boldsymbol{c}(\boldsymbol{x}). \qquad (2.68)$$

Les conditions KKT s'écrivent

$$\nabla_{\boldsymbol{x}} L(\boldsymbol{x}, \boldsymbol{\lambda}) = 0, \tag{2.69}$$
$$\boldsymbol{C}(\boldsymbol{x})\boldsymbol{\lambda} = 0, \tag{2.70}$$
$$\boldsymbol{c}(\boldsymbol{x}) \geqslant 0, \tag{2.71}$$
$$\boldsymbol{\lambda} \geqslant 0, \tag{2.72}$$

avec $\boldsymbol{C} = \text{diag}(\boldsymbol{c}_1, \ldots, \boldsymbol{c}_m)$.

L'approche par points intérieurs consiste à estimer de façon conjointe $\boldsymbol{x}$ et $\boldsymbol{\lambda}$ en résolvant une séquence de problèmes correspondant à des versions perturbées des conditions KKT paramétrées par une suite de paramètres positifs $\{\mu_k\}_{k\in\mathbb{N}}$ convergeant vers $0$ :

$$\nabla_{\boldsymbol{x}} L(\boldsymbol{x}, \boldsymbol{\lambda}) = 0, \tag{2.73}$$
$$\boldsymbol{C}(\boldsymbol{x})\boldsymbol{\lambda} = \mu_k \mathbf{1}, \tag{2.74}$$
$$\boldsymbol{c}(\boldsymbol{x}) \geqslant 0, \tag{2.75}$$
$$\boldsymbol{\lambda} \geqslant 0. \tag{2.76}$$

Chacun de ces problèmes est résolu de façon itérative. La méthode employée peut varier, mais la forme la plus courante est la suivante : une itération $i$ de l'algorithme de résolution des équations KKT perturbées par un paramètre $\mu_k > 0$ se compose de deux étapes. La première consiste à calculer les directions primales $\boldsymbol{d}^{\boldsymbol{x}}_{k,i}$ et duales $\boldsymbol{d}^{\boldsymbol{\lambda}}_{k,i}$ issues de la méthode de Newton permettant d'approcher la solution des équations (2.73) et (2.74). La deuxième consiste à calculer les pas $\alpha^{\boldsymbol{x}}_{k,i}$ $\alpha^{\boldsymbol{\lambda}}_{k,i}$ assurant à la fois la vérification stricte des équations (2.75) et (2.76) (d'où le nom de *points intérieurs*), et une décroissance suffisante d'une fonction de mérite permettant d'assurer la convergence de l'algorithme. Ces deux étapes permettent de mettre à jour les variables primales et duales selon la règle

$$\boldsymbol{x}_{k,i+1} = \boldsymbol{x}_{k,i} + \alpha^{\boldsymbol{x}}_{k,i} \boldsymbol{d}^{\boldsymbol{x}}_{k,i}, \tag{2.77}$$
$$\boldsymbol{\lambda}_{k,i+1} = \boldsymbol{\lambda}_{k,i} + \alpha^{\boldsymbol{\lambda}}_{k,i} \boldsymbol{d}^{\boldsymbol{\lambda}}_{k,i}. \tag{2.78}$$

Lorsque les équations (2.73) à (2.76) sont satisfaites, le paramètre de perturbation est diminué : $\mu_{k+1} < \mu_k$. La résolution d'un problème perturbé par un paramètre $\mu_k$ est arrêtée lorsque les résidus primal et dual de ce problème sont proches de 0. La résolution du problème global est arrêtée lorsque les résidus primal et dual sont proches de 0 ou lorsque le paramètre $\mu_k$ est proche de 0. Le détail de chaque étape de l'algorithme est donné ci-après.

**Calcul des directions primales et duales**

Les directions de Newton primales et duales sont calculées en résolvant le système linéaire

$$\begin{bmatrix} \nabla^2 f(\boldsymbol{x}_{k,i}) & -\nabla \boldsymbol{c}(\boldsymbol{x}_{k,i}) \\ \boldsymbol{\Lambda}_{k,i} \nabla \boldsymbol{c}(\boldsymbol{x}_{k,i})^{\mathrm{t}} & \boldsymbol{C}(\boldsymbol{x}_{k,i}) \end{bmatrix} \begin{bmatrix} \boldsymbol{d}^{\boldsymbol{x}}_{k,i} \\ \boldsymbol{d}^{\boldsymbol{\lambda}}_{k,i} \end{bmatrix} = \begin{pmatrix} -\nabla f(\boldsymbol{x}_{k,i}) + \nabla \boldsymbol{c}(\boldsymbol{x}_{k,i})\boldsymbol{\lambda}_{k,i} \\ \mu_k \mathbf{1} - \boldsymbol{C}(\boldsymbol{x}_{k,i})\boldsymbol{\lambda}_{k,i} \end{pmatrix}, \tag{2.79}$$

où $\nabla f(\cdot)$ et $\nabla^2 f(\cdot)$ sont respectivement le gradient et le Hessien du critère $f(\cdot)$, et $\boldsymbol{\Lambda}_{k,i} = \text{diag}(\boldsymbol{\lambda}_{k,i})$.

Le système (2.79) n'est pas inversé de façon directe. La technique de [Armand *et al.*, 2000] consiste à effectuer le calcul des directions en deux étapes : la direction primale $\boldsymbol{d}^{\boldsymbol{x}}_{k,i}$ est d'abord obtenue par inversion du système réduit

$$\boldsymbol{H}_{k,i}\boldsymbol{d}^{\boldsymbol{x}}_{k,i} = -\boldsymbol{g}_{k,i}, \tag{2.80}$$

avec

$$\boldsymbol{H}_{k,i} = \nabla^2 f(\boldsymbol{x}_{k,i}) + \nabla \boldsymbol{c}(\boldsymbol{x}_{k,i})\boldsymbol{C}(\boldsymbol{x}_{k,i})^{-1}\boldsymbol{\Lambda}_{k,i}\nabla \boldsymbol{c}(\boldsymbol{x}_{k,i})^{\mathrm{t}}, \tag{2.81}$$

$$\boldsymbol{g}_{k,i} = \nabla f(\boldsymbol{x}_{k,i}) - \mu_k \nabla \boldsymbol{c}(\boldsymbol{x}_{k,i})\boldsymbol{C}(\boldsymbol{x}_{k,i})^{-1}\mathbf{1}. \tag{2.82}$$

Ce système réduit s'obtient par substitution de $\boldsymbol{d}^{\boldsymbol{\lambda}}_{k,i}$ dans la première équation de (2.79) par son expression déduite de la seconde partie de ce système

$$\boldsymbol{d}^{\lambda}_{k,i} = \boldsymbol{C}(\boldsymbol{x}_{k,i})^{-1}\left(\mu_k \mathbf{1} - \boldsymbol{C}(\boldsymbol{x}_{k,i})\boldsymbol{\lambda}_{k,i} - \boldsymbol{\Lambda}_{k,i}\nabla \boldsymbol{c}(\boldsymbol{x}_{k,i})^{\mathrm{t}}\boldsymbol{d}^{\boldsymbol{x}}_{k,i}\right). \tag{2.83}$$

Finalement, après obtention des directions primales par l'inversion de (2.80), l'expression (2.83) est utilisée pour déterminer les directions duales.

**Recherche de pas**

Alors qu'une même valeur est généralement choisie pour le pas primal et le pas dual, [Armand *et al.*, 2013] montre qu'il est plus avantageux de considérer deux pas différents $\alpha^{\boldsymbol{x}}_{k,i} = \alpha_{k,i}\alpha^{\boldsymbol{x}}_{k,i_{\max}}$ et $\alpha^{\boldsymbol{\lambda}}_{k,i} = \alpha_{k,i}\alpha^{\boldsymbol{\lambda}}_{k,i_{\max}}$. Ces valeurs sont déterminées de façon à garantir la convergence de l'algorithme et à vérifier les deux contraintes d'inégalité de (2.75) et (2.76).

D'abord, les valeurs maximales de $\alpha^{\boldsymbol{x}}_{k,i}$ et $\alpha^{\boldsymbol{\lambda}}_{k,i}$ permettant de respecter les contraintes sont calculées par

$$\alpha^{\boldsymbol{x}}_{k,i_{\max}} = \min\left(\min(\boldsymbol{x}_{k,i}{\boldsymbol{d}^{\boldsymbol{x}}_{k,i}}^{-1}), 1\right), \tag{2.84}$$

$$\alpha^{\boldsymbol{\lambda}}_{k,i_{\max}} = \min\left(\min(\boldsymbol{\lambda}_{k,i}{\boldsymbol{d}^{\boldsymbol{\lambda}}_{k,i}}^{-1}), 1\right). \tag{2.85}$$

Ensuite, la convergence de l'algorithme est garantie sous réserve que les pas entraînent une décroissance suffisante d'une fonction de mérite primale-duale $\Psi_{\mu_k}(\boldsymbol{x}, \boldsymbol{\lambda})$ liée aux conditions d'optimalité du problème [Forsgren *et al.*, 2002]. La fonction de mérite primale-duale employée est celle de [Armand *et al.*, 2000] définie par

$$\Psi_{\mu_{k,i}}(\boldsymbol{x}, \boldsymbol{\lambda}) = f(\boldsymbol{x}) - \mu_{k,i}\mathbf{1}^{\mathrm{t}}\ln(\boldsymbol{c}(\boldsymbol{x}_{k,i})) + \boldsymbol{\lambda}^{\mathrm{t}}_{k,i}\boldsymbol{c}(\boldsymbol{x}_{k,i}) - \mu_{k,i}\mathbf{1}^{\mathrm{t}}\ln(\boldsymbol{\lambda}^{\mathrm{t}}_{k,i}\boldsymbol{c}(\boldsymbol{x}_{k,i})). \tag{2.86}$$

On remarque la présence de deux fonctions barrières logarithmiques permettant de satisfaire strictement les contraintes d'inégalité (2.75) et (2.76). La décroissance suffisante de la fonction de mérite se traduit par la vérification de la condition d'Armijo

$$\psi_{\mu_{k,i}}(\alpha_{k,i}) - \psi_{\mu_{k,i}}(0) \leqslant \sigma\, \alpha_{k,i}\nabla\psi_{\mu_{k,i}}(0) \quad \text{avec} \quad \sigma \in (0, 1/2), \tag{2.87}$$

où

$$\psi_{\mu_k}(\alpha_{k,i}) = \Psi_{\mu_k}(\boldsymbol{x}_{k,i} + \alpha_{k,i}\alpha^{\boldsymbol{x}}_{k,i_{\max}}\boldsymbol{d}^{\boldsymbol{x}}_{k,i}, \boldsymbol{\lambda}_{k,i} + \alpha_{k,i}\alpha^{\boldsymbol{\lambda}}_{k,i_{\max}}\boldsymbol{d}^{\lambda}_{k,i}). \tag{2.88}$$

L'obtention d'un pas $\alpha_{k,i}$ vérifiant (2.87) est réalisée par une stratégie de rebroussement (ou backtracking) [Nocedal et Wright, 1999]. Partant d'un pas initial $\alpha^0_{k,i} = 1$, et s'il ne vérifie pas la condition d'Armijo, on rebrousse chemin en essayant des pas plus petits $\alpha^0_{k,i}\tau$, $\alpha^0_{k,i}\tau^2$, etc. avec $\tau \in (0, 1)$.

**Règle de mise à jour du paramètre barrière et critères d'arrêt**

Le critère d'arrêt global de l'algorithme IPLS est régi par deux conditions :

$$\mu_k \leqslant \mu_{\min}, \tag{2.89}$$

$$\left(\|\boldsymbol{r}^{\text{prim}}_{0,k,i}\| + \|\boldsymbol{r}^{\text{dual}}_{0,k,i}\|\right) \leqslant \epsilon_0. \tag{2.90}$$

où les résidus primal et dual sont définis à partir des équations KKT par

$$\boldsymbol{r}^{\text{prim}}_{\mu_k,k,i} = \nabla f(\boldsymbol{x}_{k,i}) - \nabla \boldsymbol{c}(\boldsymbol{x}_{k,i})\boldsymbol{\lambda}_{k,i}, \tag{2.91}$$

$$\boldsymbol{r}^{\text{dual}}_{\mu_k,k,i} = \boldsymbol{C}(\boldsymbol{x}_{k,i})\boldsymbol{\lambda}_{k,i} - \mu_k \mathbf{1}. \tag{2.92}$$

Le paramètre de barrière $\mu_k$ est mis à jour selon la règle de $\mu$-criticité définie dans [El-Bakry *et al.*, 1996] :

$$\mu_k = \theta \frac{\delta_k}{m}, \tag{2.93}$$

où $\theta \in (0,1)$, et $\delta_k = \boldsymbol{c}(\boldsymbol{x}_{k,i})^{\text{t}}\boldsymbol{\lambda}_k$ est appelé par abus de langage saut de dualité, car il représente la différence entre la valeur de la fonction objectif et la valeur de la fonction duale à l'itéré courant.

L'arrêt de la boucle interne de résolution des conditions KKT perturbées est régi par deux conditions :

$$\|\boldsymbol{r}^{\text{prim}}_{\mu_k,k,i}\|_\infty \leqslant \epsilon^{\text{prim}}_k, \tag{2.94}$$

$$\|\boldsymbol{r}^{\text{dual}}_{\mu_k,k,i}\|_1/(n_d N) \leqslant \epsilon^{\text{dual}}_k, \tag{2.95}$$

avec $\epsilon^{\text{prim}}_k = \eta^{\text{prim}}\mu_k$, $\epsilon^{\text{dual}}_k = \eta^{\text{dual}}\mu_k$, où $\eta^{\text{prim}} > 0$ et $\eta^{\text{dual}} \in (1, \theta^{-1})$.

### 2.4.5.2 Algorithme

La méthode IPLS est mise en œuvre par l'Algorithme 3 pour la résolution du problème (2.67).

Choisir $\mu_{min} > 0$, $\epsilon_0 > 0$, $\sigma \in (0, 1/2)$, $\tau \in (0, 1)$, $\theta \in (0, 1)$, $\eta^{\text{prim}} \in (0, 1)$, $\eta^{\text{dual}} \in (0, \theta^{-1})$
Initialiser $\boldsymbol{\lambda}_0 > 0$, $\boldsymbol{x}_0$ vérifiant $\boldsymbol{c}(\boldsymbol{x}_0) < 0$, $k = 0$
**Tant que** ((2.89) et (2.90) ne sont pas vérifiées) **faire**
  Calculer le paramètre barrière $\mu_k$ d'après (2.93)
  Initialiser $i = 0$
  **Tant que** ((2.94) et (2.95) ne sont pas vérifiées) **faire**
    Calculer les directions primales $\boldsymbol{d}^{\boldsymbol{x}}_{k,i}$ d'après (2.80)
    Calculer les directions duales $\boldsymbol{d}^{\boldsymbol{\lambda}}_{k,i}$ d'après (2.83)
    Calculer les pas maximum $\alpha^{\boldsymbol{x}}_{k,i_{\max}}$ et $\alpha^{\boldsymbol{\lambda}}_{k,i_{\max}}$ d'après 2.84 et 2.84
    Initialiser le pas commun $\alpha_{k,i} = 1$
    Mettre à jour les variables primales-duales $(\boldsymbol{x}_{k,i}, \boldsymbol{\lambda}_{k,i})$ d'après (2.77) et (2.78)
    **Tant que** ((2.87) n'est pas vérifiée) **faire**
      Diminuer le pas $\alpha_{k,i} = \alpha_{k,i}\tau$
      Mettre à jour les variables primales-duales $(\boldsymbol{x}_{k,i}, \boldsymbol{\lambda}_{k,i})$ d'après (2.77) et (2.78)
    **Fait**
    $i = i + 1$
  **Fait**
  $k = k + 1$
**Fait**

Algorithme 3: Algorithme IPLS

L'Algorithme 3 dépend de plusieurs paramètres qui règlent la précision de la résolution du problème contraint initial ($\mu_{min}$, $\epsilon_0$), la mise à jour du paramètre barrière ($\theta$), la précision de la résolution des conditions KKT perturbées ($\eta^{\text{prim}}$, $\eta^{\text{dual}}$), et la recherche de pas ($\sigma$, $\tau$). Ces paramètres influent sur la vitesse de convergence de l'algorithme. Nous avons obtenu le meilleur compromis entre le nombre d'itérations de la boucle interne et le nombre d'itérations globales pour les valeurs suivantes :

$$\begin{array}{llll} \mu_{min} = 10^{-9} & \epsilon_0 = 10^{-7} & \theta = 0.5 & \eta^{\text{prim}} = 100 \\ \eta^{\text{dual}} = 1.9 & \sigma = 10^{-2} & \tau = 0.75 & \end{array}$$

#### 2.4.5.3 Convergence

La preuve de convergence de l'Algorithme 3 dans le cas strictement convexe est donnée dans [Armand *et al.*, 2000] :

**Théorème 2.2.** *La suite de variables primales $\boldsymbol{x}^k$ générée par l'algorithme 3 converge vers la solution du problème* (2.67) *lorsque $k \to +\infty$.*

#### 2.4.5.4 Application à l'estimation des cartes d'abondances

**Pour un pixel**

Commençons par montrer comment la méthode IPLS peut être utilisée pour estimer les abondances dans un seul pixel. Le problème s'écrit

$$\begin{aligned} &\underset{\boldsymbol{a}\in\mathbb{R}^P}{\text{minimiser}}\ F(\boldsymbol{a}) = \|\boldsymbol{y}-\boldsymbol{S}\boldsymbol{a}\|_2^2, \\ &\text{sous les contraintes } NN \text{ ou } NN+SLO \text{ ou } NN+STO. \end{aligned} \tag{2.96}$$

Dans le cas où la contrainte STO est utilisée, le problème (2.96) est transformé en un nouveau problème faisant apparaître des contraintes d'inégalité uniquement à l'aide d'un changement de variable. Pour tout vecteur initial $\boldsymbol{a}^{(0)} \in \mathbb{R}^P$ vérifiant la contrainte STO, le vecteur défini par $\boldsymbol{a} = \boldsymbol{a}^{(0)} + \boldsymbol{Z}\boldsymbol{c}$, avec $\boldsymbol{c} \in \mathbb{R}^{P-1}$, satisfait également cette contrainte si $\boldsymbol{Z} \in \mathbb{R}^{P\times P-1}$ est une matrice dont les colonnes forment l'espace nul de $\mathbf{1}_{1\times P}$. Par exemple, la matrice définie par

$$Z_{ij} = \begin{cases} 1 & \text{si} \quad i=j, \\ -1 & \text{si} \quad i=j+1, \\ 0 & \text{sinon}, \end{cases} \tag{2.97}$$

satisfait cette condition.

Le problème (2.96), lorsqu'il est soumis aux contraintes NN+STO, peut donc s'écrire sous la forme

$$\begin{aligned} \underset{\boldsymbol{c}\in\mathbb{R}^{P-1}}{\text{minimiser}} \quad & F(\boldsymbol{a}^{(0)}+\boldsymbol{Z}\boldsymbol{c}), \\ \text{sous les contraintes} \quad & \boldsymbol{a}^{(0)}+\boldsymbol{Z}\boldsymbol{c} \geqslant 0. \end{aligned} \tag{2.98}$$

D'une façon générale, les trois types de contraintes considérées dans cette étude peuvent être représentés par une unique formulation du problème :

$$\begin{aligned} \underset{\boldsymbol{c}\in\mathbb{R}^{n_p}}{\text{minimiser}} \quad & F(\boldsymbol{a}^{(0)}+\boldsymbol{Z}\boldsymbol{c}), \\ \text{sous les contraintes} \quad & \boldsymbol{T}_1\boldsymbol{Z}\boldsymbol{c}+\boldsymbol{T}_1\boldsymbol{a}^{(0)}+\boldsymbol{t}_0 \geqslant 0, \end{aligned} \tag{2.99}$$

où $n_p$ est le nombre de variables primales, $n_d$ est le nombre de variables duales, $\boldsymbol{a}^{(0)} \in \mathbb{R}^P$, $\boldsymbol{Z} \in \mathbb{R}^{P\times n_p}$, $\boldsymbol{T}_1 \in \mathbb{R}^{n_d\times P}$, $\boldsymbol{t}_0 \in \mathbb{R}^{n_d}$, avec les définitions adaptées à chaque type de contrainte d'après le tableau 2.1.

Le problème (2.99) peut être reformulé plus simplement :

$$\begin{aligned} \underset{\boldsymbol{c}\in\mathbb{R}^{n_p}}{\text{minimiser}} \quad & \Phi(\boldsymbol{c}), \\ \text{sous les contraintes} \quad & \boldsymbol{T}\boldsymbol{c}+\boldsymbol{t} \geqslant 0, \end{aligned} \tag{2.100}$$

avec

| NN | NN + SLO | NN + STO |
|---|---|---|
| $n_p = P$ | $n_p = P$ | $n_p = P - 1$ |
| $n_d = P$ | $n_d = P + 1$ | $n_d = P$ |
| $\boldsymbol{a}^{(0)} = \mathbf{0}_P$ | $\boldsymbol{a}^{(0)} = \mathbf{0}_P$ | $\boldsymbol{a}^{(0)} = \frac{1}{P}\mathbf{1}_P$ |
| $\boldsymbol{Z} = \boldsymbol{I}_P$ | $\boldsymbol{Z} = \boldsymbol{I}_P$ | $Z_{ij} = \begin{cases} 1 & \text{if } i = j \\ -1 & \text{if } i = j + 1 \\ 0 & \text{otherwise} \end{cases}$ |
| $\boldsymbol{T}_1 = \boldsymbol{I}_P$ | $\boldsymbol{T}_1 = \begin{bmatrix} \boldsymbol{I}_P \\ -\mathbf{1}_{1\times P} \end{bmatrix}$ | $\boldsymbol{T}_1 = \boldsymbol{I}_P$ |
| $\boldsymbol{t}_0 = \mathbf{0}_P$ | $\boldsymbol{t}_0 = \begin{bmatrix} \mathbf{0}_P \\ -1 \end{bmatrix}$ | $\boldsymbol{t}_0 = \mathbf{0}_P$ |

TABLE 2.1 – Définitions des paramètres du problème 2.99 en fonction des contraintes utilisées.

$$\begin{aligned} \Phi(\boldsymbol{c}) &= F(\boldsymbol{a}^{(0)} + \boldsymbol{Z}\boldsymbol{c}), \\ \boldsymbol{T} &= \boldsymbol{T}_1\boldsymbol{Z}, \\ \boldsymbol{t} &= \boldsymbol{T}_1\boldsymbol{a}^{(0)} + \boldsymbol{t}_0. \end{aligned}$$

Ainsi défini, le problème (2.100) peut être résolu par la méthode IPLS. Un changement de variable ayant été effectué, il est nécessaire de convertir le résultat $\hat{\boldsymbol{c}}$ de la méthode IPLS en vecteur d'abondances :

$$\hat{\boldsymbol{a}} = \boldsymbol{a}^{(0)} + \boldsymbol{Z}\hat{\boldsymbol{c}}. \tag{2.101}$$

**Pour une image entière avec pénalisation spatiale**

On considère maintenant le problème initial (2.15) portant sur $N$ pixels avec régularisation spatiale. En effectuant le même changement de variable et en vectorisant le problème, il peut également s´écrire sous la forme

$$\begin{aligned} \underset{\boldsymbol{c}\in\mathbb{R}^{n_p N}}{\text{minimiser}} \quad & \Phi(\boldsymbol{c}), \\ \text{sous les contraintes} \quad & \boldsymbol{T}\boldsymbol{c} + \boldsymbol{t} \geqslant 0, \end{aligned} \tag{2.102}$$

avec les définitions suivantes :

$$\begin{aligned} \Phi(\boldsymbol{c}) &= F(\boldsymbol{A}^{(0)} + \boldsymbol{Z}\ \mathrm{mat}(\boldsymbol{c})) \\ &= \frac{1}{2}\left\|\mathrm{vect}(\boldsymbol{Y} - \boldsymbol{S}\boldsymbol{A}^{(0)}) + (\boldsymbol{I}_N \otimes \boldsymbol{S}\boldsymbol{Z})\boldsymbol{c}\right\|_2^2 + \beta R(\mathrm{vect}(\boldsymbol{A}^{(0)}) + (\boldsymbol{I}_N \otimes \boldsymbol{Z})\boldsymbol{c}), \\ \boldsymbol{A}^{(0)} &= [\boldsymbol{a}_1, \ldots, \boldsymbol{a}_N], \\ \boldsymbol{T} &= \boldsymbol{I}_N \otimes \boldsymbol{T}_1\boldsymbol{Z}, \\ \boldsymbol{t} &= \mathrm{vect}(\boldsymbol{T}_1\boldsymbol{A}^{(0)} + \boldsymbol{T}_0), \\ \boldsymbol{T}_0 &= [\boldsymbol{t}_{01}, \ldots, \boldsymbol{t}_{0N}]. \end{aligned}$$

Le problème complet peut donc également être résolu par la méthode IPLS. Les abondances sont calculées à partir du résultat l'algorithme 3 par :

$$\hat{\boldsymbol{A}} = \boldsymbol{A}^{(0)} + \boldsymbol{Z}\,\mathrm{mat}(\hat{\boldsymbol{c}}). \tag{2.103}$$

**Remarques**

Bien que la méthode IPLS puisse être utilisée de la même façon avec ou sans pénalisation spatiale, il sera observé au chapitre 3 puis montré au chapitre 4 que la prise en compte de la pénalisation augmente significativement la complexité des calculs effectués. Des modifications algorithmiques seront alors proposées pour une résolution efficace du problème pénalisé.

La méthode IPLS est une méthode de second ordre faisant intervenir le Hessien du critère lors du calcul des directions primales. Comme toute méthode de second ordre, elle est adaptée à des problèmes de taille modeste. Dans le cadre de cette thèse, elle est employée pour la résolution d'un problème de grande taille. Néanmoins il sera observé au prochain chapitre que ses performances sont comparables aux autres méthodes étudiées, et les chapitres suivant seront consacrés à son accélération à travers une nouvelle façon de calculer les directions primales.

## 2.5 Conclusion

Ce chapitre a présenté les hypothèses permettant d'écrire le problème d'estimation des cartes d'abondances sous la forme d'un problème d'optimisation convexe sous contraintes. Quatre méthodes permettant de résoudre ce problème ont été décrites : deux méthodes de contraintes actives (NNLS et FCLS), une méthode de Lagrangien augmenté (ADMM) et une méthode de points intérieurs (IPLS).

Le chapitre 3 propose une analyse comparative de ces méthodees d'optimisation à travers un grand nombre de tests réalisés à partir d'images hyperspectrales simulées. Les chapitres suivant s'intéressent à l'accélération de la méthode IPLS.

# Chapitre 3

# Analyse comparative des méthodes existantes

## Sommaire



Ce chapitre propose une comparaison des performances des quatre méthodes décrites au chapitre précédent : NNLS, FCLS, ADMM, et IPLS. Pour cela, des cartes d'abondances sont générées et sont utilisées pour mélanger des spectres de matériaux réels. Chaque image hyperspectrale ainsi générée est ensuite traitée par les différentes méthodes. La comparaison porte sur le temps de calcul et la qualité de l'estimation des cartes. De nombreux tests sont réalisés afin d'observer l'influence de chaque paramètre sur les performances des algorithmes étudiés.

Ce travail a été en partie intégré aux publications [Chouzenoux *et al.*, 2013] et [Chouzenoux *et al.*, 2014] mentionnées dans le chapitre 1.

## 3.1 Protocole expérimental

L'étude porte sur l'influence des paramètres suivants sur les performances des trois méthodes :

- Le nombre de pixels $N$
- Le nombre de spectres purs $P$

- Le niveau de bruit, représenté par le rapport signal sur bruit RSB

$$\text{RSB} = 10\log_{10}\left(\frac{\text{var}(\boldsymbol{y}_n)}{\text{var}(\boldsymbol{e}_n)}\right), \tag{3.1}$$

  où $\text{var}(\boldsymbol{x})$ est la variance de $\boldsymbol{x}$.

- Le type de spectre pur utilisé lors de l'estimation des abondances : exact ou estimé par la méthode NFINDR [Winter, 1999]
- Le type de contrainte : NN, NN+SLO ou NN+STO
- La présence ou non de pénalisation spatiale, et la valeur du paramètre de pénalisation.

Pour chaque test réalisé, c'est-à-dire pour chaque jeu de paramètres, le protocole suivant est employé :

1. **Génération de la matrice des spectres purs :** La matrice $\boldsymbol{S}$ est obtenue en sélectionnant aléatoirement $P$ spectres parmi les 498 que compte la bibliothèque spectrale USGS (*United States Geological Survey*) [Clark *et al.*, 1993]. Chacun de ces spectres contient $L = 224$ bandes spectrales allant de 383 nm à 2508 nm. La figure 3.1 illustre un exemple de $P = 10$ spectres choisis aléatoirement. On peut voir que certains de ces spectres sont fortement corrélés sans pour autant êtres identiques.
2. **Génération de la matrice des abondances :** La matrice $\boldsymbol{A}$ est composée de $P$ cartes d'abondances simulées de façon à contenir trente motifs gaussiens, de variance égale à $N/200$, positionnés aléatoirement au sein de l'image et normalisés pour satisfaire la contrainte STO. Afin de simuler la présence de frontières nettes au sein de l'image, les abondances dont le niveau est inférieur au niveau moyen de $1/P$ sont divisées par un facteur 10. Le résultat de cette opération est normalisé pour satisfaire à nouveau la contrainte STO. La figure 3.2 illustre un exemple de cartes d'abondances générées pour $P = 10$ et $N = 100^2$. Nous pouvons constater que cette stratégie de simulation permet d'obtenir des cartes dans lesquelles il n'existe pas de pixel pur et également d'obtenir une régularité spatiale de type homogène par morceaux, comme dans les hypothèses de la partie 2.2.
3. **Formation de l'image hyperspectrale :** L'image hyperspectrale simulée $\boldsymbol{Y}$ est formée selon le modèle de mélange linéaire $\boldsymbol{Y} = \boldsymbol{SA} + \boldsymbol{E}$. Le terme $\boldsymbol{E} \in \mathbb{R}^{L\times N}$ est un bruit gaussien i.i.d. de moyenne nulle et de variance adéquate pour obtenir le rapport signal sur bruit (RSB) désiré. La figure 3.3 illustre l'image hyperspectrale formée à partir des spectres de la figure 3.1, des cartes d'abondances de la figure 3.2, et d'un bruit tel que $RSB = 10$ dB. Afin de représenter en deux dimensions un tel jeu de données, seuls l'image correspondant à la première bande spectrale (383 nm) et le spectre correspondant au premier pixel sont représentés.
4. **Estimation des cartes d'abondances :** Les différentes méthodes décrites dans le chapitre précédent sont employées pour estimer les cartes d'abondances à partir de l'image hyperspectrale. Pour chaque méthode, le temps de calcul est mesuré, et les cartes obtenues sont comparées aux cartes générées, représentant la « vérité ».
5. **Simulation de Monte Carlo :** Afin de s'affranchir des variations liées au caractère aléatoire d'une simulation, l'ensemble des étapes précédentes est répété 100 fois. Le temps de calcul moyen et l'erreur d'estimation moyenne sont retenus.

Les critères suivants sont utilisés pour comparer les méthodes :

1. **Temps de calcul :** Pour un jeu de paramètres donné, les méthodes convergent toutes vers la solution du même problème, elles fournissent donc un résultat autour de la même solution, avec une certaine tolérance. Ainsi, l'unique critère permettant de juger des performances d'une méthode est le temps nécessaire pour parvenir au résultat.
2. **Nombre d'itérations :** Il permet de mieux comprendre les différences entre les méthodes étudiées. Pour FCLS, il s'agit du nombre moyen d'itérations par pixel car l'algorithme est exécuté indépendamment sur chaque pixel. Pour IPLS, il s'agit du cumul des itérations de la méthode de Newton utilisée pour résoudre la séquence d'équations KKT perturbées.
3. **Erreur d'estimation ou de reconstruction :** Cette erreur est mesurée par l'EQMN (*Erreur Quadratique Moyenne Normalisée*) qui mesure la différence relative moyenne entre les abondances simulées $\boldsymbol{a}_p = [A_{p1}, \ldots, A_{pN}]^{\mathrm{t}}$ et les abondances estimées $\hat{\boldsymbol{a}}_p$

$$\mathrm{EQMN}(\%) = \frac{100}{P} \sum_{p=1}^{P} \frac{\|\boldsymbol{a}_p - \hat{\boldsymbol{a}}_p\|^2}{\|\boldsymbol{a}_p\|^2}. \tag{3.2}$$

   Ce critère permet de comparer entre eux les différents jeux de paramètres testés.
4. **Résidu :** Le résidu moyen $r$ est donné pour illustrer l'équivalence des différentes méthodes dans une même situation en termes de résultats. Il est calculé ainsi

$$r = \|\boldsymbol{Y} - \boldsymbol{S}\boldsymbol{A}\|_F^2 . \tag{3.3}$$

Le nombre de paramètres étudiés dans cette comparaison étant élevé, il est impossible de tester exhaustivement toutes les combinaisons de paramètres, mêmes restreints à des plages de valeurs discrètes et bornées. Dans un premier temps, chaque paramètre est fixé à une valeur de référence, et les différentes méthodes sont comparées dans cette situation de référence. Puis, l'influence de chaque paramètre sur les performances des méthodes est étudiée séparément, en faisant varier sa valeur autour de la valeur de référence.

**Matériel utilisé**

Tous les calculs présentés dans cette thèse sont effectués sur une station de travail Dell Precision T7400 contenant un processeur Intel Xeon X5472 cadencé à 3 GHz et 16 GO de mémoire RAM. Cette station est également équipée d'une carte graphique Tesla C1060 qui sera exploitée au chapitre 5.

## 3.2 Situation de référence

Les valeurs de référence des différents paramètres étudiés sont données par le tableau 3.1.

Les résultats des trois méthodes étudiées (NNLS n'est pas utilisée car la contrainte STO est considérée) sont donnés dans le tableau 3.2. On observe que, dans ces conditions, les temps de calcul sont relativement proches, avec un avantage pour IPLS, et un temps plus long pour ADMM. Mais cette proximité cache d'importantes différences internes. Ainsi, le nombre d'itérations nécessaires à FCLS pour atteindre la convergence est très faible, mais chaque itération est coûteuse en temps de calcul. Au contraire, ADMM nécessite un grand nombre d'itérations qui sont exécutées rapidement. Enfin, les 3 méthodes minimisent le même critère, il est donc normal d'obtenir les mêmes valeurs pour le résidu et l'erreur de reconstruction.

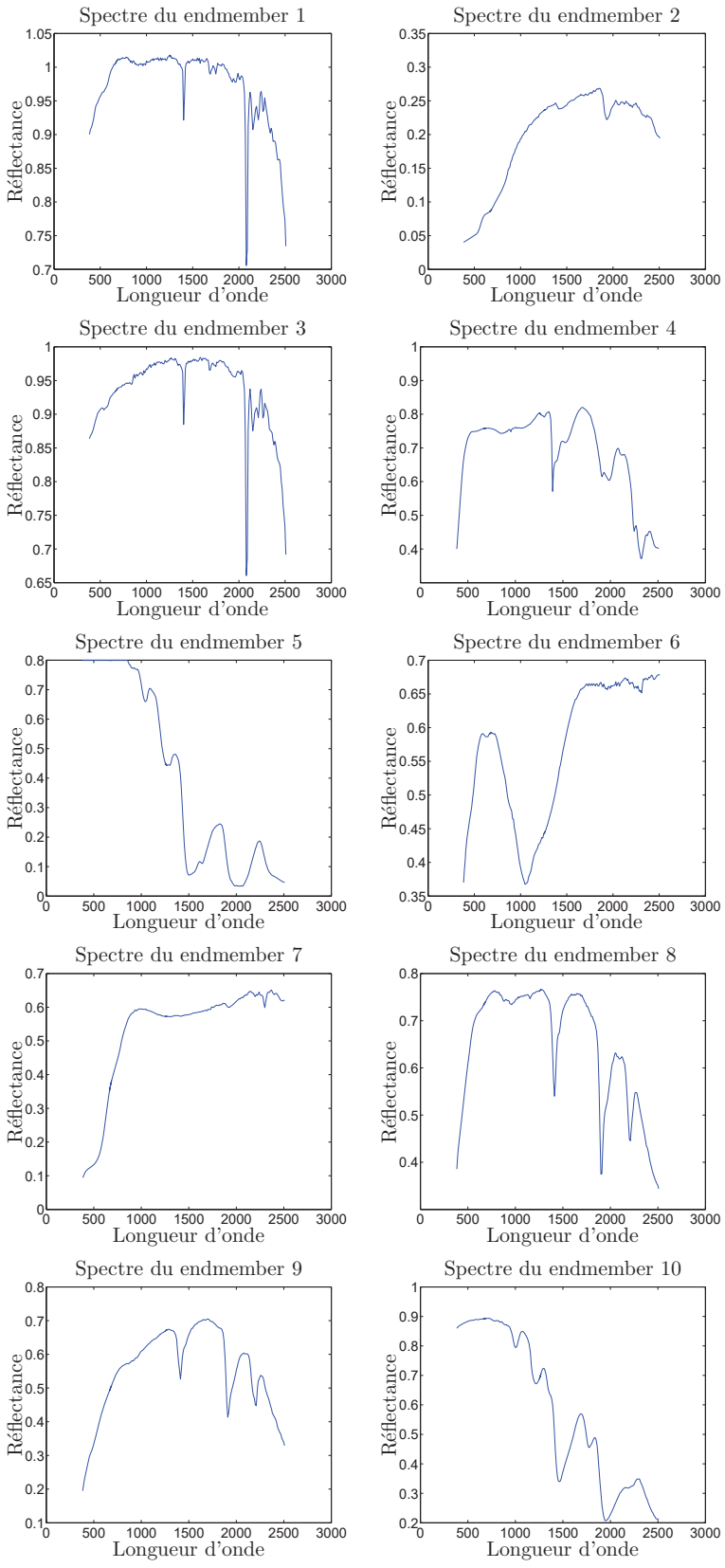



FIGURE 3.1 – Exemple de 10 spectres purs issus de la bibliothèque USGS.

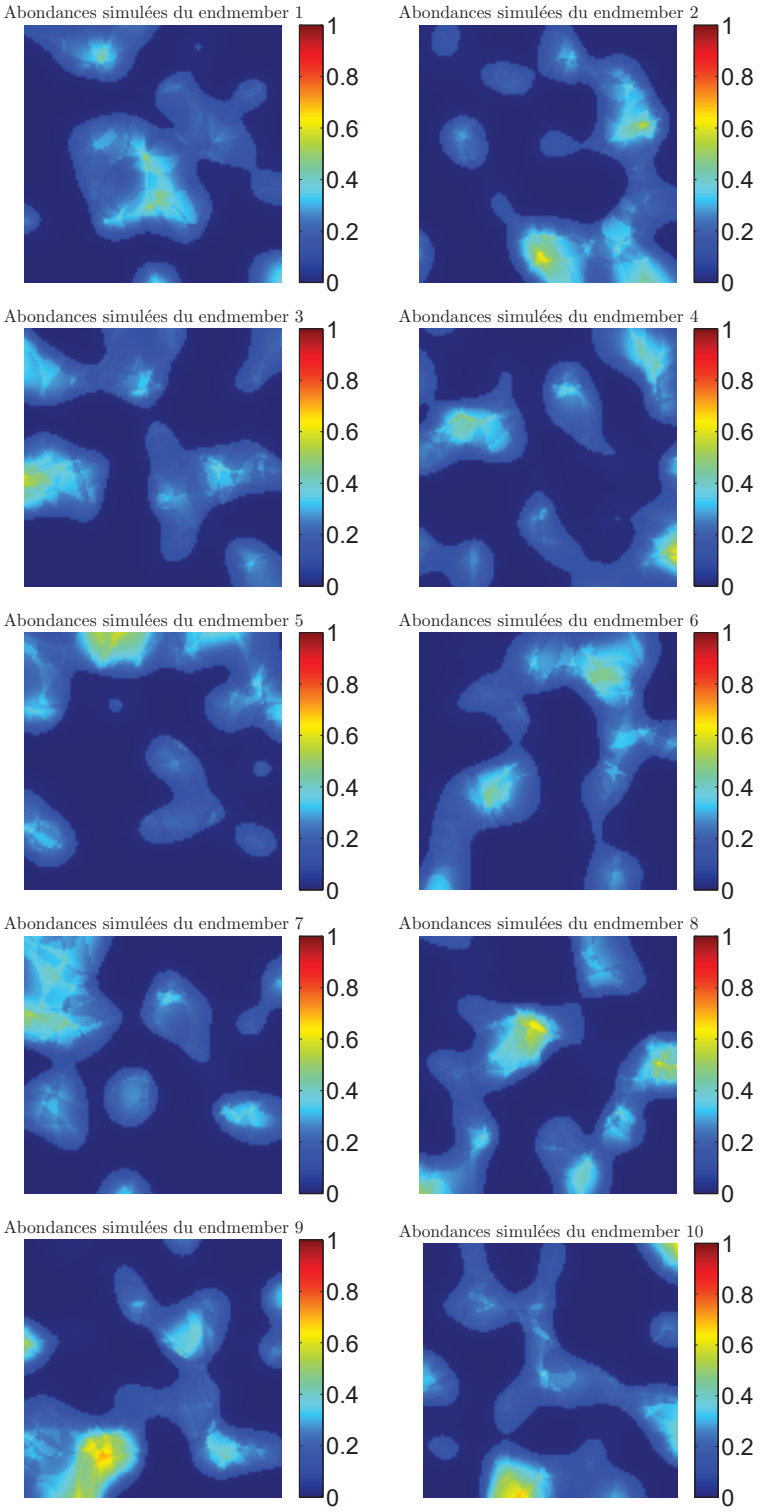



FIGURE 3.2 – Exemple de 10 cartes d'abondances simulées.

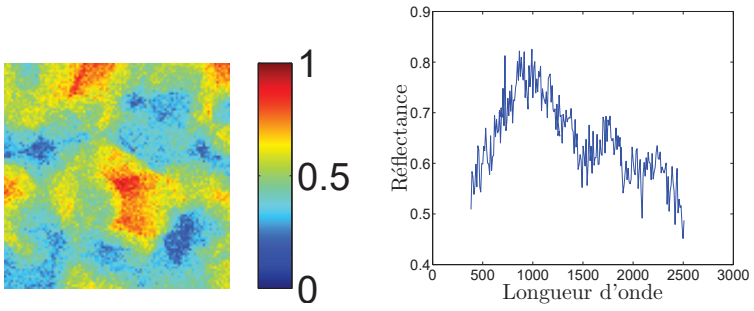



FIGURE 3.3 – Visualisation partielle d'une image hyperspectrale à travers la réflectance mesurée pour une longueur d'onde (à gauche) et pour un pixel (à droite).

| Paramètre | Valeur |
|---|---|
| N | $100^2$ |
| P | 10 |
| SNR | 10 dB |
| type de spectres purs | exacts |
| contrainte | NN+STO |
| pénalisation spatiale | non prise en compte |

TABLE 3.1 – Valeurs des paramètres utilisés dans la situation de référence.

| Méthode | FCLS | ADMM | IPLS |
|---|---|---|---|
| Temps CPU (en s) | 2.11 s | 2.55 s | 1.92 s |
| Nombre d'itérations | 2.12 | 721 | 17.5 |
| Résidu ($\times 10^{-3}$) | 2.13 | 2.13 | 2.13 |
| EQMN | 9.88 % | 9.88 % | 9.88 % |

TABLE 3.2 – Résultats dans la situation de référence.

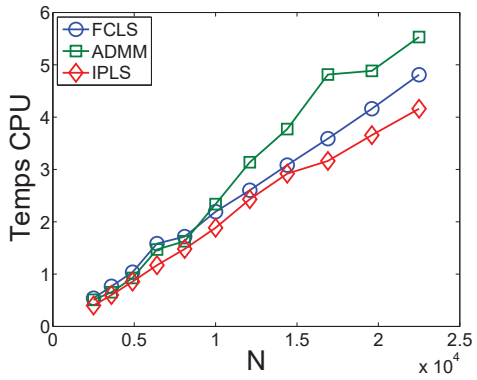



FIGURE 3.4 – Temps de calcul en fonction de la taille de l'image.

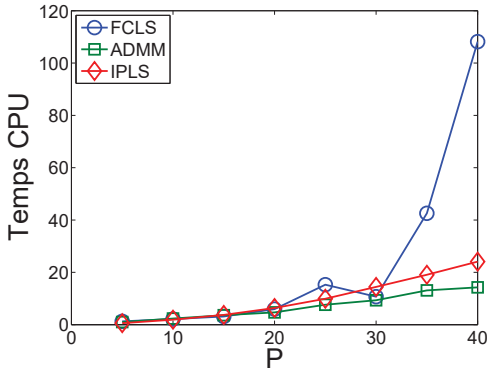



FIGURE 3.5 – Temps de calcul en fonction du nombre de spectres purs.

## 3.3 Influence du nombre de pixels $N$

La figure 3.4 illustre le comportement des 3 méthodes étudiées lorsque le nombre de pixels de l'image varie. Pour toutes les méthodes, le temps de calcul varie proportionnellement au nombre de pixels. Notons que lorsque la taille de l'image augmente, les temps de calcul de FCLS et IPLS augmentent à un rythme comparable, alors que celui de ADMM augmente plus rapidement, ce qui en fait une méthode moins adaptée aux problèmes de grande taille. Les nombres d'itérations relevés restent stables pour chaque méthode quelle que soit la taille du problème, il en est de même pour les résidus et erreurs de reconstruction.

## 3.4 Influence du nombre de spectres purs $P$

La figure 3.5 illustre les temps de calcul des 3 méthodes étudiées lorsque le nombre de spectres purs varie. Lorsque ce nombre augmente, les temps de convergence s'allongent logiquement, mais dans des proportions différentes suivant la méthode. Ainsi, il apparaît que ADMM est la méthode la plus performante pour un grand nombre de spectres purs, suivie de IPLS, alors que le temps de calcul de FCLS devient très élevé. Notons que pour FCLS, ce comportement est dû principalement au nombre d'itérations qui passe de 0.88 pour $P = 5$ à 148 pour $P = 40$, alors que ce critère reste stable pour les autres méthodes (pour FCLS, le nombre d'itération moyen peut être inférieur à 1 car aucune itération n'est comptabilisée lorsque pour un pixel la solution du problème non contraint satisfait les contraintes).

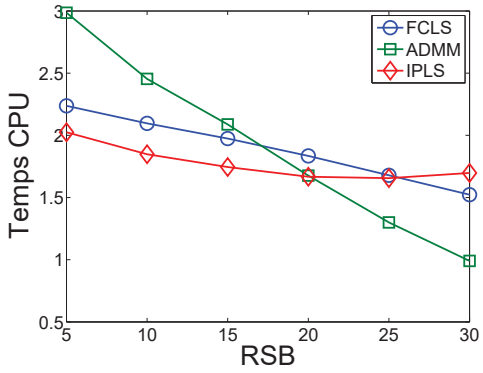



FIGURE 3.6 – Temps de calcul en fonction du bruit de mesure.

## 3.5 Robustesse face au bruit

La figure 3.6 illustre le comportement des 3 méthodes étudiées lorsque le bruit de mesure varie. On observe que les temps de calcul se réduisent lorsque le bruit de mesure diminue. Cette variation est plus marquée pour la méthode ADMM, qui est la plus rapide pour un bruit faible, mais la plus lente pour un bruit important. ADMM est donc la méthode la moins robuste face au bruit de mesure, alors que IPLS est la plus robuste. La taille des données reste constante au cours de ce test, ces variations sont donc expliquées intégralement par le nombre d'itérations qui se réduit pour des bruits faibles. Notons que le résidu et l'erreur de reconstruction diminuent également.

## 3.6 Robustesse face à l'imprécision des spectres purs

Comme évoqué au chapitre 2, l'étape de détermination des spectres purs précédant l'estimation des abondances peut se faire de deux façons : soit en sélectionnant des spectres de matériaux connus dans une bibliothèque spectrale, soit en effectuant une estimation des spectres purs à partir de l'image hyperspectrale elle-même. Dans les tests effectués jusqu'à présent, des images hyperspectrales ont été simulées en utilisant des spectres issus de la bibliothèque spectrale USGS. Ces mêmes spectres ont ensuite été utilisés lors de l'estimation des abondances. Dans le test dont les résultats sont présentés dans le tableau 3.3, cette situation est comparée à l'estimation des abondances après estimation des spectres purs à partir de l'image avec la méthode NFINDR [Plaza et Chang, 2005]. Cette méthode sélectionne dans l'image les pixels considérés comme purs, dans le sens où les spectres de ces pixels présentent la plus faible corrélation possible. Les spectres purs ainsi obtenus sont proches des spectres exacts utilisés pour simuler l'image, mais différents de ceux-ci à cause du bruit gaussien ajouté lors de la simulation, mais aussi car la méthode de simulation des abondances assure qu'aucun pixel de l'image simulée ne peut être pur.

Dans le tableau 3.3, on observe logiquement un accroissement du résidu lorsque les spectres purs estimés sont utilisés. Il est intéressant de voir que le temps de calcul est également impacté par l'imprécision des spectres purs. Pour chaque méthode, celui-ci augmente lorsque les spectres purs estimés sont utilisés. Ce ralentissement affecte d'avantage la méthode IPLS dont le temps de calcul augmente de 58 %. En revanche, FCLS est la méthode la plus robuste face à l'imprécision des spectres purs car son temps de calcul n'augmente que de 25 %.

| Méthode | FCLS | | ADMM | | IPLS | |
|---|---|---|---|---|---|---|
| spectres purs | exacts | estimés | exacts | estimés | exacts | estimés |
| Temps CPU (en s) | 2.11 | 2.63 (+25%) | 2.55 | 3.42 (+34%) | 1.92 | 3.03 (+58%) |
| Nombre d'itérations | 2.12 | 3.13 | 721 | 951 | 17.5 | 28.8 |
| Résidu ($\times 10^{-3}$) | 2.13 | 2.37 | 2.13 | 2.37 | 2.13 | 2.37 |

TABLE 3.3 – Comparaison des résultats avec spectres purs exacts et estimés par la méthode NFINDR.

| Méthode | NNLS | FCLS | ADMM | | IPLS | | |
|---|---|---|---|---|---|---|---|
| Contrainte | NN | NN + STO | NN | NN + STO | NN | NN + SLO | NN + STO |
| Temps CPU (en s) | 3.24 | 2.11 | 1.92 | 2.55 | 1.91 | 2.63 | 1.92 |
| Nombre d'itérations | 7.96 | 2.12 | 591 | 721 | 16.6 | 18.2 | 17.5 |
| Résidu ($\times 10^{-3}$) | 2.13 | 2.13 | 2.13 | 2.13 | 2.13 | 2.13 | 2.13 |
| NMSE | 221 % | 9.88 % | 91.3 % | 9.88 % | 109 % | 11.1 % | 9.88 % |

TABLE 3.4 – Comparaison des résultats avec les 3 types de contraintes étudiés : NN, SLO et STO.

## 3.7 Influence du type de contrainte utilisé

Les tests précédents ont été effectués en ne considérant que les contraintes NN et STO, c'est-à-dire de non-négativité et de somme égale à un. Or, la contrainte NN (non-négativité seule) est mise en œuvre par les méthodes NNLS et IPLS, mais aussi ADMM qui le propose en option dans son implémentation Matlab, l'algorithme correspondant peut se construire avec le même raisonnement que pour la contrainte NN+STO. Quant à la contrainte SLO, elle n'est implémentée que par la méthode IPLS, bien qu'elle puisse théoriquement être gérée par une méthode de type ADMM.

Le tableau 3.4 donne les performances des quatre méthodes étudiées en fonction du type de contraintes choisies. On y voit le résidu rester constant pour tous les types de contraintes. Avec la contrainte NN uniquement, les erreurs de reconstructions obtenues sont très élevées. De plus, ces erreurs sont différentes pour chaque méthode alors que le résidu est le même. C'est le signe d'un problème mal conditionné pour lequel des propositions éloignées de la solution fournissent un résidu proche du résidu optimal. L'ajout de la contrainte SLO ou STO permet de diminuer considérablement l'erreur de reconstruction. Ceci montre l'intérêt de considérer un maximum d'information *a priori*. Rappelons que l'annexe A montre que les contraintes NN+SLO peuvent se révéler être le meilleur choix lorsque la bibliothèque spectrale utilisée est incomplète. Enfin, les conséquences du choix de la contrainte sur le temps de calcul sont différentes pour chaque méthode. La durée d'exécution de la méthode NNLS s'explique par un nombre d'itérations plus important que pour la méthode FCLS. Pour ADMM, des simplification sont possibles dans le cas où la contrainte de somme égale à un n'est pas utilisée, ce qui accélère légèrement les calculs. Enfin pour IPLS, le nombre de variables varie en fonction des contraintes, comme il a été vu dans la partie 2.4.5. Dans les conditions de ce test, il apparaît donc que les méthodes ADMM et IPLS sont les plus rapides pour la contrainte NN, la méthode IPLS est la plus rapide pour les contraintes NN+STO, c'est également la seule à prendre en charge les contraintes NN+SLO.

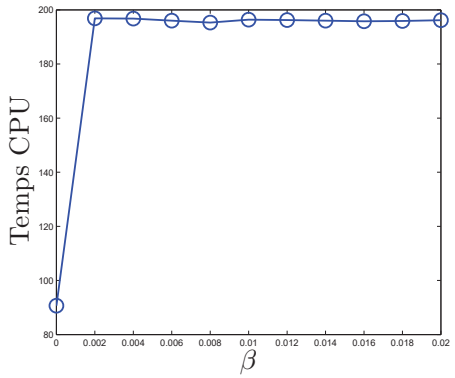



FIGURE 3.7 – Temps de calcul de ADMM en fonction du paramètre de régularisation.

## 3.8 Influence de la pénalisation spatiale

La pénalisation spatiale est disponible uniquement pour les méthodes ADMM et IPLS. Pour la méthode ADMM, une pénalisation par variation totale, c'est-à-dire une pénalisation spatiale dont la fonction de pondération est de type $\ell_1$, est disponible à travers l'algorithme SUnSAL-TV [Iordache *et al.*, 2012]. Ainsi, seul le paramètre de régularisation $\beta$ définit la pénalisation pour cette méthode. Pour la méthode IPLS, deux types de pénalisations sont envisagés : la pénalisation de type $\ell_2$ qui ne fait intervenir que le paramètre de régularisation $\beta$, ou la pénalisation de type $\ell_2 - \ell_1$ qui fait intervenir $\beta$ et le paramètre $\delta$ définissant le seuil entre le comportement $\ell_2$ et le comportement $\ell_1$.

Dans un premier temps, il convient de choisir les paramètres de pénalisation pour chacun de ces cas. Comme nous travaillons avec des images simulées, nous pouvons choisir les paramètres qui minimisent l'erreur de reconstruction. Ensuite, une comparaison avec la situation de référence non-pénalisée sera effectuée.

### 3.8.1 Choix des paramètres de pénalisation

**Choix de $\beta$ pour ADMM**

Les conditions de référence décrites dans la partie 3.2 sont à nouveau utilisées, à l'exception de la partie pénalisation spatiale qui est à présent étudiée. Les figures 3.7 à 3.9 présentent les performances de la méthode ADMM avec une pénalisation de type $\ell_1$ lorsque le paramètre de régularisation varie. On peut voir sur la figure 3.8 que la valeur $\hat{\beta} = 7 \times 10^{-3}$ permet d'obtenir l'erreur de reconstruction minimale de 1.29 %, contre une erreur de 9.88 % sans pénalisation. Ce gain significatif montre l'intérêt de l'utilisation de la pénalisation spatiale. La contrepartie de ce gain est le temps de calcul qui comme le montre la figure 3.7 est de 195 s alors qu'il était de 2.35 s dans le cas non-pénalisé. Enfin la figure 3.9 illustre l'augmentation attendue du résidu lorsque le poids de la pénalisation augmente.

**Choix de $\beta$ pour IPLS avec pénalisation $\ell_2$**

Comme pour ADMM, l'influence du paramètre de régularisation sur les performances de la méthode IPLS assortie d'une pénalisation spatiale de type $\ell_2$ est illustrée par les figures 3.10 à 3.12. La valeur optimale du paramètre de régularisation est cette fois $\hat{\beta} = 9$. Cette valeur permet d'obtenir une erreur de reconstruction de 3.43 %, contre 9.88 % sans pénalisation. Cela montre que la pénalisation de type $\ell_2$ améliore le résultat, mais pas autant que la pénalisation de type $\ell_1$

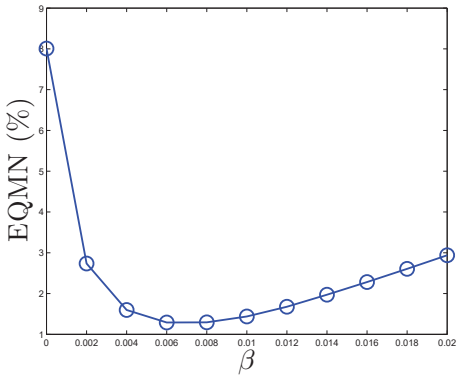



FIGURE 3.8 – Erreur de reconstruction de ADMM en fonction du paramètre de régularisation.

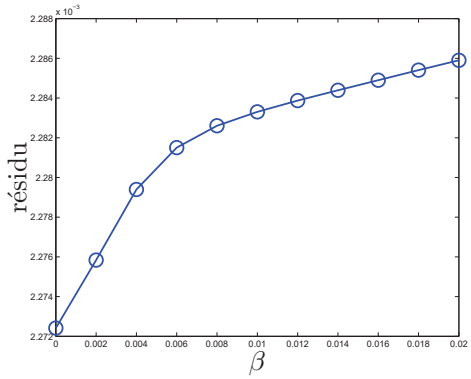



FIGURE 3.9 – Résidu de ADMM en fonction du paramètre de régularisation.

utilisée avec ADMM. Cependant, l'augmentation du temps de calcul est moins importante que pour ADMM puisqu'il passe de 2.34 s à 124 s au lieu de 200 s.

**Choix de $\beta$ et $\delta$ pour IPLS avec pénalisation $\ell_2 - \ell_1$**

Pour la méthode IPLS avec pénalisation de type $\ell_2 - \ell_1$, les paramètres $\beta$ et $\delta$ doivent être déterminés conjointement afin de minimiser l'erreur de reconstruction. Dans le cas des données simulées dans cette étude suivant les hypothèses du chapitre 2, la valeur de $\delta$ conduisant à l'erreur de reconstruction minimale est $\delta = 0$, c'est-à-dire une pénalisation de type $\ell_1$. Notons que dans certaines situations réelles, une valeur de $\delta$ non nulle peut générer la plus faible erreur de reconstruction [Idier, 2013]. Cet exemple ne prétend pas montrer la supériorité de la pénalisation de type $\ell_1$ sur celle de type $\ell_2 - \ell_1$, cette question fait l'objet de recherches actives et la meilleure méthode dépend des cas d'étude. Le choix de $\delta = 0$ étant inaccessible pour la méthode IPLS qui nécessite un paramètre $\delta$ strictement positif, on répète pour différentes valeurs de $\delta$ l'étude menée précédemment pour ADMM et IPLS avec pénalisation $\ell_2$. La figure 3.13 présente les valeurs de $\beta$ minimisant l'erreur de reconstruction pour différentes valeurs de $\delta$, tandis que la figure 3.14 présente l'erreur de reconstruction pour chaque couple $(\delta, \hat{\beta}(\delta))$. Cela montre que plus la valeur de $\delta$ est faible, meilleur est le résultat. Pour les valeurs de $\delta$ les plus faibles, cette erreur remonte brutalement car le temps de convergence est alors trop long et l'algorithme se termine avant d'atteindre la convergence. Ceci est confirmé par la figure 3.15 qui montre que le temps de calcul devient prohibitif pour ces valeurs de $\delta$. Il s'agit donc de trouver un compromis entre une erreur de reconstruction faible, et un temps de calcul raisonnable. Dans le reste de la thèse, le couple $(\hat{\delta} = 0.1; \hat{\beta} = 1)$ sera retenu.

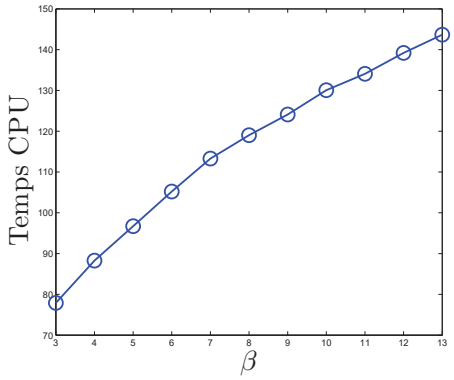



FIGURE 3.10 – Temps de calcul de IPLS avec pénalisation $\ell_2$ en fonction du paramètre de régularisation.

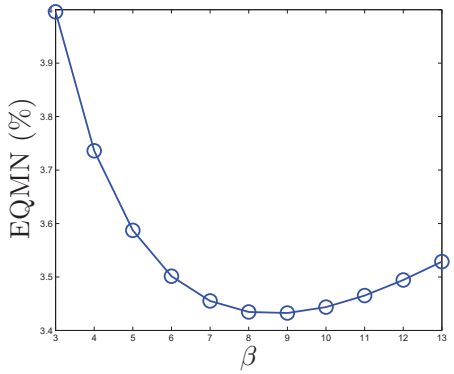



FIGURE 3.11 – Erreur de reconstruction de IPLS avec pénalisation $\ell_2$ en fonction du paramètre de régularisation.

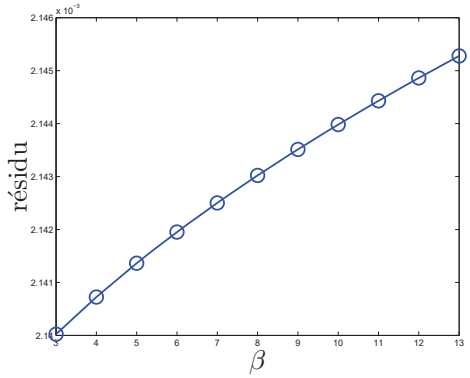



FIGURE 3.12 – Résidu de IPLS avec pénalisation $\ell_2$ en fonction du paramètre de régularisation.

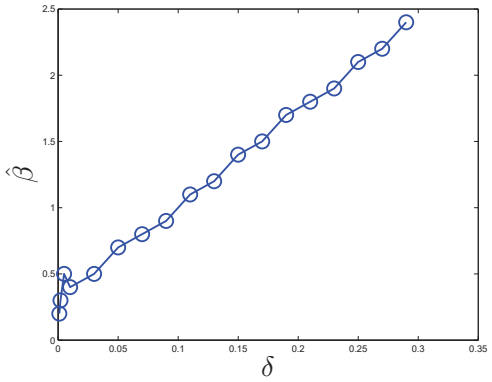



FIGURE 3.13 – Valeurs optimales du paramètre de régularisation en fonction du paramètre de la fonction de pondération $\ell_2 - \ell_1$.

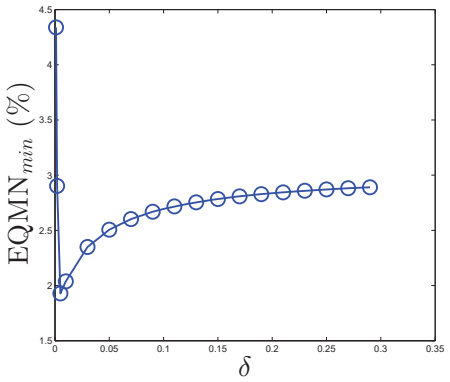



FIGURE 3.14 – Erreur de reconstruction optimale en fonction du paramètre de la fonction de pondération $\ell_2 - \ell_1$.

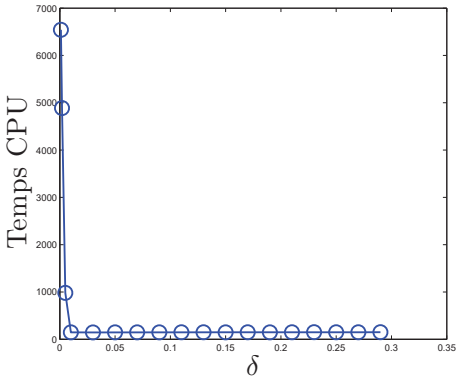



FIGURE 3.15 – Temps de calcul en fonction du paramètre de la fonction de pondération $\ell_2 - \ell_1$.

| Méthode | ADMM | | IPLS | | |
|---|---|---|---|---|---|
| Pénalisation | non-pénalisé | type $\ell_1$ | non-pénalisé | type $\ell_2$ | type $\ell_2 - \ell_1$ |
| Temps CPU (en s) | 2.35 | 196 | 2.34 | 106 | 139 |
| Résidu ($\times 10^{-3}$) | 2.13 | 2.13 | 2.13 | 2.13 | 2.13 |
| EQMN | 9.88 % | 1.29 % | 9.88 % | 3.43 % | 2.69 % |

TABLE 3.5 – Comparaison des résultats avec et sans pénalisation.

### 3.8.2 Situation de référence avec pénalisation

Le tableau 3.5 résume les différences entre le cas non pénalisé et le cas pénalisé pour ADMM et IPLS.

La figure 3.16 illustre les résultats obtenus avec les différents types de pénalisation en reprenant l'exemple présenté en partie 3.1. Le bruit de mesure introduit dans l'image hyperspectrale simulée se retrouve dans les cartes d'abondances estimées sans pénalisation spatiale. Il est nettement atténué avec une pénalisation spatiale de type $\ell_2$. Pour les pénalisations de types $\ell_2 - \ell_1$ et $\ell_1$, les améliorations apportées ne sont pas aussi flagrantes visuellement. Elles sont plus visibles sur la figure 3.17 présentant les erreurs d'estimation des cartes d'abondances $|\boldsymbol{A}_{\text{exact}} - \boldsymbol{A}_{\text{estimé}}|$.

## 3.9 Conclusion

L'analyse effectuée dans ce chapitre montre que chaque méthode possède ses avantages et peut se révéler être la meilleure solution dans certaines conditions. La méthode FCLS est la moins affectée par une imprécision de la matrice des spectres purs. En revanche elle est peu performante lorsque le nombre de endmembers devient élevé et elle ne permet pas la prise en compte d'un *a priori* de régularité spatial. La méthode ADMM est la plus performante pour un nombre de endmembers élevé et permet une pénalisation de type $\ell_1$ pour une meilleure qualité de reconstruction. Par contre son temps de calcul est sensible au bruit de mesure et au nombre de pixels de l'image, il est aussi très élevé lorsque la pénalisation est prise en compte. La méthode IPLS est la moins sensible au nombre de pixels et au bruit de mesure, permet la prise en compte de la contrainte SLO et d'une pénalisation de type $\ell_2$ ou $\ell_2 - \ell_1$ en un temps plus raisonnable de la méthode ADMM, mais son temps de calcul est plus sensible à l'imprécision des endmembers.

La suite de cette thèse se focalise sur la méthode IPLS et propose de réduire son temps de calcul. Des modifications algorithmiques au niveau du calcul des directions primales sont proposées dans le cas pénalisé. Une implémentation efficace sur Matlab puis sur GPU est réalisée avec ou sans pénalisation.

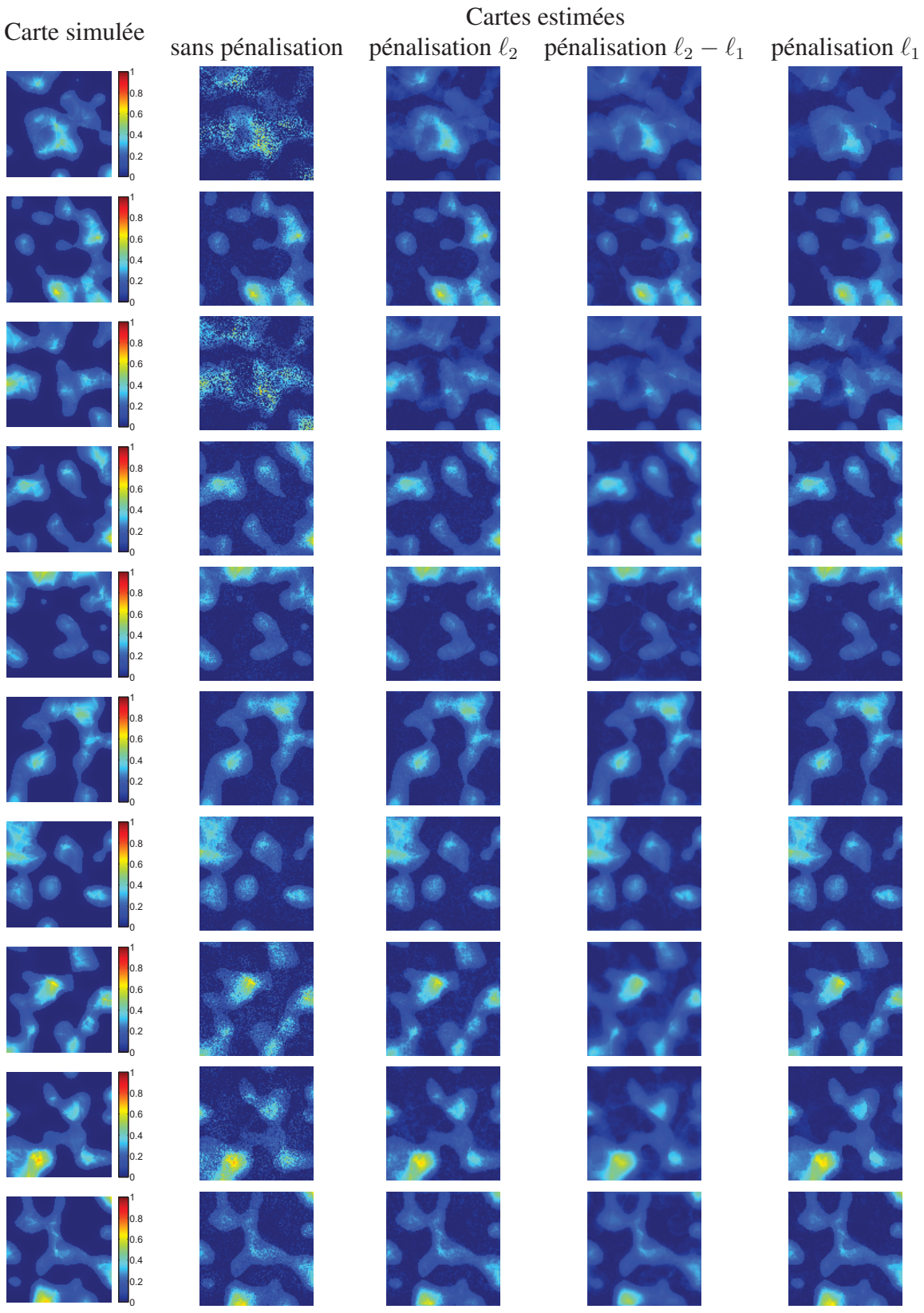



FIGURE 3.16 – cartes d'abondances estimées pour différents types de pénalisation.

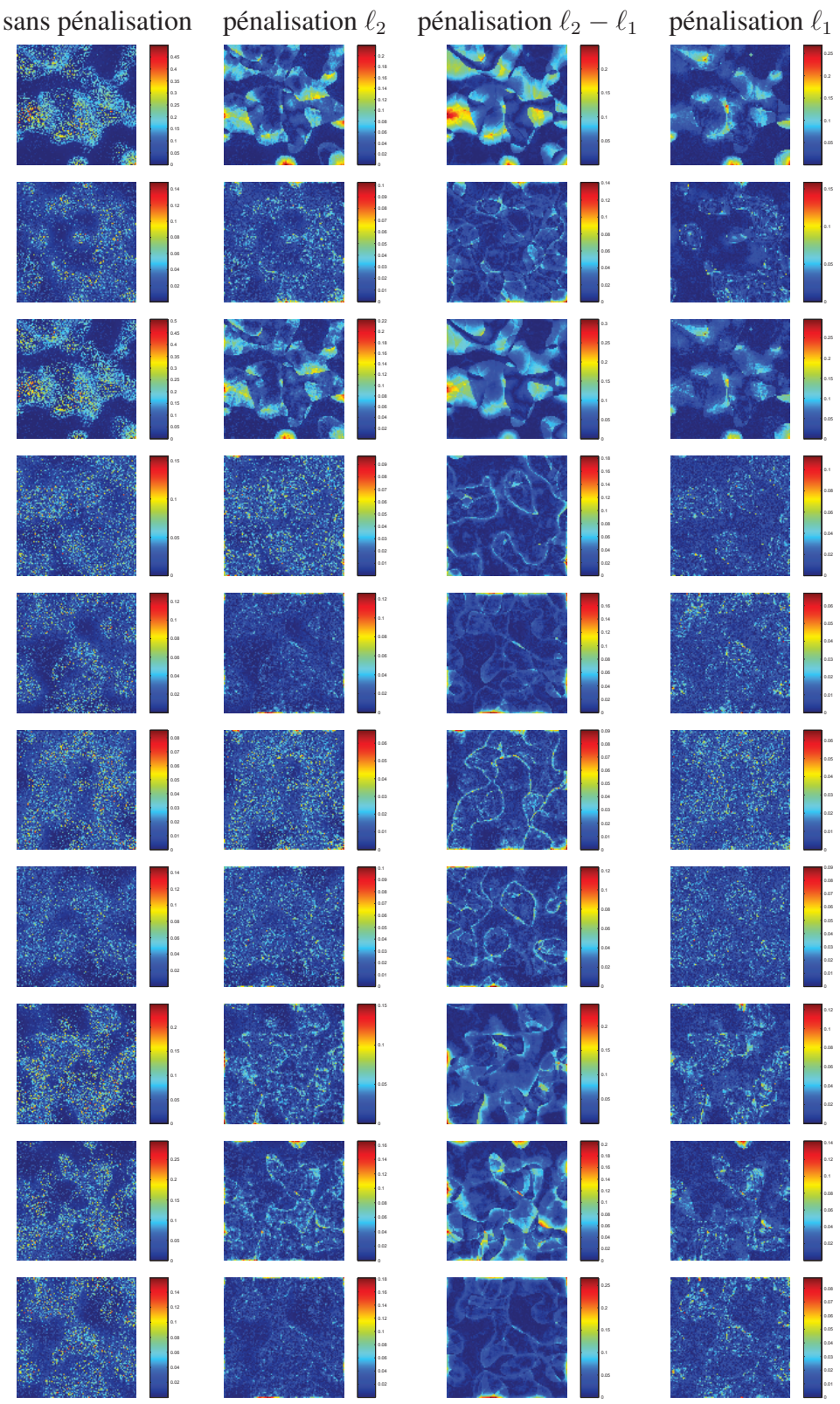



FIGURE 3.17 – Erreurs d'estimation des cartes d'abondances pour différents types de pénalisation.

# Chapitre 4

# Accélération algorithmique de la méthode de points intérieurs

## Sommaire



Nous avons montré que la méthode de points intérieurs IPLS était compétitive par rapport aux autres méthodes existantes pour l'estimation des cartes d'abondances. Néanmoins, le temps de calcul reste un facteur déterminant pour l'utilisation de cette méthode en situation réelle. Une réduction de celui-ci permettrait d'utiliser IPLS à plus grande échelle, afin d'obtenir des cartes d'abondances sur des zones étendues au lieu de se restreindre à certaines zones d'intérêt.

Pour accélérer l'exécution d'un algorithme itératif tel qu'IPLS, deux approches sont envisageables. La première consiste à réduire le nombre d'itérations effectuées. Pour cela, on cherche à raffiner les calculs effectués à l'intérieur d'une itération, ce qui a en général pour effet d'allonger le temps d'exécution de chaque itération. La deuxième approche est l'exact contraire : elle consiste à accélérer le temps d'exécution d'une itération, ce qui a en général pour effet d'augmenter leur nombre. Ainsi, tout est question de compromis entre la durée d'exécution d'une itération et leur nombre. La méthode IPLS étant une méthode du deuxième ordre, elle présente un nombre d'itérations peu important comme montré dans le chapitre 3. Cette observation dirige naturellement cette étude vers la recherche d'une façon plus rapide d'effectuer une itération dans la méthode de points intérieurs.

Dans ce chapitre, l'étape critique de l'algorithme IPLS du point de vue du temps de calcul est identifiée. Il s'agit du calcul des directions primales. Cette étape est alors analysée et il est montré que, dans le cas non pénalisé, elle possède une structure particulière qui peut être exploitée. Cette observation conduit à une meilleure implémentation de l'algorithme pour une exécution plus rapide sans modification du nombre d'itérations. Dans le cas pénalisé, une modification algorithmique est nécessaire avant d'appliquer la même technique. Celle-ci consiste à calculer une approximation des directions primales, ce qui augmente automatiquement le nombre d'itérations totales. Les simulations effectuées permettent de constater que cette modification aboutit globalement à une accélération. A chaque étape de ce travail, une attention particulière est portée sur la facilité à paralléliser l'algorithme ainsi construit, point qui sera exploité dans le chapitre 5.

Les travaux développés dans ce chapitre ont fait l'objet des publications [Chouzenoux *et al.*, 2014] pour la partie non-pénalisée et [Legendre *et al.*, 2014] pour la partie pénalisée.

## 4.1 Identification de l'étape critique

Une itération de l'algorithme IPLS est composée de deux étapes : le calcul des directions primales-duales, et la recherche de pas. Dans la version présentée au chapitre 2, le calcul des directions est lui même séparé en deux : les directions primales sont d'abord calculées, puis les directions duales sont déduites. Considérons une réalisation du cas de référence présenté dans le chapitre 3. L'outil *Profiler* de Matlab permet d'obtenir des informations sur le temps d'exécution de chaque commande. En regroupant les commandes correspondant à chaque étape de l'algorithme, on établit le tableau 4.1 qui présente les temps d'exécutions par étape dans le cas de la situation de référence introduite dans la partie 3.2.

| Etape | Temps d'exécution | Part du temps d'exécution total |
|---|---|---|
| Initialisation | 0.19 s | 9.90 % |
| Calcul de la direction primale | 1.20 s | 62.5 % |
| Calcul de la direction duale | 0.05 s | 2.70 % |
| Recherche de pas | 0.37 s | 19.3 % |
| Vérifications des conditioins d'arrêt | 0.11 s | 5.73 % |
| Total | 1.92 s | 100 % |

TABLE 4.1 – Temps de calcul pour chaque étape de l'algorithme IPLS.

Avec 62.5 % du temps d'exécution total passé à calculer les directions primales, une réduction significative du temps de calcul passe nécessairement par une étude approfondie de cette étape de l'algorithme.

Notons qu'il n'est pas étonnant que cette étape soit la plus coûteuse. En effet, elle consiste à inverser le système linéaire

$$\boldsymbol{H}_{k,i}\boldsymbol{d}^{\boldsymbol{c}}_{k,i} = -\boldsymbol{g}_{k,i}, \tag{4.1}$$

avec

$$\boldsymbol{H}_{k,i} = \left[\nabla^2\Phi(\boldsymbol{c}_{k,i}) + \boldsymbol{T}^{\mathrm{t}}\mathrm{Diag}(\boldsymbol{T}\boldsymbol{c}_{k,i}+\boldsymbol{t})^{-1}\boldsymbol{\Lambda}_{k,i}\boldsymbol{T}\right], \tag{4.2}$$

$$\boldsymbol{g}_{k,i} = \nabla\Phi(\boldsymbol{c}_{k,i}) + \boldsymbol{T}^{\mathrm{t}}\mathrm{Diag}(\boldsymbol{T}\boldsymbol{c}_{k,i}+\boldsymbol{t})^{-1}\boldsymbol{\mu}_k, \tag{4.3}$$

où $\boldsymbol{H}_{k,i}$ est une matrice de taille $(n_p N \times n_p N)$, avec $n_p = P$ ou $P-1$ selon le type de contrainte utilisé. Cette matrice étant de très grande taille, son simple stockage pose des problèmes de mémoire, en plus du problème lié à son inversion. Cependant, son expression fait intervenir l'opérateur de dérivation spatiale $\boldsymbol{\nabla}_s$ qui est creux par construction. En pratique, la matrice $\boldsymbol{H}_{k,i}$ est construite à l'aide de la structure *sparse* de Matlab. Le problème de l'occupation mémoire s'en trouve résolu. En revanche, l'inversion du système est effectuée à l'aide de la commande d'inversion directe *mldivide* qui, bien que supportant le format *sparse*, n'est pas optimisée pour cette situation. Dans ce chapitre, nous proposons plusieurs stratégies d'implémentation de cette inversion. Les cas non-pénalisé et pénalisé sont étudiés séparément.

## 4.2 Stratégies de calcul dans le cas non-pénalisé

### 4.2.1 Traitement séparé de chaque pixel

D'après la définition dans le problème (2.100) du critère utilisé dans IPLS, en l'absence de pénalisation spatiale, le critère à minimiser peut s'écrire

$$\begin{aligned}\Phi(\boldsymbol{c}) &= F(\boldsymbol{A}^{(0)} + \boldsymbol{Z}\ \mathrm{mat}(\boldsymbol{c})) \\ &= \sum_{n=1}^{N} \frac{1}{2} \left\| \boldsymbol{y}_n - \boldsymbol{S}(\boldsymbol{a}_n^{(0)} + \boldsymbol{Z}\boldsymbol{c}_n) \right\|_2^2 \\ &= \sum_{n=1}^{N} \frac{1}{2} \left\| \tilde{\boldsymbol{y}}_n - \tilde{\boldsymbol{S}}\boldsymbol{c}_n \right\|_2^2, \end{aligned} \tag{4.4}$$

en notant $\tilde{\boldsymbol{y}}_n = \boldsymbol{y}_n - \boldsymbol{S}\boldsymbol{a}_n^{(0)}$ et $\tilde{\boldsymbol{S}} = \boldsymbol{S}\boldsymbol{Z}$. Ce critère étant séparable par pixel, sa solution est la concaténation des solutions des $N$ problèmes portant chacun sur un pixel dont les critères s'écrivent

$$\Phi(\boldsymbol{c}_n) = \frac{1}{2} \left\| \tilde{\boldsymbol{y}}_n - \tilde{\boldsymbol{S}}\boldsymbol{c}_n \right\|_2^2. \tag{4.5}$$

Ainsi, une première stratégie d'implémentation, que nous appellerons *résolution par pixel*, consiste à résoudre le problème (2.100) en appliquant l'algorithme IPLS à chaque pixel séparément. Cela revient à estimer les cartes d'abondances pour $N$ images de taille $1$ pixel, au lieu d'une image de taille $N$ pixels. Pour chacun de ces problèmes, le calcul des directions primales consiste à inverser un système linéaire caractérisé par une matrice de taille $(n_p \times n_p)$, ce qui est très rapide et ne pose pas de problème de mémoire tant que le nombre des spectres purs n'est pas trop élevé. De plus, une telle implémentation peut être portée facilement sur un outil de calcul parallèle puisqu'il s'agit de résoudre un grand nombre de problèmes indépendants de faible taille. Cependant, cette implémentation ne peut pas être utilisée en présence de pénalisation spatiale car le couplage des variables appartenant à des pixels voisins empêche alors la décomposition du problème (2.100) en $N$ problèmes indépendants.

### 4.2.2 Traitement séparé de tous les pixels

En revenant au problème portant sur l'image entière, que nous noterons *résolution par image*, nous proposons une alternative à l'implémentation initiale pour gagner à la fois en temps de calcul et en occupation mémoire.

#### 4.2.2.1 Analyse de la structure du calcul des directions primales

Rappelons l'expression du système linéaire à inverser pour calculer les directions primales, en omettant les indices liés à l'itération

$$\boldsymbol{H}\boldsymbol{d} = -\boldsymbol{g}, \tag{4.6}$$

avec

$$\boldsymbol{H} = \nabla^2\Phi(\boldsymbol{c}) + \boldsymbol{T}^{\mathrm{t}}\mathrm{Diag}(\boldsymbol{T}\boldsymbol{c}+\boldsymbol{t})^{-1}\boldsymbol{\Lambda}\boldsymbol{T}, \tag{4.7}$$
$$\boldsymbol{g} = \nabla\Phi(\boldsymbol{c}) - \boldsymbol{T}^{\mathrm{t}}\mathrm{Diag}(\boldsymbol{T}\boldsymbol{c}+\boldsymbol{t})^{-1}\boldsymbol{\mu}. \tag{4.8}$$

Une analyse de la structure de la matrice creuse $\boldsymbol{H}$ est nécessaire afin de trouver une stratégie d'implémentation appropriée.

D'abord, d'après la définition du critère $\Phi(\cdot)$ dans le cas non-pénalisé, le Hessien $\nabla^2\Phi(\cdot)$ peut s'écrire

$$\begin{aligned}\nabla^2\Phi(\boldsymbol{c}) &= (\boldsymbol{I}_N \otimes \tilde{\boldsymbol{S}})^{\mathrm{t}}(\boldsymbol{I}_N \otimes \tilde{\boldsymbol{S}}) \\ &= \boldsymbol{I}_n \otimes (\tilde{\boldsymbol{S}}^{\mathrm{t}}\tilde{\boldsymbol{S}}).\end{aligned} \tag{4.9}$$

Cette expression du Hessien est la conséquence des propriétés suivantes du produit de Kronecker

$$(\boldsymbol{A}\otimes\boldsymbol{B})^{\mathrm{t}} = \boldsymbol{A}^{\mathrm{t}}\otimes\boldsymbol{B}^{\mathrm{t}} \quad et \quad (\boldsymbol{A}\otimes\boldsymbol{B})(\boldsymbol{C}\otimes\boldsymbol{D}) = (\boldsymbol{A}\boldsymbol{C}\otimes\boldsymbol{B}\boldsymbol{D}). \tag{4.10}$$

Ainsi, $\nabla^2\Phi(\cdot)$ est une matrice bloc-diagonale composée de $N$ blocs identiques $\tilde{\boldsymbol{S}}^{\mathrm{t}}\tilde{\boldsymbol{S}}$. On utilisera désormais la notation

$$\nabla^2\Phi(\boldsymbol{c}) = \underset{n=1\ldots N}{\mathrm{Bdiag}}(\tilde{\boldsymbol{S}}^{\mathrm{t}}\tilde{\boldsymbol{S}}). \tag{4.11}$$

Ensuite, la matrice $\mathrm{Diag}(\boldsymbol{T}\boldsymbol{c}+\boldsymbol{t})^{-1}\boldsymbol{\Lambda}$ étant diagonale, elle peut s'écrire sous forme bloc-diagonale

$$\mathrm{Diag}(\boldsymbol{T}\boldsymbol{c}+\boldsymbol{t})^{-1}\boldsymbol{\Lambda} = \underset{n=1\ldots N}{\mathrm{Bdiag}}(\boldsymbol{D}_n), \tag{4.12}$$

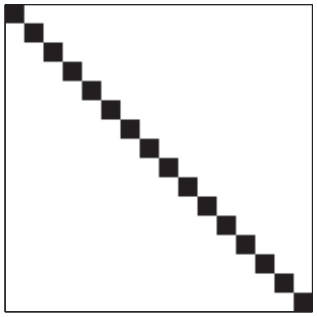


FIGURE 4.1 – Structure bloc-diagonale de la matrice $\boldsymbol{H}$ dans le cas non-pénalisé.

avec $\boldsymbol{D}_n$ la matrice diagonale de taille $(n_d \times n_d)$ dont les éléments diagonaux sont $\text{Diag}(\boldsymbol{T}_1\boldsymbol{Z}\boldsymbol{c}_n + \boldsymbol{T}_1\boldsymbol{a}_n^{(0)} + \boldsymbol{t}_{0,n})^{-1}\lambda_n$, où $\boldsymbol{c}_n \in \mathbb{R}^{n_p}$, $\boldsymbol{a}_n^{(0)} \in \mathbb{R}^P$, $\boldsymbol{t}_{0,n} \in \mathbb{R}^{n_d}$, et $\boldsymbol{\lambda}_n \in \mathbb{R}^{n_d}$ représentent respectivement les n-ièmes colonnes des matrices $\text{mat}(\boldsymbol{c}) \in \mathbb{R}^{n_p \times N}$, $\boldsymbol{A}^{(0)} \in \mathbb{R}^{P \times N}$, $\boldsymbol{T}_0 \in \mathbb{R}^{n_d \times N}$, et $\text{mat}(\lambda) \in \mathbb{R}^{n_d \times N}$.

On rappelle également que

$$\boldsymbol{T} = \boldsymbol{I}_N \otimes \boldsymbol{T}_1\boldsymbol{Z} = \underset{n=1\ldots N}{\text{Bdiag}}(\boldsymbol{T}_1\boldsymbol{Z}). \tag{4.13}$$

Notons $\tilde{\boldsymbol{T}_1} = \boldsymbol{T}_1\boldsymbol{Z}$. Finalement, l'expression de la matrice $\boldsymbol{H}$ se réduit à

$$\begin{aligned} \boldsymbol{H} &= \underset{n=1\ldots N}{\text{Bdiag}}(\tilde{\boldsymbol{S}}^{\text{t}}\tilde{\boldsymbol{S}}) + \underset{n=1\ldots N}{\text{Bdiag}}\tilde{\boldsymbol{T}_1}^{\text{t}}\underset{n=1\ldots N}{\text{Bdiag}}(\boldsymbol{D}_n)\underset{n=1\ldots N}{\text{Bdiag}}\tilde{\boldsymbol{T}_1} \\ &= \underset{n=1\ldots N}{\text{Bdiag}}(\boldsymbol{H}_n), \end{aligned} \tag{4.14}$$

avec $\boldsymbol{H}_n = \tilde{\boldsymbol{S}}^{\text{t}}\tilde{\boldsymbol{S}} + \tilde{\boldsymbol{T}_1}^{\text{t}}\boldsymbol{D}_n\tilde{\boldsymbol{T}_1}$.

Le système linéaire à résoudre pour le calcul des directions primales est donc caractérisé par une matrice $\boldsymbol{H}$ bloc-diagonale avec $N$ blocs de taille $(n_p \times n_p)$. La figure 4.1 permet de visualiser cette structure en mettant en évidence les éléments non-nuls de $\boldsymbol{H}$ pour une image contenant seulement 16 pixels.

#### 4.2.2.2 Stratégies d'implémentations possibles

Lorsqu'on applique la stratégie de *résolution par image* à des données de grande taille, l'espace mémoire requis pour le stockage de la matrice $\boldsymbol{H}$ peut dépasser l'espace disponible, même avec l'utilisation de la structure *sparse* de Matlab. Un calcul des directions primales moins gourmand en terme d'occupation mémoire peut être réalisé en résolvant séparément les $N$ systèmes linéaires de taille réduite

$$\boldsymbol{H}_n\boldsymbol{d}_n = \boldsymbol{g}_n, \quad \forall n = 1 \ldots N, \tag{4.15}$$

où $\boldsymbol{d}_n$ et $\boldsymbol{g}_n$ sont respectivement les $n^{\text{ièmes}}$ colonnes des matrices $\text{mat}(\boldsymbol{d})$ et $\text{mat}(\boldsymbol{g})$. Cette implémentation sera appelée *résolution par image - calcul des directions par pixel*. Par opposition,

| Méthode | IPLS *par pixel* | IPLS *par image* |
|---|---|---|
| Temps CPU (en s) | 45.7 s | 2.34 s |
| Résidu ($\times 10^{-3}$) | 2.13 | 2.13 |
| EQNM | 9.88 % | 9.88 % |

TABLE 4.2 – Comparaison des résultats avec et sans pénalisation.

l'implémentation initiale où la matrice $\boldsymbol{H}$ est entièrement construite est appelée *résolution par image - calcul des directions par image*. La stratégie de *calcul des directions par pixel* étant basée sur la résolution de $N$ systèmes linéaires indépendants de taille réduite, elle est adaptée à une implémentation parallèle.

Une stratégie intermédiaire entre le calcul des directions *par pixel* et *par image* est également considérée. Elle sera appelée *résolution par image - calcul des directions par bloc*. Il s'agit de diviser le système (4.6) en $1 \leqslant K \leqslant N$ blocs afin d'adapter la taille des équations $K$ à la quantité de mémoire disponible. Lorsque $K = 1$, on retrouve la stratégie du *calcul des directions par image*. Lorsque $K = N$, on retrouve la stratégie du *calcul des directions par pixel*.

### 4.2.3 Résultats des différentes versions proposées

Comparons d'abord l'implémentation *par pixel* et l'implémentation initiale *par image*. Le tableau 4.2 donne les résultats en termes de temps de calcul, de résidu, et d'erreur de reconstruction, pour ces deux implémentations. On voit que la version *par pixel* est nettement moins rapide que la version *par image*. Lors d'une exécution de Matlab sur CPU, il est donc préférable de conserver un problème de grande taille au lieu de le décomposer en plusieurs problèmes de faible taille. L'implémentation *par pixel* est néanmoins conservée pour être testée sur GPU au chapitre 5. La concordance du résidu et de l'erreur de reconstruction montre, comme cela était attendu, que les deux versions convergent bien vers le même résultat.

Dans le cas de l'implémentation *par image*, comparons à présent les différentes façons de calculer les directions de Newton : *par pixel*, *par bloc*, ou *par image*. La figure 4.2 montre le temps de calcul obtenu avec un calcul des directions *par bloc* pour différentes tailles de bloc, pour un nombre de pixels de $N = 100^2$ et différents nombres de spectres purs $P$. Chaque courbe admet un minimum qui n'est situé ni en 1 (calcul des directions *par pixel*), ni en $N$ (calcul des directions *par image*). Plus le nombre de spectres purs est élevé, plus la taille optimale des blocs est faible, et plus la tolérence autour de cette valeur optimale est faible. Il apparait qu'avec des blocs de 100 pixels, le temps de calcul est proche du meilleur temps de calcul pour différentes valeurs de $P$. Dans les conditions prises comme référence ($P = 10$), le temps de calcul passe alors de 2.34 s à 1.44 s, soit un gain de 38 %. De plus, cette valeur permet d'économiser fortement l'espace mémoire requis par IPLS qui est proportionnel à la taille des blocs comme le montre le deuxième graphique de la figure 4.2.

Ainsi, une implémentation *par image* avec un calcul des directions de Newton *par bloc* de 100 pixels est adoptée pour le reste de cette thèse. Rapellons que l'implémentation *par image* est de toute façon nécessaire lorsque la pénalisation spatiale est utilisée car le problème n'est alors pas décomposable en plusieurs problèmes indépendants.

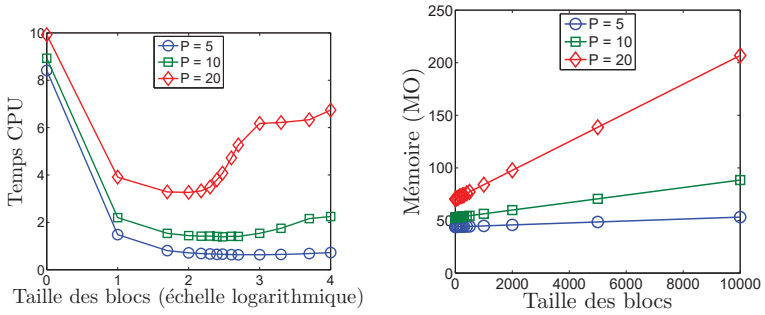



FIGURE 4.2 – Temps de calcul (en secondes) et occupation mémoire (en MO) de l'algorithme IPLS avec calcul des directions de Newton par bloc pour différentes tailles de bloc et différents nombre de endmembers.

## 4.3 Accélération dans le cas pénalisé

### 4.3.1 Analyse de la structure du calcul des directions primales

Le système linéaire à résoudre pour obtenir les directions primales est toujours

$$\boldsymbol{H}\boldsymbol{d} = -\boldsymbol{g}, \tag{4.16}$$

avec

$$\boldsymbol{H} = \nabla^2\Phi(\boldsymbol{c}) + \boldsymbol{T}^{\mathrm{t}}\mathrm{Diag}(\boldsymbol{T}\boldsymbol{c}+\boldsymbol{t})^{-1}\boldsymbol{T}, \tag{4.17}$$
$$\boldsymbol{g} = \nabla\Phi(\boldsymbol{c}) - \boldsymbol{T}^{\mathrm{t}}\mathrm{Diag}(\boldsymbol{T}\boldsymbol{c}+\boldsymbol{t})^{-1}\boldsymbol{\mu}. \tag{4.18}$$

Dans le cas pénalisé, le critère $\Phi(\cdot)$ et ses dérivées s'écrivent

$$\begin{aligned}\Phi(\boldsymbol{c}) &= F(\boldsymbol{Y} - \boldsymbol{S}(\boldsymbol{A}^{(0)} + \boldsymbol{Z}\ \mathrm{mat}(\boldsymbol{c})))\\ &= \left\|\mathrm{vect}(\tilde{\boldsymbol{Y}}) - (\boldsymbol{I}_N \otimes \tilde{\boldsymbol{S}})\boldsymbol{c}\right\|_2^2 + \beta\sum_{i=1}^{2PN}\varphi(\left[(\boldsymbol{\nabla}_s \otimes \boldsymbol{I}_P)(\mathrm{vect}(\boldsymbol{A}^{(0)}) + (\boldsymbol{I}_N \otimes \boldsymbol{Z})\boldsymbol{c})\right]_i)\\ &= \left\|\mathrm{vect}(\tilde{\boldsymbol{Y}}) - (\boldsymbol{I}_N \otimes \tilde{\boldsymbol{S}})\boldsymbol{c}\right\|_2^2 + \beta\sum_{i=1}^{2PN}\varphi([(\boldsymbol{\nabla}_s \otimes \boldsymbol{Z})\boldsymbol{c})]_i) \quad \text{car } \boldsymbol{A}^{(0)} \text{ est constante,}\end{aligned} \tag{4.19}$$

$$\nabla\Phi(\boldsymbol{c}) = (\boldsymbol{I}_N \otimes \tilde{\boldsymbol{S}}^{\mathrm{t}}\tilde{\boldsymbol{S}})\boldsymbol{c} - (\boldsymbol{I}_N \otimes \tilde{\boldsymbol{S}})\ \mathrm{vect}(\tilde{\boldsymbol{Y}}) + \beta(\boldsymbol{\nabla}_s \otimes \boldsymbol{Z})^{\mathrm{t}}\dot{\varphi}((\boldsymbol{\nabla}_s \otimes \boldsymbol{Z})\boldsymbol{c}), \tag{4.20}$$

$$\nabla^2\Phi(\boldsymbol{c}) = \boldsymbol{I}_N \otimes \tilde{\boldsymbol{S}}^{\mathrm{t}}\tilde{\boldsymbol{S}} + \beta(\boldsymbol{\nabla}_s \otimes \boldsymbol{Z})^{\mathrm{t}}\mathrm{Diag}(\ddot{\varphi}((\boldsymbol{\nabla}_s \otimes \boldsymbol{Z})\boldsymbol{c}))(\boldsymbol{\nabla}_s \otimes \boldsymbol{Z}), \tag{4.21}$$

où $\dot{\varphi}(\boldsymbol{x})$ est le vecteur dont les éléments sont $\frac{\partial\varphi(x_i)}{\partial x_i}$, et $\ddot{\varphi}(\boldsymbol{x})$ est le vecteur dont les éléments sont $\frac{\partial^2\varphi(x_i)}{\partial x_i^2}$.

On peut donc écrire

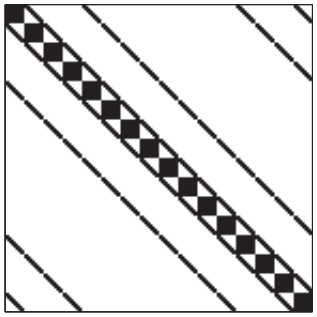


FIGURE 4.3 – Structure non bloc-diagonale de la matrice $\boldsymbol{H}$ dans le cas pénalisé.

$$\boldsymbol{H} = \underset{n=1\ldots N}{\text{Bdiag}}(\boldsymbol{H}_n) + \beta(\boldsymbol{\nabla}_s \otimes \boldsymbol{Z})^{\text{t}}\text{Diag}(\ddot{\varphi}((\boldsymbol{\nabla}_s \otimes \boldsymbol{Z})\boldsymbol{c}))(\boldsymbol{\nabla}_s \otimes \boldsymbol{Z}). \tag{4.22}$$

Nous avons déjà montré le caractère bloc-diagonal du premier terme. En revanche, le second terme faisant intervenir la pénalisation ne l'est pas. Par conséquent, la matrice $\boldsymbol{H}$ n'est pas bloc-diagonale. On peut observer ce phénomène sur la figure 4.3 qui donne les positions des valeurs non nulles de $\boldsymbol{H}$. Cela s'explique par le couplage que réalise la pénalisation entre les variables des pixels voisins. Une stratégie d'inversion rapide d'un tel système est proposé dans [Moussaoui *et al.*, 2012]. L'idée de cet article est d'appliquer une méthode de Newton tronqué dans le cadre d'une méthode de points intérieurs [Dembo et Steihaug, 1983; Armand *et al.*, 2012]. La solution du système (4.16) est approchée à l'aide d'un algorithme de Gradient Conjugué préconditionné dans lequel le préconditionneur est obtenu par factorisation de Cholesky incomplète de la matrice $\boldsymbol{H}$. Cette technique permet une accélération du calcul des directions primales, mais celle-ci n'exploite pas la structure particulière du système. En effet, sur la figure 4.3, nous observons que les termes non-nuls qui n'entrent pas dans la structure bloc-diagonale sont peu nombreux. La matrice $\boldsymbol{H}$ est donc « proche » d'une matrice bloc-diagonale. Cette observation conduit à l'idée de remplacer l'inversion du système (4.16) par une séquence d'inversions de systèmes caractérisés par des matrices bloc-diagonales. Il s'agit donc également d'appliquer une méthode de Newton tronqué, mais cette fois en exploitant la structure du problème. Cette idée peut être mise en œuvre dans un premier temps grâce à une approche de type Majoration-Minimisation. Nous verrons par la suite que cette démarche peut être améliorée avec une méthode de Gradient Conjugué préconditionné dont le préconditionneur est issu des développements de l'approche MM.

## 4.3.2 Résolution par approche par Majoration-Minimisation Quadratique

### 4.3.2.1 L'approche MMQ

Avant de décrire comment une approche par Majoration-Minimisation Quadratique (MMQ) peut accélérer le calcul des directions primales, présentons l'algorithme MMQ d'une manière générale. L'algorithme MM apparaît pour la première fois dans [Ortega et Rheinboldt, 2000]. Une description détaillée est donnée dans [Hunter et Lange, 2004]. Cette approche permet de résoudre un problème sous la forme

$$\underset{\boldsymbol{x}\in\mathcal{D}_f}{\text{minimiser}} \quad F(\boldsymbol{x}), \tag{4.23}$$

où $\mathcal{D}_f \subset \mathbb{R}^n$ et $F : \mathcal{D}_f \to \mathbb{R}$ est une fonction convexe. Afin d'appliquer la méthode MMQ à ce problème, nous devons introduire la notion d'approximation tangente majorante d'un critère en un point.

La fonction $H(\cdot, \boldsymbol{y})$ est dite approximation tangente majorante de $F(\cdot)$ en $\boldsymbol{y}$ sur le domaine $\mathcal{D}_f$ si pour tout $\boldsymbol{x} \in \mathcal{D}_f$,

$$\begin{aligned} H(\boldsymbol{x}, \boldsymbol{y}) &\geqslant F(\boldsymbol{x}), \\ H(\boldsymbol{y}, \boldsymbol{y}) &= F(\boldsymbol{y}). \end{aligned} \tag{4.24}$$

Lorsque $H(\cdot, \boldsymbol{y})$ est différentiable sur $\mathcal{D}_f$, cette définition implique que la fonction $H(\cdot, \boldsymbol{y})$ est tangente à $F(\cdot)$ en $\boldsymbol{y}$, c'est-à-dire

$$\nabla_1 H(\boldsymbol{y}, \boldsymbol{y}) = \nabla F(\boldsymbol{y}), \tag{4.25}$$

en notant $\nabla_1$ le gradient de $H(\cdot, \cdot)$ par rapport à sa première variable.

L'algorithme MMQ se base sur la construction d'approximations tangentes majorantes quadratiques du critère. Il suit de la définition d'une approximation tangente majorante qu'une approximation tangente majorante quadratique s'écrit

$$H(\boldsymbol{x}, \boldsymbol{y}) = F(\boldsymbol{y}) + \nabla F(\boldsymbol{y})^{\mathrm{t}}(\boldsymbol{x} - \boldsymbol{y}) + \frac{1}{2}(\boldsymbol{x} - \boldsymbol{y})^{\mathrm{t}}\boldsymbol{A}(\boldsymbol{y})(\boldsymbol{x} - \boldsymbol{y}), \tag{4.26}$$

où $\boldsymbol{A}(\boldsymbol{y}) \in \mathbb{R}^{n\times n}$ est une matrice définie positive assurant que (4.26) vérifie les conditions (4.24). D'une façon générale, toute matrice $\boldsymbol{A}(\boldsymbol{y})$ telle que $\boldsymbol{A}(\boldsymbol{y}) - \nabla^2 F(\boldsymbol{y})$ soit définie positive entraîne que (4.26) est une approximation majorante de $F$ [Hunter et Lange, 2004].

Décrivons maintenant comment ces approximations majorantes sont utilisées pour résoudre le problème initial. La minimisation du critère $F(\cdot)$ par Majoration-Minimisation consiste à résoudre le problème (4.23) par la règle de mise à jour MM

$$\boldsymbol{x}_{j+1} = \underset{\boldsymbol{x}\in\mathcal{D}_f}{\text{argmin}} \quad H(\boldsymbol{x}, \boldsymbol{x}_j), \tag{4.27}$$

illustrée dans la figure 4.4. La convergence de cet algorithme est assurée dès lors que $F(\cdot)$ est une fonction convexe. Pour que la méthode MM soit efficace en pratique, il faut que la minimisation de la fonction majorante $H(\cdot, \boldsymbol{x}_j)$ soit plus rapide que celle du critère $F(\cdot)$. C'est généralement le cas lorsque $H(\cdot, \boldsymbol{x}_j)$ est quadratique car son minimum s'obtient alors de façon analytique

$$\boldsymbol{x}_{j+1} = \boldsymbol{x}_j - \boldsymbol{A}(\boldsymbol{x}_j)^{-1}\nabla F(\boldsymbol{x}_j). \tag{4.28}$$

L'implémentation de la méthode MMQ est réalisé à travers l'algorithme 4

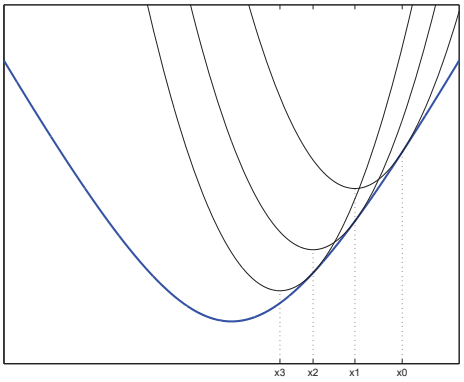



FIGURE 4.4 – Illustration de l'algorithme MM.

Initialiser $j = 0$ et $\boldsymbol{x}_0 \in \mathcal{D}_f$
**Tant que** ( la convergence n'est pas atteinte) **faire**
  Choisir $\boldsymbol{A}(\boldsymbol{x}_j)$ telle que $\boldsymbol{A}(\boldsymbol{x}_j) - \nabla^2 F(\boldsymbol{x}_j)$ soit définie positive
  Mettre à jour $\boldsymbol{x}_{j+1} = \boldsymbol{x}_j - \boldsymbol{A}(\boldsymbol{x}_j)^{-1} \nabla F(\boldsymbol{x}_j)$
  $j = j + 1$
**Fait**

Algorithme 4: Algorithme MMQ

#### 4.3.2.2 Intégration de l'approche MM dans l'algorithme IPLS

**De la résolution d'un système à la minimisation d'une fonction**

La minimisation d'un critère dérivable $F$, en l'absence de contrainte, se fait généralement par l'annulation de son gradient $\nabla F$. On transforme alors un problème de minimisation de critère en une équation à résoudre $\nabla F(\hat{\boldsymbol{x}}) = 0$. Ici, on cherche à résoudre l'équation (4.16) : $\boldsymbol{H}\boldsymbol{d} = -\boldsymbol{g}$. On peut transformer ce problème en un problème de minimisation de la primitive de cette expression. Ainsi, la résolution de l'équation (4.16) est équivalente à

$$\underset{\boldsymbol{d} \in \mathbb{R}^{n_p N}}{\text{minimiser}} \quad f(\boldsymbol{d}) = \boldsymbol{g}^{\mathrm{t}} \boldsymbol{d} + \frac{1}{2} \boldsymbol{d}^{\mathrm{t}} \boldsymbol{H} \boldsymbol{d}. \tag{4.29}$$

Nous proposons de résoudre le problème (4.29) à l'aide de la méthode MMQ.

**Approximation tangente majorante quadratique**

D'après l'équation (4.26), l'approximation tangente majorante de $f(\cdot)$ en un point $\boldsymbol{d}_j \in \mathbb{R}^{n_p N}$ s'écrit

$$h(\boldsymbol{d}, \boldsymbol{d}_j) = f(\boldsymbol{d}_j) + \nabla f(\boldsymbol{d}_j)^{\mathrm{t}} (\boldsymbol{d} - \boldsymbol{d}_j) + \frac{1}{2} (\boldsymbol{d} - \boldsymbol{d}_j)^{\mathrm{t}} \boldsymbol{B}(\boldsymbol{d}_j) (\boldsymbol{d} - \boldsymbol{d}_j). \tag{4.30}$$

D'après le résultat de [Hunter et Lange, 2004] rappelé précédemment, la fonction $h(\cdot, \boldsymbol{d}_j)$ est une approximation tangente majorante de $f(\cdot)$ en $\boldsymbol{d}_j$ à condition que $\boldsymbol{B}(\boldsymbol{d}_j) - \nabla^2 f(\boldsymbol{d}_j)$ soit définie positive.

**Proposition 1.** *La fonction* $h(\cdot, \boldsymbol{d}_j)$ *définie par* (4.30) *avec*

$$\boldsymbol{B}(\boldsymbol{d}_j) = \boldsymbol{B} = \underset{n=1\ldots N}{Bdiag}(\boldsymbol{H}_n) + 8\beta m_{\boldsymbol{c}} \boldsymbol{I}_N \otimes (\boldsymbol{Z}^{\mathrm{t}}\boldsymbol{Z}), \tag{4.31}$$

$$m_{\boldsymbol{c}} \geqslant \ max(\ddot{\varphi}((\boldsymbol{\nabla}_s \otimes \boldsymbol{Z})\boldsymbol{c})), \tag{4.32}$$

*est une approximation tangente majorante quadratique de* $f(\cdot)$ *définie par* (4.29) *en* $\boldsymbol{d}_k$.

**Preuve 1.** *Notons* $\boldsymbol{A} \geqslant 0$ *la propriété de définie positivité d'une matrice* $\boldsymbol{A}$. *Il s'agit de montrer que*

$$\boldsymbol{B} - \nabla^2 f(\boldsymbol{d}_j) \geqslant 0. \tag{4.33}$$

*On rappelle d'abord que*

$$\begin{aligned} \nabla^2 f(\boldsymbol{d}_j) &= \boldsymbol{H} \\ &= \underset{n=1\ldots N}{Bdiag}(\boldsymbol{H}_n) + \beta(\boldsymbol{\nabla}_s \otimes \boldsymbol{Z})^{\mathrm{t}} Diag(\ddot{\varphi}((\boldsymbol{\nabla}_s \otimes \boldsymbol{Z})\boldsymbol{c}))(\boldsymbol{\nabla}_s \otimes \boldsymbol{Z}). \end{aligned} \tag{4.34}$$

*Avec les notations*

$$\boldsymbol{M}_1 = 8m_{\boldsymbol{c}} \boldsymbol{I}_N \otimes (\boldsymbol{Z}^{\mathrm{t}}\boldsymbol{Z}), \tag{4.35}$$

$$\boldsymbol{M}_2 = (\boldsymbol{\nabla}_s \otimes \boldsymbol{Z})^{\mathrm{t}} m_{\boldsymbol{c}} (\boldsymbol{\nabla}_s \otimes \boldsymbol{Z}), \tag{4.36}$$

$$\boldsymbol{M}_3 = (\boldsymbol{\nabla}_s \otimes \boldsymbol{Z})^{\mathrm{t}} max(\ddot{\varphi}((\boldsymbol{\nabla}_s \otimes \boldsymbol{Z})\boldsymbol{c}))(\boldsymbol{\nabla}_s \otimes \boldsymbol{Z}), \tag{4.37}$$

$$\boldsymbol{M}_4 = (\boldsymbol{\nabla}_s \otimes \boldsymbol{Z})^{\mathrm{t}} Diag(\ddot{\varphi}((\boldsymbol{\nabla}_s \otimes \boldsymbol{Z})\boldsymbol{c}))(\boldsymbol{\nabla}_s \otimes \boldsymbol{Z}), \tag{4.38}$$

*la présente preuve se réduit à montrer que*

$$\boldsymbol{M}_1 - \boldsymbol{M}_4 \geqslant 0. \tag{4.39}$$

*La convexité de* $\varphi$ *entraîne directement la relation*

$$\boldsymbol{M}_3 - \boldsymbol{M}_4 \geqslant 0. \tag{4.40}$$

*La condition sur* $m_{\boldsymbol{c}}$ *donnée par l'équation* (4.32) *entraîne directement la relation*

$$\boldsymbol{M}_2 - \boldsymbol{M}_3 \geqslant 0. \tag{4.41}$$

*Ainsi, il est suffisant de montrer que*

$$\boldsymbol{M}_1 - \boldsymbol{M}_2 \geqslant 0. \tag{4.42}$$

*D'après les propriété du produit de Kronecker, cette relation est équivalente à*

$$(8\boldsymbol{I}_N - \boldsymbol{\nabla}_s^{\mathrm{t}}\boldsymbol{\nabla}_s) \otimes (\boldsymbol{Z}^{\mathrm{t}}\boldsymbol{Z}) \geqslant 0. \tag{4.43}$$

*Par construction, on sait que* $\boldsymbol{Z}^{\mathrm{t}}\boldsymbol{Z} \geqslant 0$*, il est donc suffisant de montrer que*

$$8\boldsymbol{I}_N - \boldsymbol{\nabla}_s^{\mathrm{t}}\boldsymbol{\nabla}_s \geqslant 0, \tag{4.44}$$

*ou, de façon équivalente, que*

$$4\boldsymbol{I}_N - \boldsymbol{\nabla}_v^{\mathrm{t}}\boldsymbol{\nabla}_v + 4\boldsymbol{I}_N - \boldsymbol{\nabla}_h^{\mathrm{t}}\boldsymbol{\nabla}_h \geqslant 0. \tag{4.45}$$

*Montrons que* $4\boldsymbol{I}_N - \boldsymbol{\nabla}_v^{\mathrm{t}}\boldsymbol{\nabla}_v \geqslant 0$*. Par construction, on a*

$$\left[\boldsymbol{\nabla}_v^{\mathrm{t}}\boldsymbol{\nabla}_v\right]_{ij} = \begin{cases} 2 & \textit{si} \quad i = j, \\ -1 & \textit{si} \quad i = (j+1) \quad \textit{modulo} \quad N, \\ -1 & \textit{si} \quad i = (j-1) \quad \textit{modulo} \quad N, \\ 0 & \textit{sinon}. \end{cases} \tag{4.46}$$

*D'après [Gray, 2006, equation (3.7)], les valeurs propres de* $\boldsymbol{\nabla}_v^{\mathrm{t}}\boldsymbol{\nabla}_v$ *sont, pour* $m = \{1, \ldots, N\}$

$$\begin{aligned} \textit{eig}(\boldsymbol{\nabla}_v^{\mathrm{t}}\boldsymbol{\nabla}_v)_m &= 2 - \exp(-2i\pi m/N) - \exp(2i\pi m/N) \\ &= 2 - 2\cos(2\pi m/N). \end{aligned} \tag{4.47}$$

*D'après [Petersen et Pedersen, 2006, equation (264)], on a*

$$\begin{aligned} \textit{eig}(4\boldsymbol{I}_N - \boldsymbol{\nabla}_v^{\mathrm{t}}\boldsymbol{\nabla}_v)_m &= 4 - \textit{eig}(\boldsymbol{\nabla}_v^{\mathrm{t}}\boldsymbol{\nabla}_v)_m \\ &= 2 + 2\cos(2\pi m/N) \\ &\geqslant 0. \end{aligned} \tag{4.48}$$

*Ainsi,* $4\boldsymbol{I}_N - \boldsymbol{\nabla}_v^{\mathrm{t}}\boldsymbol{\nabla}_v \geqslant 0$*. Une approche similaire permet de montrer que* $4\boldsymbol{I}_N - \boldsymbol{\nabla}_h^{\mathrm{t}}\boldsymbol{\nabla}_h \geqslant 0$*.* ■

**Règle de mise à jour MM**

La minimisation de la fonction quadratique $h(\cdot, \boldsymbol{d}_j)$ produit le nouvel itéré de l'algorithme MMQ d'après l'expression

$$\boldsymbol{d}_{j+1} = \boldsymbol{d}_j - \boldsymbol{B}^{-1}(\boldsymbol{g} + \boldsymbol{H}\boldsymbol{d}_j). \tag{4.49}$$

**Remarques**

La matrice $\boldsymbol{B}$ est par construction bloc-diagonale. On en déduit que la matrice $\boldsymbol{B}^{-1}$ est également bloc-diagonale. De plus, elle reste constante au cours de l'algorithme MMQ. Il est donc possible d'effectuer le pré-calcul de $\boldsymbol{B}^{-1}$ en dehors des itérations MM.

Désormais, comme dans le cas non-pénalisé, il est possible d'effectuer le pré-calcul de $\boldsymbol{B}^{-1}$ ainsi que les mises à jour de $\boldsymbol{d}_k$ avec une stratégie *par pixel*, *par bloc*, ou *par image*.

On peut affecter à $m_{\boldsymbol{c}}$ la valeur exacte de $\max(\ddot{\varphi}((\boldsymbol{\nabla}_s \otimes \boldsymbol{Z})\boldsymbol{c}))$ pour une majoration plus proche du critère à minimiser. Cependant, cela nécessite un grand nombre d'évaluations de la fonction $\nabla^2\varphi(\cdot)$. Pour les fonctions de régularisation de type $\ell_2$ et $\ell_2 - \ell_1$ considérées dans ce travail, il est plus rapide d'affecter à $m_{\boldsymbol{c}}$ la valeur $\max_{x\in\mathbb{R}}(\nabla^2\varphi(x))$. En effet, dans le cas $\ell_2$, on a

$$\begin{aligned} \varphi(x) &= \frac{1}{2}x^2, \\ \nabla\varphi(x) &= x, \\ \nabla^2\varphi(x) &= 1, \\ \max_{x\in\mathbb{R}}(\nabla^2\varphi(x)) &= 1, \end{aligned} \tag{4.50}$$

et dans le cas $\ell_2 - \ell_1$, on a

$$\begin{aligned} \varphi(x) &= (\delta^2 + x^2)^{1/2} - \delta, \quad \text{avec } \delta > 0, \\ \nabla\varphi(x) &= x(\delta^2 + x^2)^{-1/2}, \\ \nabla^2\varphi(x) &= (\delta^2 + x^2)^{-1/2} - x^2(\delta^2 + x^2)^{-3/2}, \\ \max_{x\in\mathbb{R}}(\nabla^2\varphi(x)) &= \delta^{-1}. \end{aligned} \tag{4.51}$$

On prendra donc $m_{\boldsymbol{c}} = 1$ dans le cas d'une régularisation de type $\ell_2$, et $m_{\boldsymbol{c}} = \delta^{-1}$ dans le cas $\ell_2 - \ell_1$.

**Critère d'arrêt**

L'algorithme MMQ est arrêté lorsque

$$\|\boldsymbol{r}_{Pj}\|_2 \leqslant \mu \|\boldsymbol{r}_{P0}\|_2 , \tag{4.52}$$

où $\boldsymbol{r}_{Pj} = \boldsymbol{g} + \boldsymbol{H}\boldsymbol{d}_k$ est le résidu du système primal et $\boldsymbol{r}_{P0} = \boldsymbol{g}$ est le résidu initial, et $\mu$ est le paramètre de perturbation des conditions KKT [Armand *et al.*, 2012; Nocedal et Wright, 1999; Johnson *et al.*, 2000].

**Algorithme**

Dans l'algorithme IPLS, la résolution du système primal $\boldsymbol{H}\boldsymbol{d} = -\boldsymbol{g}$ est effectuée de façon approchée avec l'algorithme 5.

Initialiser $j = 0$ et $\boldsymbol{d}_0 = 0$
Former la matrice bloc-diagonale $\boldsymbol{B}$ d'après (4.31)
Calculer la matrice bloc-diagonale $\boldsymbol{B}^{-1}$
**Tant que** ( (4.52) n'est pas vérifiée) **faire**
  Mettre à jour $\boldsymbol{d}_{j+1} = \boldsymbol{d}_j - \boldsymbol{B}^{-1}(\boldsymbol{g} + \boldsymbol{H}\boldsymbol{d}_j)$
  $j = j + 1$
**Fait**

Algorithme 5: Algorithme MMQ pour le calcul des directions primales

**Convergence**

Dans [Armand *et al.*, 2012], un algorithme de points intérieurs avec résolution approchée du système primal-dual complet est étudié. La convergence est démontrée lorsque le résidu de ce système respecte la condition

$$\|\boldsymbol{r}_{\mu_k}(\boldsymbol{c}_j, \boldsymbol{\lambda}_j)\|_2 \leqslant \eta_k \|\boldsymbol{r}_{\mu_k}(\boldsymbol{c}_0, \boldsymbol{\lambda}_0)\|_2 + \zeta_k, \tag{4.53}$$

avec $k$ l'itération courante de l'algorithme de points intérieurs, $j$ l'itération de l'algorithme MMQ, $\{\eta_k\}$ une suite telle que $0 \leqslant \eta_k < 1$ pour tout $k$, $\{\zeta_k\}$ une suite convergeant vers 0, et $\boldsymbol{r}_\mu(\boldsymbol{c}, \boldsymbol{\lambda}) = \left[\boldsymbol{r}_\mu(\boldsymbol{c}, \boldsymbol{\lambda})^{\text{prim}}, \boldsymbol{r}_\mu(\boldsymbol{c}, \boldsymbol{\lambda})^{\text{dual}}\right]^{\text{t}}$ le résidu primal-dual. Plusieurs choix pour les suites $\{\eta_k\}$ et $\{\zeta_k\}$ sont discutés. Un choix simple et qui donne de bons résultats est $\eta_k = \mu_k$ et $\zeta_k = 0$ pour tout $k$. La condition (4.53) devient alors

$$\|\boldsymbol{r}_{\mu_k}(\boldsymbol{c}_j, \boldsymbol{\lambda}_j)\|_2 \leqslant \mu_k \|\boldsymbol{r}_{\mu_k}(\boldsymbol{c}_0, \boldsymbol{\lambda}_0)\|_2 . \tag{4.54}$$

On en déduit le théorème suivant :

**Théorème 4.1.** *L'algorithme IPLS dans lequel les directions primales sont calculées de façon approchée avec l'algorithme 5 converge vers la solution du problème* (2.100).

**Preuve 2.** *La condition nécessaire de convergence dans [Armand* et al.*, 2012] porte sur le résidu du système primal-dual* (2.79)*, alors que le critère d'arrêt donné par* (4.52) *porte sur le résidu du système réduit* (2.80)*. Montrons que ces deux résidus ont la même norme dès lors que les directions duales sont déduites des directions primales par l'équation* (2.83)*.*

*Simplifions les notations en écrivant le système primal-dual*

$$\begin{bmatrix} \boldsymbol{A} & \boldsymbol{B} \\ \boldsymbol{C} & \boldsymbol{D} \end{bmatrix} \begin{bmatrix} \boldsymbol{x}_1 \\ \boldsymbol{x}_2 \end{bmatrix} = \begin{bmatrix} \boldsymbol{y}_1 \\ \boldsymbol{y}_2 \end{bmatrix}, \tag{4.55}$$

*et le système réduit*

$$(\boldsymbol{A} - \boldsymbol{B}\boldsymbol{D}^{-1}\boldsymbol{C})\boldsymbol{x}_1 = \boldsymbol{y}_1 - \boldsymbol{B}\boldsymbol{D}^{-1}\boldsymbol{y}_2. \tag{4.56}$$

*Il s'agit de montrer que la norme résidu $\boldsymbol{r}$ du système complet est égale celle du résidu $\boldsymbol{r}_R$ du système réduit lorsque $\boldsymbol{x}_2 = \boldsymbol{D}^{-1}(\boldsymbol{y}_2 - \boldsymbol{C}\boldsymbol{x}_1)$.*

*On a*

$$\begin{aligned}
\|\boldsymbol{r}\| &= \left\| \begin{bmatrix} \boldsymbol{y}_1 \\ \boldsymbol{y}_2 \end{bmatrix} - \begin{bmatrix} \boldsymbol{A} & \boldsymbol{B} \\ \boldsymbol{C} & \boldsymbol{D} \end{bmatrix} \begin{bmatrix} \boldsymbol{x}_1 \\ \boldsymbol{x}_2 \end{bmatrix} \right\| \\
&= \left\| \begin{bmatrix} \boldsymbol{y}_1 \\ \boldsymbol{y}_2 \end{bmatrix} - \begin{bmatrix} \boldsymbol{A}\boldsymbol{x}_1 & \boldsymbol{B}\boldsymbol{D}^{-1}(\boldsymbol{y}_2 - \boldsymbol{C}\boldsymbol{x}_1) \\ \boldsymbol{C}\boldsymbol{x}_1 & \boldsymbol{D}\boldsymbol{D}^{-1}(\boldsymbol{y}_2 - \boldsymbol{C}\boldsymbol{x}_1) \end{bmatrix} \right\| \\
&= \left\| \begin{bmatrix} \boldsymbol{y}_1 - \boldsymbol{B}\boldsymbol{D}^{-1}\boldsymbol{y}_2 - (\boldsymbol{A} - \boldsymbol{B}\boldsymbol{D}^{-1})\boldsymbol{x}_1 \\ 0 \end{bmatrix} \right\| \\
&= \left\| \begin{bmatrix} \boldsymbol{r}_R \\ 0 \end{bmatrix} \right\| \\
&= \|\boldsymbol{r}_R\| .
\end{aligned} \tag{4.57}$$

*Ainsi, la condition d'arrêt de l'algorithme 5, donnée par l'équation (4.52), correspond exactement à la condition (4.54), qui est un cas particulier respectant la condition nécessaire de convergence de [Armand et al., 2012], donnée par l'équation (4.53).* ■

### 4.3.3 Résolution par gradient conjugué préconditionné

Avec l'approximation majorante considérée, la règle de mise à jour de l'algorithme MMQ s'écrit $\boldsymbol{x}_{j+1} = \boldsymbol{x}_j - \boldsymbol{B}^{-1}\nabla f(\boldsymbol{x}_j)$. On peut reconnaître la règle de mise à jour d'un algorithme de gradient péconditionné à pas fixe avec un pas unitaire et un préconditionneur $\boldsymbol{B}$. Le choix de l'approche MMQ pour le calcul des directions primales a donc abouti à un algorithme de gradient préconditionné. Cependant, le passage par l'approche MMQ a permis d'exhiber une règle de recherche de pas simple garantissant la convergence ainsi qu'un préconditionneur adapté au problème permettant une convergence plus rapide. Cette observation conduit naturellement à la méthode du gradient conjugué qui est une amélioration de la méthode du gradient.

#### 4.3.3.1 La méthode du gradient conjugué

La méthode du gradient conjugué permet de résoudre un système linéaire de la forme $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$, avec $\boldsymbol{A} \in \mathbb{R}^{n\times n}$ une matrice symétrique définie positive. Comme montré précédemment, la résolution d'un tel système est équivalente au problème d'optimisation non contraint suivant

$$\underset{\boldsymbol{x}\in\mathbb{R}^n}{\text{minimiser}} \quad f(\boldsymbol{x}) = \frac{1}{2}\boldsymbol{x}^{\mathrm{t}}\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}^{\mathrm{t}}\boldsymbol{x}, \tag{4.58}$$

car le gradient du critère s'écrit alors $\nabla f(\boldsymbol{x}) = \boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}$.

La méthode du gradient conjugué fait partie des méthodes de descente qui ont comme principe commun la recherche de la solution $\hat{\boldsymbol{x}}$ à partir d'un point initial $\boldsymbol{x}_0$ suivant le procédé itératif

$$\boldsymbol{x}_{j+1} = \boldsymbol{x}_j + \alpha_j \boldsymbol{d}_j, \tag{4.59}$$

avec $\boldsymbol{d}_j \in \mathbb{R}^n$ une direction de descente, c'est-à-dire respectant $\boldsymbol{d}_j^{\mathrm{t}}\nabla f(\boldsymbol{x}_j) \leqslant 0$, et $\alpha_j > 0$ le pas de descente assurant une décroissance suffisante du critère dans la direction définie par $\boldsymbol{d}_j$.

**Choix du pas de descente $\alpha_j$**

La fonction $f$ étant quadratique, son minimiseur suivant une direction donnée peut se calculer simplement. Étant donné un point courant $\boldsymbol{x}_j$ et une direction de descente $\boldsymbol{d}_j$, le minimiseur de la fonction $\alpha \mapsto f(\boldsymbol{x}_j + \alpha\boldsymbol{d}_j)$ est

$$\alpha_j = -\frac{(\boldsymbol{A}\boldsymbol{x}_j - \boldsymbol{b})^{\mathrm{t}}\boldsymbol{d}_j}{\boldsymbol{d}_j\boldsymbol{A}\boldsymbol{d}_j}. \tag{4.60}$$

**Choix de la direction de descente $\boldsymbol{d}_j$**

Dans cette méthode, les directions de descente sont choisies de façon à ce qu'elles soient conjuguées par rapport à la matrice $\boldsymbol{A}$, c'est-à-dire $\boldsymbol{d}_i^{\mathrm{t}}\boldsymbol{A}\boldsymbol{d}_j = 0$ pour tout $i \neq j$. De telles directions sont construites par récurrence

$$\boldsymbol{d}_0 = -\nabla f(\boldsymbol{x}_0), \tag{4.61}$$

$$\boldsymbol{d}_{j+1} = -\nabla f(\boldsymbol{x}_{j+1}) + \gamma \boldsymbol{d}_j \quad \text{pour tout } j \geqslant 0 \quad \text{avec} \quad \gamma = \frac{\nabla f(\boldsymbol{x}_{j+1})^{\mathrm{t}} \nabla f(\boldsymbol{x}_{j+1})}{\nabla f(\boldsymbol{x}_j)^{\mathrm{t}} \nabla f(\boldsymbol{x}_j)}. \tag{4.62}$$

Grâce à cette construction, les propriétés suivantes sont garanties [Nocedal et Wright, 1999] :

- A l'itération $j$, le point $\boldsymbol{x}_j$ réalise le minimum de $f$ sur l'espace vectoriel de dimension $j$ :

$$\boldsymbol{x}_0 + \text{Vect}\left\{\nabla f(\boldsymbol{x}_0), \ldots, \nabla f(\boldsymbol{x}_{j-1})\right\}.$$

- Lorsque $j = n$, cet espace est $\mathbb{R}^n$, donc la méthode du gradient conjugué converge vers la solution en $n$ itérations au plus.

En pratique, l'algorithme est arrêté avant la fin des $n$ itérations lorsque $\|\nabla f(\boldsymbol{x}_j)\|_2^2 < \epsilon$ avec une tolérance $\epsilon > 0$.

**Préconditionnement**

D'après [Nocedal et Wright, 1999], la vitesse de convergence de l'algorithme de gradient conjugué peut être évalué par la relation

$$\|\boldsymbol{x}_j - \boldsymbol{x}^\star\| \leqslant \left(\frac{\sqrt{\kappa(\boldsymbol{A})} - 1}{\sqrt{\kappa(\boldsymbol{A})} + 1}\right)^{2j} \|\boldsymbol{x}_0 - \boldsymbol{x}^\star\|, \tag{4.63}$$

avec $\kappa(\boldsymbol{A})$ est le conditionnement de la matrice $\boldsymbol{A}$ défini par

$$\kappa(\boldsymbol{A}) = \frac{\sigma_{max}}{\sigma_{min}}, \tag{4.64}$$

où $\sigma_{max}$ et $\sigma_{min}$ sont respectivement la valeur singulière maximale et minimale de $\boldsymbol{A}$. Ainsi, plus les valeurs singulières de $\boldsymbol{A}$ sont rapprochées ($\kappa(\boldsymbol{A}) \approx 1$), plus l'algorithme de gradient conjugué converge rapidement.

L'utilisation d'un préconditionneur a pour objet d'accélérer la convergence des méthodes de descente en remplaçant le problème $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$ par le problème équivalent $\boldsymbol{C}^{-1}\boldsymbol{A}\boldsymbol{x} = \boldsymbol{C}^{-1}\boldsymbol{b}$, avec $\boldsymbol{C} \in \mathbb{R}^{n \times n}$ une matrice symétrique définie positive. Afin de remplir sa fonction, le préconditionneur $\boldsymbol{C}$ doit être une approximation de la matrice $\boldsymbol{A}$, afin de resserrer les valeurs singulières de la matrice $\boldsymbol{C}^{-1}\boldsymbol{A}$, tout en étant plus facile à inverser que cette dernière. La résolution de ce problème par la méthode du gradient conjugué, aussi appelée PCG (*Preconditioned Conjugate Gradient*), est donnée par l'algorithme 6.

Initialiser $j = 0$, $\boldsymbol{C} \in \mathbb{R}^{n \times n}$ symétrique définie positive et $\boldsymbol{x}_0 \in \mathbb{R}^n$, $\epsilon > 0$
Calculer le gradient $\boldsymbol{g} = \boldsymbol{A}\boldsymbol{x}_0 - \boldsymbol{b}$
Calculer $\boldsymbol{y}_0$ solution du système $\boldsymbol{C}\boldsymbol{y}_0 = \boldsymbol{g}_0$
Calculer la direction de descente $\boldsymbol{d}_0 = -\boldsymbol{y}_0$
**Tant que** ( $\|\boldsymbol{r}_k\| > \epsilon$ ) **faire**
  Calculer le pas optimal $\alpha_j = (\boldsymbol{g}_j^{\mathrm{t}}\boldsymbol{y}_j)/(\boldsymbol{d}_j^{\mathrm{t}}\boldsymbol{A}\boldsymbol{d}_j)$
  Mettre à jour $\boldsymbol{x}_{j+1} = \boldsymbol{x}_j + \alpha_j \boldsymbol{d}_j$
  Calculer le gradient $\boldsymbol{g}_{j+1} = \boldsymbol{g}_j + \alpha_j \boldsymbol{A}\boldsymbol{d}_j$
  Calculer $\boldsymbol{y}_{j+1}$ solution du système $\boldsymbol{C}\boldsymbol{y}_{j+1} = \boldsymbol{g}_{j+1}$
  Calculer $\beta_{j+1} = (\boldsymbol{g}_{j+1}^{\mathrm{t}}\boldsymbol{y}_{j+1})/(\boldsymbol{g}_j^{\mathrm{t}}\boldsymbol{y}_j)$
  Calculer la direction de descente $\boldsymbol{d}_{j+1} = -\boldsymbol{y}_j + \beta_{j+1}\boldsymbol{d}_j$
  $j = j + 1$
**Fait**

Algorithme 6: Algorithme PCG

#### 4.3.3.2 Mise en œuvre de la méthode PCG pour le calcul des directions primales

Dans le cadre de la méthode IPLS, la méthode PCG est appliquée pour calculer les directions primales en résolvant le système $\boldsymbol{H}\boldsymbol{d} = -\boldsymbol{g}$. Grâce au travail réalisé pour l'approche MMQ, nous savons que la matrice $\boldsymbol{B}$ définie à l'équation (4.31) est une approximation symétrique définie positive de $\boldsymbol{H}$. De plus, l'inversion de $\boldsymbol{B}$ est plus aisée que l'inversion de $\boldsymbol{H}$ du fait de sa structure bloc-diagonale. Par conséquent, $\boldsymbol{B}$ est utilisée comme préconditionneur dans la méthode PCG pour le calcul des directions primales.

### 4.3.4 Résultats

Comparons les performances de l'implémentation initiale de IPLS et son implémentation avec calcul des directions primales par la méthode MM ou par la méthode PCG avec le préconditionneur issus de la méthode MM. Ces approches sont également comparées à la méthode PCG avec préconditionneur calculé par factorisation de Cholesky incomplète (ICHOL) de la matrice $\boldsymbol{H}$ [Chouzenoux *et al.*, 2013]. Pour cela, reprenons les conditions de références définies dans la partie 3.2, et ajoutons une pénalisation spatiale, soit de type $\ell_2$ avec $\beta = 9$, soit de type $\ell_2 - \ell_1$ avec $\beta = 1$ et $\delta = 0.1$ comme dans le chapitre précédent.

Dans le tableau 4.3, on peut voir que pour un même résultat, l'implémentation incluant la méthode MM est plus rapide que l'implémentation initiale. Le gain de temps est de 58 % pour la pénalisation de type $\ell_2$, et de 65 % pour la pénalisation de type $\ell_2 - \ell_1$. Ces temps de calcul sont encore améliorés par l'introduction de la méthode du gradient conjugué préconditionné. Cette version de IPLS permet un gain de temps de 82 % par rapport à la version initiale avec pénalisation $\ell_2$, et 85 % avec pénalisation $\ell_2 - \ell_1$. Cela n'est pas encore aussi performant que la version utilisant la méthode du gradient conjugué avec préconditionneur ICHOL, mais les version MM et PCG+MM présentent l'avantage d'être adaptées à une implémentation en parallèle. En effet, la factorisation de Cholesky incomplète nécessite d'utiliser le formalisme de matrices creuses (*sparse*) qui est moins adapté à la parallélisation. Par ailleurs, en observant les nombres d'itérations effectués dans chaque cas, on remarque que l'augmentation attendue entre les calculs

| Type de pénalisation | L2 | | | |
|---|---|---|---|---|
| Accélération | sans | MM | PCG+MM | PCG+ICHOL |
| Temps CPU (en s) | 106 | 44.3 | 19.1 | 8.95 |
| Nombre d'itérations | 13.7 | 17.9 | 20.3 | 19.6 |
| Résidu ($\times 10^{-3}$) | 2.13 | 2.13 | 2.13 | 2.13 |
| EQMN | 3.43 % | 3.43 % | 3.43 % | 3.43 % |
| Type de pénalisation | L2L1 | | | |
| Accélération | sans | MM | PCG+MM | PCG+ICHOL |
| Temps CPU (en s) | 139 | 48.8 | 20.6 | 9.63 |
| Nombre d'itérations | 17.3 | 18.1 | 20.2 | 19.2 |
| Résidu ($\times 10^{-3}$) | 2.13 | 2.13 | 2.13 | 2.13 |
| EQMN | 2.69 % | 2.69 % | 2.69 % | 2.69 % |

TABLE 4.3 – Comparaison des différentes façons d'accélérer le calcul des directions primales.

exacts et approchés des directions primales est très modérée. Ainsi, les accélérations obtenues proviennent d'accélérations importantes de chaque itération, combinées à des augmentations faibles des nombres d'itérations. Enfin, il faut noter que si l'utilisation de la matrice $\boldsymbol{B}$ définie par l'équation (4.31) dans les algorithmes MM et PCG donne des résultats satisfaisants, il est possible qu'une formulation différente de cette matrice conduise à un algorithme plus rapide, devançant éventuellement la version PCG+ICHOL.

## 4.4 Conclusion

Dans ce chapitre la méthode IPLS a été accélérée en se focalisant sur l'étape de calcul des directions primales qui représente l'essentiel du temps de calcul. Sans pénalisation, une implémentation efficace de la méthode a été proposée, permettant un gain de temps important (38 % dans les conditions prises comme référence au chapitre 3. Avec pénalisation, l'algorithme IPLS a été modifié en remplaçant un calcul exact par un calcul approché basé sur la notion d'approximation majorante séparable permettant un gain de temps significatif (jusqu'à 85 %) tout en conservant les propriétés de convergence de l'algorithme initial. Ces modifications font d'IPLS une méthode particulièrement efficace comparativement aux méthodes existantes pour l'estimation des cartes d'abondances. Un autre avantage des modifications effectuées est l'apparition d'une structure de calcul qui se prête naturellement à une implémentation en parallèle. Le chapitre suivant concerne l'implémentation de la méthode IPLS sur GPU.

# Chapitre 5

# Accélération matérielle de la méthode de points intérieurs

## Sommaire



Ce chapitre est en partie tiré des publications [Legendre *et al.*, 2013c], [Legendre *et al.*, 2013b] et [Legendre *et al.*, 2013a].

## 5.1 Introduction

### 5.1.1 Le calcul parallèle

Avec des données de taille toujours plus grande, l'analyse d'images hyperspectrales demande des ressources informatiques importantes. Les développements effectués dans le chapitre précédent ont eu pour effet de réduire cette pression en réduisant le nombre de calculs nécessaires pour estimer des cartes d'abondances. Une autre façon de répondre à ce défi technique est l'utilisation de ressources matérielles plus adaptées aux calcul intensif. Le développement d'outils de calcul parallèle durant la dernière décennie répond à cette démarche [Asanovic *et al.*, 2006; Golub et Ortega, 2014]. Différentes options de parallélisation, chacune basée sur une architecture matérielle spécifique, sont désormais disponibles.

La figure 5.1 illustre les différences fondamentales entre ces solutions. Commençons par décrire de façon simplifiée la composition d'un processeur classique, aussi appelé CPU (*Central Processing Unit*). Celui-ci comprend trois éléments : une *unité de commande* qui lit les instructions arrivant, les décode puis commande l'unité de calcul ; une *unité de calcul* qui accomplit les tâches que lui a donné l'unité de commande ; et une *mémoire cache*, généralement séparée en plusieurs niveaux en fonction de leur proximité avec l'unité de calcul. Enfin, le CPU échange des données avec une mémoire de travail appelée mémoire RAM. Les différentes solutions de calcul parallèle sont toutes basées sur la multiplication de ces éléments de base. Tout d'abord, la plupart des processeurs récents sont dits *multi-cœurs*, ce qui signifie que les trois éléments qui le composent sont répliqués, en général deux ou quatre fois. Cela permet d'exécuter deux ou quatre fois plus de calculs en un temps donné. La réplication des trois composants de base permet un parallélisme de tâches aussi bien qu'un parallélisme de données. Un type différent de parallélisation est représenté par le modèle du GPU (*Graphics Processing Unit*). Celui-ci ne remplace pas le CPU mais est utilisé en complément de ce dernier. Sa structure comprend plusieurs blocs, chacun composé d'une unité de commande, une mémoire cache, et plusieurs unités de calcul. Typiquement, plusieurs centaines unités de calculs sont présentes sur un GPU. Cet ensemble échange des données avec une mémoire de travail qui lui est dédiée, appelée *mémoire globale*. Le fait d'avoir un plus grand nombre d'unités de calcul que d'unités de commande rend le GPU plus adapté au parallélisme de données. Une description plus détaillée de l'architecture matérielle du GPU est donnée dans la section 5.2.2. Enfin, le *calcul distribué* est une solution qui consiste à mettre en réseau plusieurs postes informatiques, c'est-à-dire multiplier l'ensemble du matériel afin d'augmenter la capacité de calcul. Ces postes peuvent éventuellement comprendre des CPU multi-cœurs ainsi que des GPU. Cette solution est la plus coûteuse mais elle devient de plus en plus accessible grâce au développement du *Cloud-Computing* qui permet d'ajuster le nombre de postes utilisés en fonction des besoins, alors que les autres solutions offrent une capacité de calcul fixe. La communication entre les postes est cependant nettement plus lente qu'entre différents processeurs d'un même poste.

Le domaine du traitement d'images est naturellement adapté à l'utilisation du GPU dont la fonction première est l'affichage d'images à l'écran. En imagerie hyperspectrale, plusieurs méthodes de traitement ont fait l'objet d'une implémentation GPU [Setoain *et al.*, 2008; Plaza *et al.*, 2011; Sanchez *et al.*, 2012]. Ce chapitre est consacré à l'utilisation d'un GPU, en complément d'un CPU, pour l'implémentation de la méthode de points intérieurs obtenue à l'issue des développements du chapitre précédent.

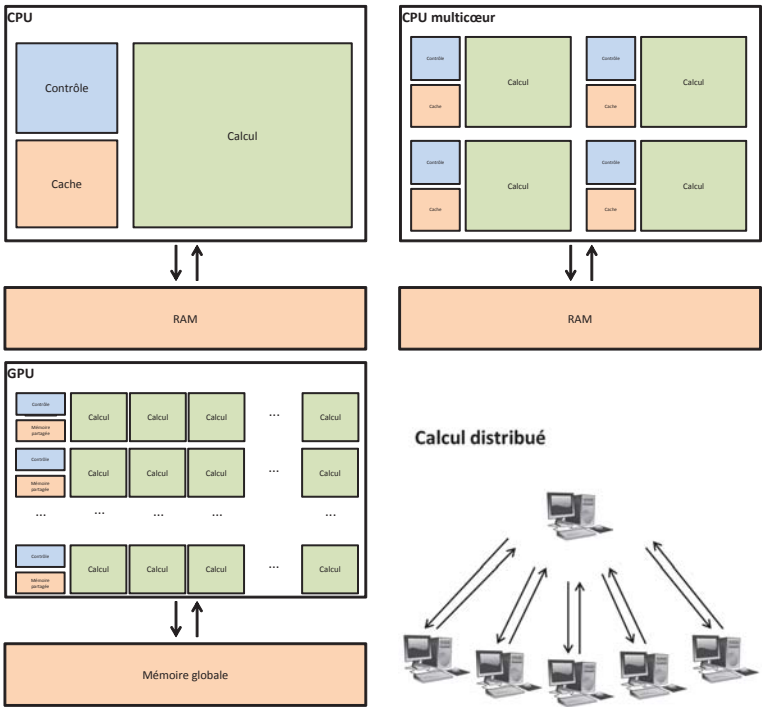



FIGURE 5.1 – Différentes solutions de calcul parallèle.

### 5.1.2 Histoire du GPU : du jeu-video au calcul scientifique

La première carte graphique grand publique apparaît sur le marché en 1981 [IBMC, 1981]. Elle est alors capable d'adresser des points avec une résolution de 320 colonnes sur 200 lignes en 4 couleurs différentes. Par la suite sortent une série de GPU améliorant le nombre de colonnes, de lignes, et de couleurs disponibles. Les années 1990 voient l'émergence des premières cartes capables d'afficher des éléments représentés en 3 dimensions. Le principal marché de ces composants est alors l'industrie du jeu vidéo qui demande toujours plus de performances graphiques. A ce moment, l'architecture des GPU est encore très rigide et uniquement vouée à l'affichage graphique : à chaque fonction réalisée par le GPU correspond un composant interne dédié et non programmable. Peu à peu, de nouvelles fonctions sont prises en charge par les GPU, comme la compression/décompression vidéo, qui permettent de réduire la charge du CPU. Pour cela, les composants internes du GPU deviennent paramétrables, on peut par exemple renvoyer les résultats du traitement en mémoire et pas directement à l'écran [Owens *et al.*, 2008]. Les années 2000 voient l'arrivée d'une nouvelle forme d'utilisation des GPU : le calcul scientifique. En effet les performances atteintes par les GPU grâce à l'industrie du jeu vidéo en font une alternative très économique aux super-calculateurs.

L'architecture massivement parallèle du GPU permet de réaliser des calculs simples sur un très grand nombre de données en un temps raisonnable. Lors des premières applications de ce type, la difficulté réside dans l'adaptation des données afin qu'elles puissent être traitées par le GPU, qui est encore à vocation graphique [Thompson *et al.*, 2002; Venkatasubramanian, 2003]. Rapidement, les constructeurs font évoluer leurs GPU afin de les rendre facilement programmable. Les composants internes sont unifiés et le GPU devient finalement un réseau d'unités de calcul toutes identiques, programmables, et fonctionnant en parallèle. Afin de faciliter la programmation et d'améliorer la portabilité des programmes, les constructeurs proposent aussi depuis 2006 des langages de programmation de haut niveau comme CUDA pour les GPU de marque Nvidia, ou CTM chez le constructeur AMD. Un langage unifié permettant la programmation sur la

plupart des GPU apparaît en 2008 avec OpenCL [Diaz *et al.*, 2012].

Si ces avancées récentes permettent de s'affranchir du fonctionnement exact du GPU lors de l'implémentation d'un programme, il n'en n'est pas moins nécessaire de connaître son schéma général de fonctionnement afin d'en extraire les règles qui permettront d'exploiter au mieux ses capacités.

## 5.2 La programmation sur GPU avec CUDA

### 5.2.1 Introduction à CUDA

La solution CUDA (*Compute Unified Device Architecture*) développée par Nvidia est un ensemble d'outils permettant une utilisation autre que graphique de certains GPU compatibles[1]. Au centre de cette architecture se trouve un langage de programmation appelé *C for CUDA* qui est une extension du langage *C* permettant toutes les manipulations qui seront décrites dans la partie 5.2.3. Cette solution comprend également des éléments plus bas niveau fonctionnant de manière transparente comme les *drivers* qui sont à l'interface entre le CPU et le GPU et gèrent les flux de données, ou encore un compilateur appelé *NVCC* capable de générer les programmes exécutables sur les GPU compatibles.

### 5.2.2 Architecture matérielle

Si l'architecture des CPU est à peu près figée de nos jours, celle des GPU évolue rapidement. La description donnée dans cette section concerne le modèle Tesla T10 de Nvidia, présent sur la carte Tesla C1060 qui est utilisée pour les tests de ce chapitre [Nvidia, 2008; Kirk *et al.*, 2010]. La figure 5.2 illustre l'architecture matérielle de ce GPU.

On y trouve 30 parties indépendantes que l'on appelle *streaming multiprocessors* (SM). On peut voir sur la figure 5.2 que l'on retrouve à l'intérieur d'un SM les trois parties d'un CPU : le contrôleur, les mémoires et les unités de calcul. La différence principale avec un CPU réside dans le nombre d'unités de calcul, on compte huit *streaming processor* (SP) par SM (il y en a donc 240 au total), et dans le fait qu'au même moment, chacun de ces SP exécute la même instruction sur des données différentes. Un SM est donc ce que l'on appelle un cœur SIMD (*Single Instruction Multiple Data*). En revanche, différentes instructions peuvent être exécutées au même moment sur les différents SM. On trouve également dans un SM des unités de calcul spécialisés : les *Special Function Units* (SFU) qui sont des unités de traitement complexes (trigonométrie, racine carré,...) et une unité de calcul double précision (64 bits) appelée DPU (*Double Precision Unit*). Il est en théorie possible d'effectuer des calculs dans toutes ces unités en parallèle. Cela dit il est rare que les données à traiter dans un programme soient exactement formatées pour utiliser toutes les capacités du GPU, un tel programme ne serait d'ailleurs adapté qu'à un seul modèle de GPU. Différents niveaux de mémoire sont également représentés : la mémoire globale, d'une capacité de 4 giga-octets (Go), située sur la carte graphique mais extérieure au GPU est accessible depuis toutes les unités de calcul avec un temps d'accès de plusieurs centaines de cycles d'horloge ; la mémoire partagée présente au niveau de chaque SM, d'une capacité de 16 kilo-octets (Ko) par SM, est accessible seulement par les unités de

[1] http://www.nvidia.com/object/cuda_home_new.html

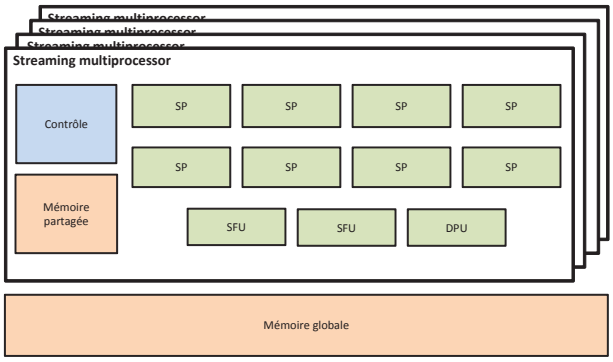



FIGURE 5.2 – Architecture matérielle d'un GPU.

calcul du même SM avec un temps d'accès de quelques cycles d'horloge. Afin de maximiser le nombre d'unités de calcul utilisées, on utilise des outils de programmation fournis par les constructeurs qui convertissent un code de haut niveau en une suite d'instruction adaptée à chaque GPU.

### 5.2.3 Architecture logicielle

Bien qu'il soit indispensable de connaître l'architecture matérielle pour une implémentation GPU efficace, celle-ci est gérée de façon transparente par CUDA. Seule l'architecture logicielle est directement utilisée par le programmeur. Celle-ci est basée sur la programmation en C. Seules les parties du programme devant être exécutées sur le GPU sont différentes. Celles-ci sont écrites dans des *noyaux* qui sont des fonctions dont la particularité est qu'ils sont exécutés un grand nombre de fois en parallèle. Lors de l'appel d'un noyau, deux paramètres sont choisis : le nombre de *blocs*, et le nombre de *threads par bloc*. Cela définit un nombre total de threads correspondant au nombre de versions du noyau qui seront exécutées. Lors de l'exécution, un indice unique est attribué à chaque thread. Lors de l'écriture du noyau, des mots-clés définis par CUDA permettent d'utiliser cet indice afin que chaque thread exécute une tache différente. Cela permet en particulier à chaque thread de traiter des données différentes [Nvidia, 2012; Sanders *et al.*, 2011].

La figure 5.3 illustre le déroulement d'un programme contenant plusieurs noyaux. Lorsqu'un noyau est appelé, il est exécuté à travers un grand nombre de threads qui sont exécutés sur le GPU. Lorsque cette exécution est terminée, le programme reprend sur le CPU. Pendant la durée d'utilisation du GPU, le CPU est disponible pour un autre programme ou des instructions du même programme ne nécéssitant pas les résultats issus du noyau. Les instructions de transfert de données entre la mémoire du CPU et celle du GPU ne sont pas représentées dans ce schéma. Elles sont comprises dans les parties « Instructions CPU ». Les codes 5.1 et 5.2 donne un exemple simple de noyau CUDA et de son utilisation au sein du programme pour effectuer la somme élément par élément de deux tableaux, le résultat étant stocké dans un troisième tableau. Les parties en bleu sont les mots-clés (noms de constantes ou de fonctions réservés) définis par CUDA et qui viennent s'ajouter aux mots-clés existants dans le langage C.

```
1
2  __global__ void add(int *a, int *b, int *c)
3  {
4    // Calcul de l'indice unique de chaque thread grace aux mots-cles de
     CUDA
5    int indice = threadIdx.x + blockIdx.x * blockDim.x
```

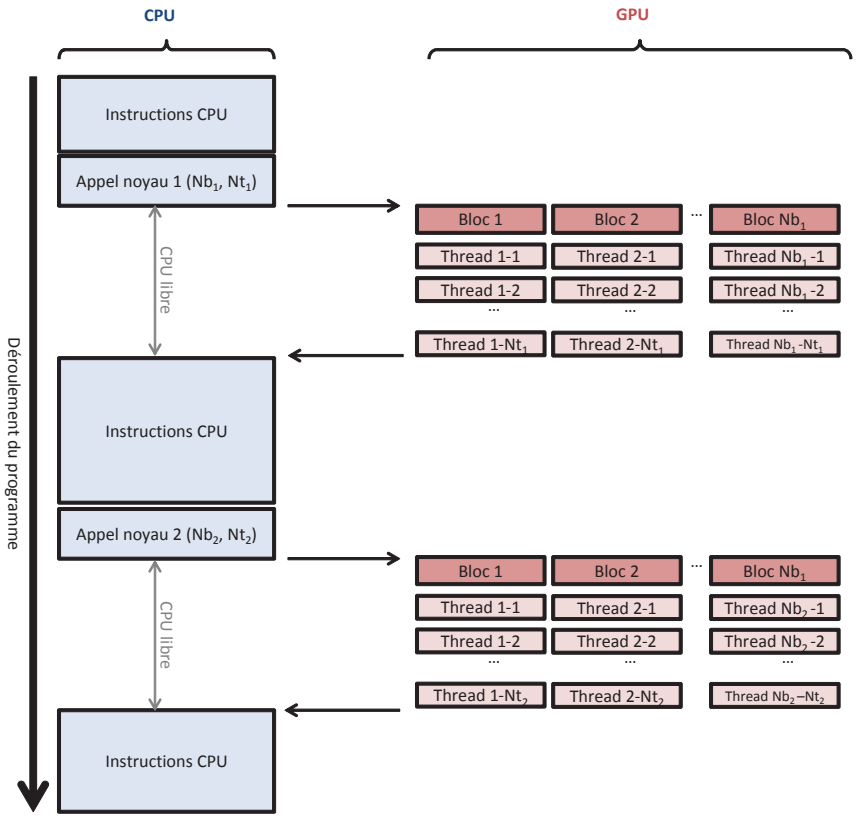



FIGURE 5.3 – Déroulement d'un programme utilisant le GPU.

```
6
7     // Chaque thread realise la somme d'un element de a et d'un element de
      b
8     c[indice] = a[indice] + b[indice];
9   }
```

Listing 5.1 – Noyau CUDA réalisant la somme de deux tableaux

```
1
2   // Situation initiale :
3   //   - a, b et c sont 3 tableaux de N entiers definis precedement dans le
        programme
4   //   - nbBlocs et nbTheardsParBloc sont definis precedement dans le
      programme tels que N = nbBlocs * nbTheardsParBloc
5
6   // Allocation de memoire globale sur le GPU
7   cudaMalloc(&a_gpu, N*sizeof(int));
8   cudaMalloc(&b_gpu, N*sizeof(int));
9   cudaMalloc(&c_gpu, N*sizeof(int));
10
11  // Copie des tableaux a et b du CPU vers le GPU
12  cudaMemcpy(a_gpu, a, N*sizeof(int), cudaMemcpyHostToDevice);
13  cudaMemcpy(b_gpu, b, N*sizeof(int), cudaMemcpyHostToDevice);
14
15  // Lancement du noyau calculant la somme de a et b sur le GPU
16  add<<<nbBlocs,nbTheardsParBloc>>>(a_gpu, b_gpu, c_gpu);
17
18  // Copie du tableau c du GPU vers le CPU
19  cudaMemcpy(c, c_gpu, N*sizeof(int), cudaMemcpyDeviceToHost);
20
21  // Liberation de la memoire globale du GPU
```

```
22  cudaFree(a_gpu);
23  cudaFree(b_gpu);
24  cudaFree(c_gpu);
```

Listing 5.2 – Utilisation d'un noyau CUDA

### 5.2.4 Règles d'exécutions

Le nombre de threads lancés par un programme peut être bien plus grand que le nombre d'unités de calcul disponibles sur le GPU. Ils ne peuvent alors pas être tous exécutés en parallèle au même moment. Un ordonnenceur réparti automatiquement la charge de travail sur les différents éléments du GPU. L'exécution des threads, regroupés en blocs, se fait sur les unités de calcul, regroupées en SM, en obéissant aux règles suivantes [Kirk *et al.*, 2010] :

- 1 SM peut héberger au maximum 8 blocs ou 1024 threads
- Exécution d'un bloc sur un SM : par *warp* de 32 threads. Ainsi, $8 \times 32$ threads peuvent être en cours d'exécution au même moment sur un SM contenant seulement 8 unités de calcul. Cela permet de masquer certain temps de latence dus au temps d'accès à la mémoire globale.
- Le nombre de bloc par SM est automatiquement réduit si une des mémoires est insuffisante (globale ou partagée). Ainsi, le transfert d'une donnée dans la mémoire partagée afin d'y accéder plus rapidement peut avoir pour effet d'augmenter la mémoire nécessaire à l'exécution de chaque thread, donc de diminuer le nombre de blocs exécutés simultanément sur chaque SM, limitant ainsi le masquage des temps de latence, et finalement augmentant le temps de calcul total du noyau. **La mémoire partagée étant présente en quantité limitée, il convient de l'utiliser avec parcimonie** pour maximiser le nombre de blocs exécutés simultanément.
- En cas de structure conditionnelle (*if, then, else*), si les deux conditions sont vérifiées par des threads différents d'un même warp, alors les instructions liées aux deux conditions sont exécutées par tous les threads de ce warp. Des opérations inutiles sont alors effectuées. Leurs résultats sont ignorés, mais cela a pour effet d'augmenter le temps d'exécution. **Les structures conditionnelles sont donc à éviter autant que possible dans un noyau.**
- L'accès à la mémoire globale ce fait par demi-warp avec un cache de 128 octets. Afin de réduire le nombre de lecture en mémoire et donc d'accélérer l'exécution, **les threads d'un demi-warp doivent lire des données consécutives en mémoire.**

## 5.3 Inversion d'un grand nombre de systèmes linéaires sur GPU

### 5.3.1 Contexte de l'étude

Cette section résume une étude effectuée dans le cadre d'un contrat de collaboration avec l'Agence Spatiale Européenne dont les résultats détaillés sont présentés dans l'article [Legendre *et al.*, 2013c] donné en annexe B.

**Pour** $k$ **de** $1$ **à** $n-1$ **faire**
  **Pour** $j$ **de** $k+1$ **à** $n$ **faire**
    $v \leftarrow A_{jk}/A_{kk}$
    **Pour** $i$ **de** $j$ **à** $n$ **faire**
      $A_{ij} \leftarrow A_{ij} - A_{ik} \times v$
    **Fin Pour**
    $A_{jk} \leftarrow v$
  **Fin Pour**
**Fin Pour**
La partie inférieur de $\boldsymbol{A}$ a été remplacé part la partie inférieur de $\boldsymbol{L}$
La diagonale de $\boldsymbol{A}$ a été remplacée par $\boldsymbol{D}$

Algorithme 7: Factorisation $\boldsymbol{LDL}^{\mathrm{t}}$ d'une matrice $\boldsymbol{A} \in \mathbb{R}^{n \times n}$ symétrique inversible

La mission spatiale Gaia [Lindegren *et al.*, 2008] de l'Agence Spatiale Européenne a pour objet de créer un catalogue astrométrique aussi précis et complet que possible. Plus d'un milliard d'astres seront observés à de multiples reprises durant les 5 ans de la durée de vie nominale du satellite, lancé le 19 décembre 2013, produisant plus de 100 TO de données brutes à traiter. Ce traitement est effectué par le consortium baptisé DPAC comprenant 400 personnes et 6 centres de calcul en Europe. Au cœur de ce traitement ce trouve l'algorithme AGIS (*Astrometric Global Iterative Solution*) [Lindegren *et al.*, 2012]. Pour chaque astre, 5 paramètres sont estimés (2 de position, 2 de vitesse et un de parallaxe) par maximum de vraisemblance. Ce traitement requiert l'inversion d'un système linéaire définit par une matrice de taille $5 \cdot 10^9 \times 5 \cdot 10^9$, ce qui est impossible en pratique. Des approximations conduisent à une décomposition du système en $10^9$ systèmes linéaires symétriques de taille $5 \times 5$. L'objectif de cette étude est de proposer une implémentation GPU efficace pour la résolution d'un grand nombre de systèmes linéaires symétriques de faible taille.

### 5.3.2 Algorithme d'inversion

Notons un système linéaire

$$\boldsymbol{Ax} = \boldsymbol{b} \tag{5.1}$$

avec $\boldsymbol{A} \in \mathbb{R}^{n \times n}$ une matrice symétrique, $\boldsymbol{b} \in \mathbb{R}^n$ un vecteur, et $\boldsymbol{x} \in \mathbb{R}^n$ un vecteur inconnu à calculer. La taille des données est ici $n = 5$.

L'algorithme d'inversion de système proposé est basé sur la factorisation $\boldsymbol{LDL}^{\mathrm{t}}$ [Golub et Van Loan, 1996]. En effet, la matrice symétrique $\boldsymbol{A}$ peut s'écrire $\boldsymbol{A} = \boldsymbol{LDL}^{\mathrm{t}}$, avec $\boldsymbol{L}$ une matrice triangulaire inférieure unitaire (contenant des 1 sur sa diagonale), et $\boldsymbol{D}$ une matrice diagonale. Cette factorisation est effectuée par l'algorithme 7. Le système à résoudre devient alors $\boldsymbol{LDL}^{\mathrm{t}}\boldsymbol{x} = \boldsymbol{b}$, dont la résolution est effectuée en trois étapes par l'algorithme 8.

**Pour** $i$ **de** $2$ **à** $n$ **faire**
  **Pour** $j$ **de** $1$ **à** $i-1$ **faire**
    $b_i = b_i - L_{ij} b_j$
  **Fin Pour**
**Fin Pour**
**Pour** $i$ **de** $1$ **à** $n$ **faire**
  $b_i = b_i / D_{ii}$
**Fin Pour**
**Pour** $i$ **de** $n-1$ **à** $1$ **faire**
  **Pour** $j$ **de** $i+1$ **à** $n$ **faire**
    $b_i = b_i - L_{ji} b_j$
  **Fin Pour**
**Fin Pour**
Le second membre $\boldsymbol{b}$ a été remplacée par la solution $\boldsymbol{x}$

Algorithme 8: Inversion du système $\boldsymbol{L}\boldsymbol{D}\boldsymbol{L}^{\mathrm{t}}\boldsymbol{x} = \boldsymbol{b}$

### 5.3.3 Implémentation GPU

Une implémentation CPU a d'abord été réalisée, puis quatre implémentations GPU, chacune étant une amélioration de la précédente.

**Implémentation CPU**

Ce travail est réalisée en langage C. Il permet de charger un grand nombre de couples matrices-vecteurs à partir d'un fichier texte, et de les traiter avec les algorithmes 7 et 8 afin d'obtenir les résultats souhaités. Les matrices considérées étant symétriques, seules leurs parties inférieures sont stockées en mémoire. L'organisation des données en mémoire est donné en figure 5.4 sous l'appellation *par pixel*.

**Implémentation GPU version 1 : réplique de l'implémentation CPU**

La partie résolution de systèmes de l'implémentation CPU est portée sur GPU. Pour cela, les couples matrice-vecteurs sont transférés de la mémoire RAM du CPU vers la mémoire globale du GPU. Puis un noyau implémentant les algorithmes 7 et 8 pour la résolution d'un système linéaire est exécuté à travers autant de threads qu'il y a de systèmes à résoudre. Enfin, les vecteurs solutions sont transférés de la mémoire globale du GPU vers la mémoire RAM du CPU.

**Implémentation GPU version 2 : réorganisation des données en mémoire**

Comme nous l'avons vu dans la section 5.2.4, les accès successifs à la mémoire globale du GPU doivent se faire sur des adresses mémoires contiguës pour minimiser les temps de latence. Or, les accès successifs en mémoire sont faits par la même instruction dans des threads d'indice successifs. On en déduit que les éléments de même indice, pour les matrices ou les vecteurs, doivent être stockés dans des adresses mémoires contiguës, comme illustré dans la figure 5.4

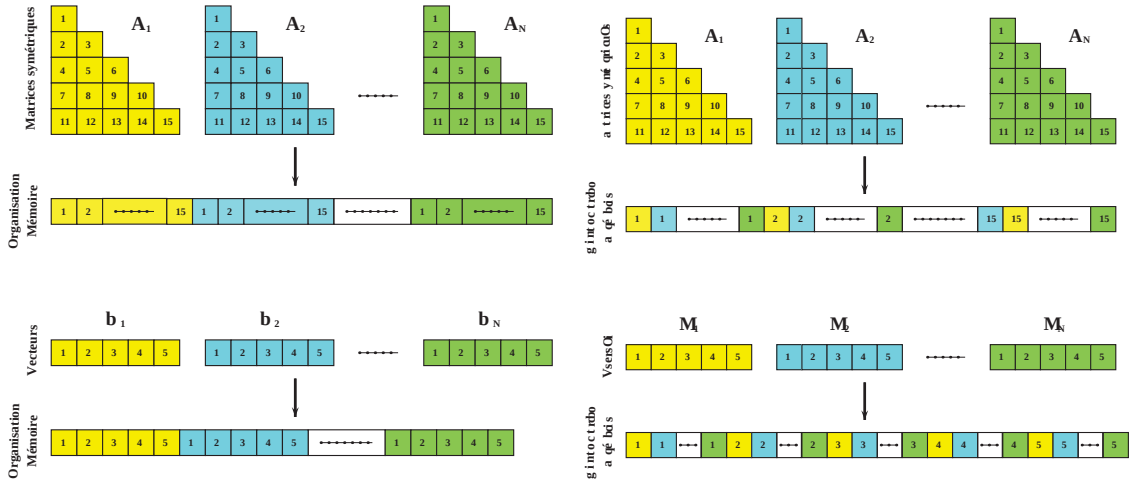



FIGURE 5.4 – Organisation des données en mémoire. A gauche : l'organisation *par pixel* utilisée pour les implémentations CPU et GPU version 1. A droite : l'organisation *par élément*pour les implémentations GPU versions 2, 3 et 4.

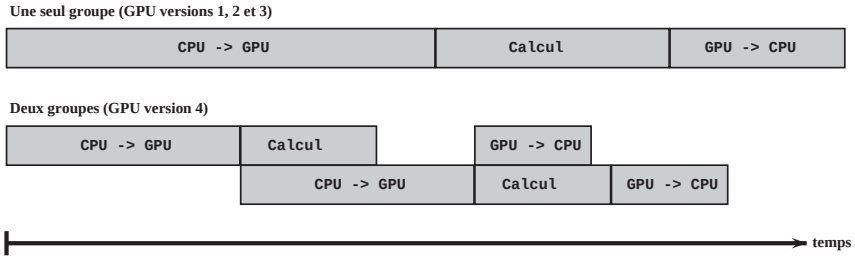



FIGURE 5.5 – Avantage de la séparation des données de deux groupes avec l'utilisation des flux CUDA.

avec l'organisation *par élément*. Cette organisation des données en mémoire est adoptée tout au long du programme, sur la partie CPU comme sur la partie GPU.

**Implémentation GPU version 3 : utilisation de la mémoire hôte non-paginée**

Lors de tout transfert de données entre les mémoires du CPU et du GPU, une copie intermédiaire est automatiquement réalisée sur la mémoire hôte non-paginée (*pinned-memory*). Il s'agit d'un espace alloué sur la mémoire RAM du CPU dont on force l'adresse à rester fixe, c'est-à-dire que le système d'exploitation ne peut pas prendre la décision de déplacer des données de la mémoire hôte non-paginée comme cela peut être le cas avec des données stockées de façon classiques. CUDA permet au programmeur d'allouer directement les données dans la mémoire hôte non-paginée du CPU, ce qui accélère les transferts de données avec la mémoire du GPU.

**Implémentation GPU version 4 : séparation des données en deux groupes**

Cette version exploite la capacité qu'à la carte graphique utilisée de réaliser simultanément un transfert de données entre les mémoires du CPU et du GPU, et l'exécution d'un noyau sur le GPU. Cela est permis par l'utilisation des flux CUDA (*CUDA Streams*). En séparant les données en deux groupes, chacun constitué d'un grand nombre de couples matrice-vecteur, il est possible de gagner du temps en réalisant simultanément un transfert pour un groupe et des calculs pour l'autre, comme illustré dans la figure 5.5.

| **Temps CPU** | **136 ms** | | | |
|---|---|---|---|---|
| **Temps GPU** | **version 1** | **version 2** | **version 3** | **version 4** |
| Transfert CPU vers GPU | 7.7 ms | 7.7 ms | 3.4 ms | n.a. |
| Résolution | 21.6 ms | 3.5 ms | 3.5 ms | n.a. |
| Transfert GPU vers CPU | 2.4 ms | 2.4 ms | 1.0 ms | n.a. |
| **Total** | **31.7 ms** | **13.6 ms** | **7.9 ms** | **6.2 ms** |
| **Gain** | **4.4** | **10.0** | **17.2** | **21.9** |

TABLE 5.1 – Temps de calcul et gains pour toutes les implémentations GPU réalisées en simple précision

| **Temps CPU** | **136 ms** | | | |
|---|---|---|---|---|
| **Temps GPU** | **version 1** | **version 2** | **version 3** | **version 4** |
| Transfert CPU vers GPU | 15.0 ms | 15.0 ms | 6.7 ms | n.a. |
| Résolution | 21.2 ms | 5.5 ms | 5.5 ms | n.a. |
| Transfert GPU vers CPU | 4.4 ms | 4.4 ms | 1.7 ms | n.a. |
| **Total** | **40.6 ms** | **24.9 ms** | **13.9 ms** | **11.1 ms** |
| **Gain** | **3.3** | **5.5** | **9.8** | **12.3** |

TABLE 5.2 – Temps de calcul et gains pour toutes les implémentations GPU réalisées en double précision

### 5.3.4 Résultats

Un ensemble de 250531 systèmes linéaires de taille $5 \times 5$ fourni par l'Agence Spatiale Européenne est utilisé pour mesurer le gain apporté par chaque version implémentée sur GPU par rapport à la version CPU. Le matériel utilisé est le même qu'aux chapitres précédents, une station de travail Dell Precision T7400 contenant deux processeurs Intel Xeon X5472 (quatre unités de calcul chacun) cadencés à 3 GHz et 16 GO de mémoire RAM. Cette fois la carte graphique embarquée sur ce poste est également utilisée. Il s'agit d'une carte Tesla C1060 contenant 240 unités de calcul simple précision et 30 double précision cadencées à 1.3 GHz et 4 GO de mémoire globale. Il faut noter que la version CPU n'exploite qu'un seul des 8 cœurs disponibles.

Les résultats en simple précision sont présentés dans le tableau 5.1, et en double précision dans le tableau 5.2. Pour l'implémentation GPU la plus optimisée, le facteur de gain en temps de calcul est de 21.9 en simple précision et 12.3 en double précision. On montre que ce gain ne peut pas être amélioré pour cette application car de facteur limitant de la version 4 est le temps de transfert des données entre le CPU et le GPU, qui est incompressible. La différence de gain entre simple et double précision s'explique par la présence d'unités de calcul simple précision en plus grand nombre.

Dans ces implémentations, la mémoire partagée du GPU, présente au sein de chaque SM, dont l'accès est plus rapide que la mémoire globale, n'est pas utilisée. Or, lors de l'exécution d'un noyau, il est possible d'ordonner à chaque thread de copier les données dont il a besoin dans la mémoire partagée afin de réaliser les calculs plus rapidement, puis de copier les résultats dans la mémoire globale. Cette implémentation a été réalisée mais a abouti à une augmentation du temps de calcul. Dans cette situation, bien que chaque thread soit exécuté plus rapidement, la taille limités de la mémoire partagée empêche l'exécution de nombreux threads simultanément.

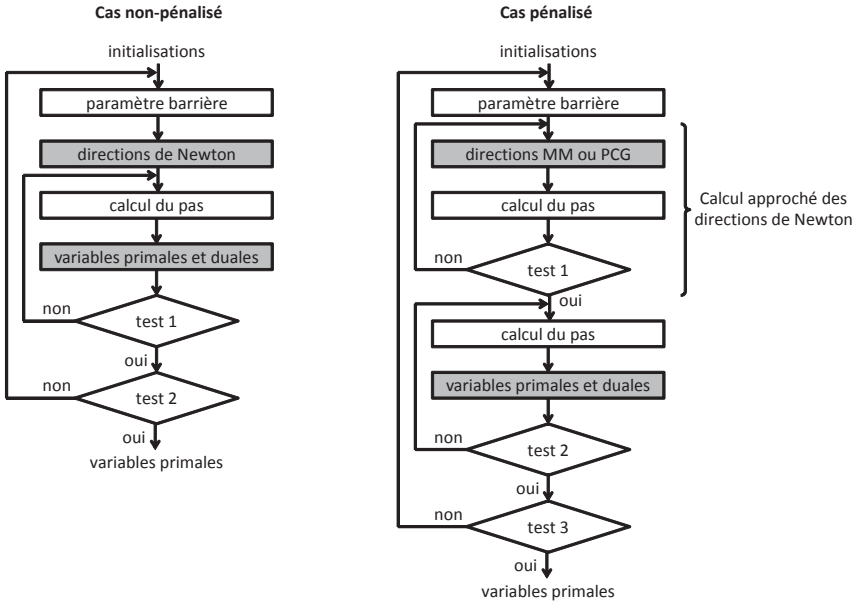



FIGURE 5.6 – Organigramme de la méthode primale-duale de points intérieurs. Les étapes grisées sont réalisées sur la GPU alors que les autres sont réalisées partiellement sur le GPU puis finalisées sur le CPU.

## 5.4 Implémentation de l'algorithme de points intérieurs

Comme nous l'avons vu au chapitre 4, l'étape la plus coûteuse en temps de calcul dans la méthode de points intérieurs dont l'organigramme est donné dans la figure 5.6 est le calcul des directions de Newton, et en particulier des directions primales. Nous avons montré que dans le cas non-pénalisé cette étape se réduit à la résolution d'un grand nombre de systèmes linéaires symétriques. Dans le cas pénalisé, l'utilisation d'une approche MM a également permis remplacer cette étape par une séquence de résolution d'un grand nombre de systèmes linéaires symétriques. Dans ces deux cas, il est donc possible d'utiliser l'implémentation GPU qui vient d'être présentée.

Cette section s'intéresse à l'implémentation complète de la méthode de points intérieurs sur GPU. Excepté le calcul des direction primale qui nécessite l'inversion d'un système de grande taille, les calculs effectués dans cette méthodes sont simples, il ne s'agit que d'opérations de base sur des matrices. On ne détaillera donc pas leur implémentation qui est réalisée suivant le modèle de la version 3 présentée dans la section précédente. Cette section porte sur les avantages et inconvénients de la version *par pixel* et de la version *par image* présentées dans la section 4.2 dans le cas non-pénalisé. En effet, si la version *par pixel* est nettement moins performante dans une implémentation CPU réalisée avec Matlab, elle présente un potentiel de parallélisation plus important.

### 5.4.1 Implémentation *par pixel*

Il est naturel de penser que c'est en rendant le problème totalement parallélisable que le GPU serait le mieux exploité. Cela est possible avec l'implémentation *par pixel* qui réalise $N$ minimisations indépendantes des critères liés aux $N$ pixels :

$$F_n(\boldsymbol{a}_n) = \|\boldsymbol{y}_n - \boldsymbol{S}\boldsymbol{a}_n\|_2^2, \quad \forall n = 1, \ldots, N. \tag{5.2}$$

De cette façon, $N$ threads peuvent être lancés sur le GPU, chacun exécutant entièrement l'algorithme d'estimation des abondances sur un pixel de l'image. Néanmoins cette méthode présente un inconvénient. En effet, comme on l'a vu dans la section 5.2.4, les structures conditionnelles de type *« if, then, else »* au sein d'un noyau sont à éviter car dans le modèle défini par CUDA les threads sont organisés par groupes de 32, appelés warps, se comportant comme autant d'unités SIMD. En cas de structure conditionnelle, si les deux conditions sont vérifiées par des threads différents d'un même warp, alors les instructions liées aux deux conditions sont exécutées par tous les threads de ce warp. Des opérations inutiles dont les résultats sont ignorés sont alors effectuées, ce qui a pour effet d'augmenter le temps d'exécution. Ainsi, lors de l'exécution de l'algorithme de points intérieurs, le temps total est fixé par le pixel dont le traitement nécessite le plus d'itérations dans chaque groupe de 32 pixels consécutifs.

Dans cette version le CPU n'est utilisé que pour initier et terminer le programme. Tous les calculs se déroulent sur le GPU sans transfert de données intermédiaires. Notons également qu'une telle implémentation est impossible lorsque la pénalisation spatiale est prise en compte car le traitement n'est alors pas indépendant sur chaque pixel.

### 5.4.2 Implémentation *par image*

L'idée est de revenir au problème initial, avec ou sans pénalisation, et de minimiser le critère global défini par l'équation

$$F(\boldsymbol{A}) = \|\boldsymbol{Y} - \boldsymbol{S}\boldsymbol{A}\|_F^2 + \beta R(\boldsymbol{A}) \tag{5.3}$$

en tirant parti au mieux des caractéristiques différentes du CPU et du GPU. Le CPU est utilisé pour implémenter la structure de l'algorithme, gérant le lancement de chaque étape en fonction des résultats des tests d'arrêt. Le GPU est utilisé au sein de chaque étape afin d'en accélérer l'exécution.

Cette version présente l'avantage de ne pas introduire de calculs inutiles, cependant certaines étapes de l'algorithme nécessitent des transferts de données entre la mémoire du CPU et celle du GPU, ce qui ralentit leur exécution. On peut distinguer deux types d'étapes : celles pour lesquelles une parallélisation totale est possible car elles contiennent des calculs indépendants sur chaque pixel (en gris dans la figure 5.6), ces étapes ne demandent aucun transfert de données ; et celles dont le résultat est une variable unique pour l'image entière (en blanc dans la figure 5.6). C'est le cas pour le calcul du paramètre barrière, du pas de Newton, et des différents critères d'arrêt. Ce type d'étape est appelé *réduction* et n'est effectuée que partiellement sur le GPU.

Une réduction est une opération associative et commutative appliquée à un ensemble d'éléments et n'en retournant qu'un seul, comme le calcul d'une somme ou l'extraction d'un minimum. Cette opération peut être effectuée avec une stratégie itérative au cours de laquelle une itération réduit le nombre d'éléments par un facteur deux en appliquant l'opération à toutes les paires d'éléments. Le processus se termine lorsqu'il ne reste qu'un seul élément. Le nombre d'opérations indépendantes est élevé durant les premières itérations d'une réduction et diminue d'un facteur deux à chaque itération. C'est pourquoi le plus efficace est d'effectuer une réduction partielle (premières itérations) sur le GPU, puis de transférer le résultat partiel dans la mémoire

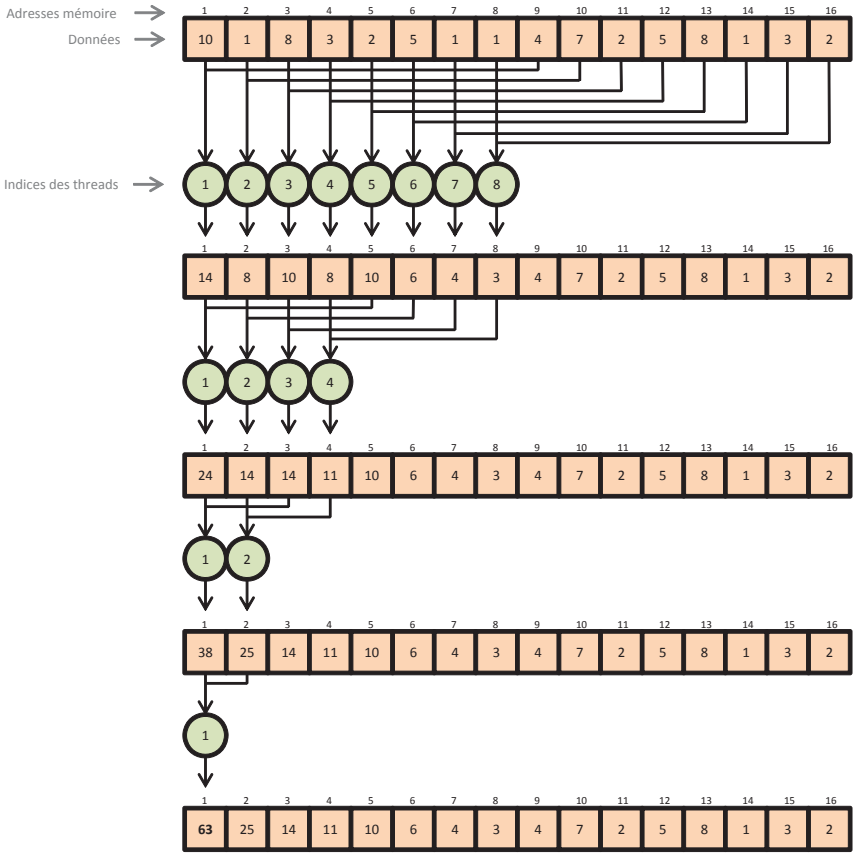



FIGURE 5.7 – Fonctionnement d'un bloc de threads pendant une étape de réduction. Exemple de la somme des éléments d'un tableau.

du CPU pour y terminer le calcul [Kirk *et al.*, 2010]. Pour implémenter cette procédure, on utilise l'organisation des theads par bloc. En effet, CUDA offre la possibilité de synchroniser tous les threads d'un même bloc, mais pas les threads de différents blocs. Hors, une synchronisation est indispensable entre deux itérations. Ainsi, chaque bloc effectue une réduction sur une partie des éléments. Le nombre de blocs choisis détermine le nombre d'éléments issus de la réduction sur GPU. Ces éléments sont ensuite transférés dans la mémoire du CPU où le calcul est terminé. En pratique, un bon compromis a été trouvé en fixant le nombre de blocs à 256. Le nombre de threads par bloc est déduit afin de disposer d'au moins un thread par pixel. La figure 5.7 illustre le travail effectué par un bloc pour effectuer la somme des éléments d'un tableau. Notons qu'un thread n'effectue pas la somme de deux éléments consécutifs, mais d'un élément de la première moitié du tableau et un élément de la deuxième moitié. Cela permet à des threads successifs d'accéder à des adresses mémoires successives afin de mutualiser les accès mémoire comme expliqué en section 5.2.4.

## 5.5 Résultats sur images simulées

### 5.5.1 Sans pénalisation : choix de la meilleur implémentation GPU

Les implémentations *par pixel* et *par image* ont chacune leurs avantages et lesus inconvéniants. Dans cette section, des tests sont réalisés pour confronter ces deux approches. Commençons par mesurer leurs temps de calcul dans la situation prise comme référence dans ll chapitre 3.

| Implémentation | *par pixel* | *par image* |
|---|---|---|
| Temps de calcul (s) | 0.04 | 0.08 |
| NMSE (%) | 9.58 | 9.88 |
| Résidu ($\times 10^{-3}$) | 2.13 | 2.13 |

TABLE 5.3 – Comparaison des deux implémentations proposées dans la situation de référence.

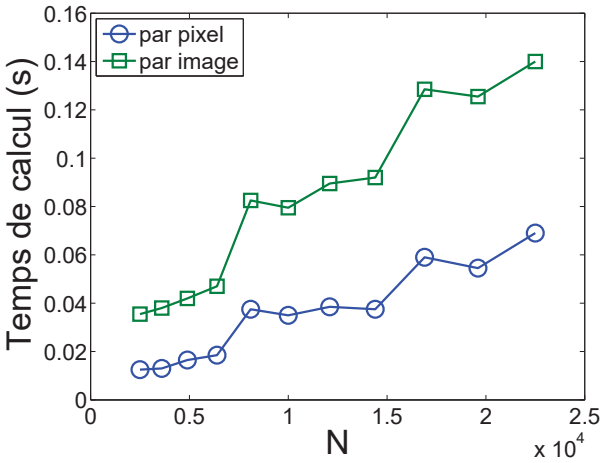



FIGURE 5.8 – Influence du nombre de pixels sur les temps de calcul de la méthode IPLS sur GPU pour les deux implémentations proposées.

Les résultats sont donnés dans le tableau 5.3. On voit que le la version *par pixel* est la plus rapide pour un résultat similaire en terme de qualité de reconstruction. Afin de compléter cette analyse, cette comparaison est répliquée pour différents nombres de pixels dans la figure 5.8 et différents nombres de endmembers dans la figure 5.9. Tous ces tests montrent l'avantage de la version *par pixel* qui est systématiquement plus rapide, alors qu'elle était bien plus lente dans Matlab. Le traitement indépendant de chaque pixel permettant une parallélisation de l'ensemble de l'algorithme est donc l'approche la plus appropriée pour l'utilisation du GPU en l'abscence de pénalisation spatiale.

### 5.5.2 Sans pénalisation : comparaison CPU/GPU

Dans cette section sont comparées les meilleures implémentations sur CPU et sur GPU. La version CPU est réalisée avec Matlab et met en œuvre l'implémentation *par image* avec calcul des directions de Newton *par bloc*. Aucune restriction n'est imposée à Matlab quant à l'utilisation des 8 cœurs disponibles, un grand nombre d'opérations matricielles sont donc réalisées automatiquement en parallèle au sein du CPU. La version GPU est réalisée avec CUDA et met en œuvre l'implémentation *par pixel*, certaines étapes sont exécutées sur le CPU et n'utilisent qu'un seul des 8 cœurs disponibles. Les facteurs de gain entre le CPU et le GPU sont donc à prendre à titre indicatif uniquement car ils sont issus de deux langages au fonctionnement différent et ne sont valables que pour le matériel utilisé dans cette thèse. Une implémentation GPU de la méthode ADMM a également été réalisée sur le modèle de l'implémentation GPU de la méthode IPLS *par image*. Le tableau 5.4 donne les temps de calcul des méthodes IPLS et ADMM sur CPU et sur GPU dans la situation de référence. Si l'accélération obtenue par l'utilisation du GPU est intéressante pour les deux méthodes, elle nettement est supérieure dans le cas d'IPLS. Ceci peut s'expliquer par le fait qu'ADMM nécéssite un grand nombre d'itérations avant convergence.

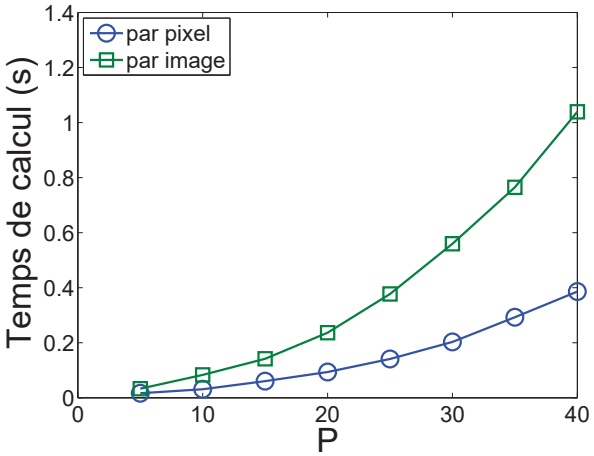



FIGURE 5.9 – Influence du nombre de endmembers sur les temps de calcul de la méthode IPLS sur GPU pour les deux implémentations proposées.

| Méthode | IPLS | | ADMM | |
|---|---|---|---|---|
| Implémentation | CPU | GPU | CPU | GPU |
| Temps de calcul (s) | 1.44 | 0.04 | 2.55 | 0.53 |
| Facteur de gain | 36 | | 4.8 | |

TABLE 5.4 – Comparaison des implémentations CPU et GPU pour les méthodes IPLS et ADMM dans la situation de référence.

L'implémentation *par image* utilisée pour rester fidèle au code Matlab utilisé n'est donc pas la meilleure option pour cette méthode. Elle génère un grand nombre d'aller-retours entre CPU et GPU, ce qui empêche le GPU d'être utilisé pleinement. Une implémentation *par pixel* pour la méthode ADMM donnerait probablement de meilleurs résultats. Sur la figure 5.10, on voit que le gain reste relativement stable avec le nombre de pixel. On peut en déduire que même pour une image de faible taille le nombre de thread lancé sur le GPU est suffisant pour que l'ordonnanceur ait la possibilité d'optimiser automatiquement l'exécution. Un plus grand nombre de threads n'apporte donc par d'accélération supplémentaire. De plus, la figure 5.11 montre que lorsque le nombre de endmembers augmente, l'accélération reste stable pour IPLS alors qu'elle diminue pour ADMM.

### 5.5.3 Avec pénalisation : comparaison CPU/GPU

Lorsque la pénalisarion spatiale est prise en compte, seule l'implémentation *par image* et possible. C'est donc celle qui est utilisée dans cette section. De plus, plusieurs méthodes d'estimation des directions de descente ont été abordées dans le chapitre 4 : la méthode MM, la méthode PCG avec préconditionneur MM, PCG avec préconditionneur de Cholesky incomplet. La factorisation de Cholesky incomplète nécéssite l'utilisation du formalisme des matrices creuses. Ce formalisme est disponible à travers la bibliothèque *cuSPARSE* développée récemment par Nvidia [Nvidia, 2014]. Cette bibliothèque n'est pas utilisée dans cette thèse. Parmi les deux autres possibilités, seule la méthode MM a fait l'objet d'une implémentation GPU. Dans cette section sont donc comparées les versions CPU et GPU de la méthode IPLS avec pénalisation de type $\ell_2$ ou $\ell_2 - \ell_1$ avec calcul des directions de descente par la méthode MM. Les valeurs de tous les paramètres sont les mêmes que dans les sections 3.8 et 4.2. Le tableau 5.5 donne les temps de

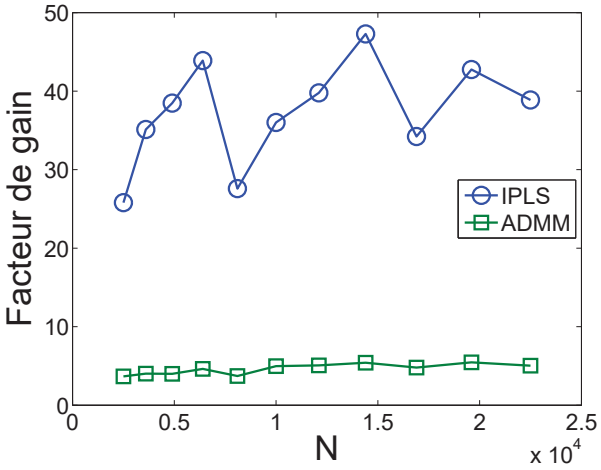



FIGURE 5.10 – Influence du nombre de pixels sur le gain de temps de calcul entre les implémentation CPU et GPU pour les méthodes IPLS et ADMM.

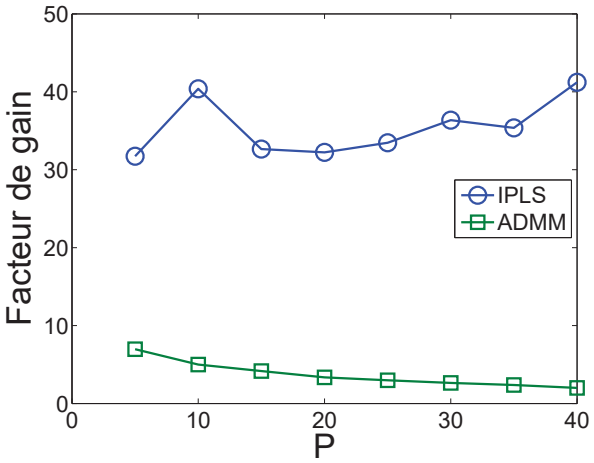



FIGURE 5.11 – Influence du nombre de endmembers sur le gain de temps de calcul entre les implémentation CPU et GPU pour les méthodes IPLS et ADMM.

| Méthode | IPLS | | | |
|---|---|---|---|---|
| Type de pénalisation | $\ell_2$ | | $\ell_2 - \ell_1$ | |
| Implémentation | CPU | GPU | CPU | GPU |
| Temps de calcul (s) | 44.3 | 1.57 | 48.8 | 1.80 |
| Facteur de gain | 28 | | 27 | |

TABLE 5.5 – Comparaison des implémentations CPU et GPU pour les méthodes IPLS et ADMM dans la situation de référence.

calcul et les gain constatés. Ceux-ci sont légèrement plus faibles que dans le cas non-pénalisé mais restent importants

## 5.6 Conclusion

Dans un premier temps, une étude indépendante a été menée sur l'utilisation du GPU pour l'inversion d'un grand nombre de systèmes linéaires de faible taille. Ce travail a pu être exploité pour l'implémentation d'IPLS grâce aux modifications apportées à la méthode au chapitre 4. Les facteurs d'accélérations apportés par le GPU varient entre 25 et 50 dans les conditions testées. Cette implémentation efficace permet d'envisager des applications à grande échelle. Le chapitre suivant expose deux applications d'IPLS sur des images réelles réalisées dans le cadre de collaborations.

# Chapitre 6

# Application à des images réelles

## Sommaire



Un cas d'étude portant sur les données de la mission Gaia de l'Agence Spatiale Européenne (ESA) a été abordé au chapitre 5. Dans ce chapitre, deux autres applications sont proposées. La première concerne les images hyperspectrales issues de la mission Mars Express de l'ESA. Ce travail a contribué à la publication [Schmidt *et al.*, 2014]. La seconde s'intéresse au suivi de la végétation côtière mené par l'OSUNA. Ces applications permettent de valider l'applicabilité de la méthode IPLS développée dans cette thèse.

## 6.1 Cartographie des formations géologiques sur Mars

### 6.1.1 Mars Express et OMEGA

Mars Express est une sonde spatiale de l'ESA lancée le 2 juin 2003 pour étudier la planète Mars. Il s'agit de la première mission d'exploration d'une autre planète du système solaire lancée par l'Agence européenne. Sa mission est de recueillir des données sur la surface, l'atmosphère, l'ionosphère et le sous-sol de la planète. Pour mener à bien sa mission, 7 instruments sont embarqués à bord de la sonde. Parmi eux, le spectro-imageur OMEGA (*Observatoire pour la Minéralogie, l'Eau, les Glaces et l'Activité*) permet l'acquisition d'images hyperspectrales de la surface de la planète [Bibring *et al.*, 2004]. Il étudie la composition minéralogique de la surface et la distribution de certains composés de l'atmosphère. 352 bandes spectrales sont acquises par cet instrument, avec une résolution de 7 nm pour les longueurs d'onde comprises entre 0.35 et 1 $\mu$m, 14 nm entre 1 et 2.5 $\mu$m, et 20 nm entre 2.5 et 5.1 $\mu$m. L'orbite de Mars Express

étant elliptique, la résolution spatiale dépend de l'altitude d'acquisition. Elle est comprise entre 300 m par pixel à une altitude de 350 km et 4.8 km par pixel à une altitude de 4000 km. Une image compte entre 16 et 128 pixels dans sa largeur, entre 300 et 20000 dans sa longueur. Ces variations sont une conséquence de l'orbite particulière de la sonde.

### 6.1.2 Démixage spectral supervié : application à grande échelle sans pénalisation spatiale

Cette application a été menée en collaboration avec Luca Capriotti, stagiaire à l'ESA, ainsi que ses encadrants Albrecht Schmidt de l'ESA et Frederic Schmidt de l'Université Paris-Sud. Elle a pour objet l'estimation des cartes d'abondances à grande échelle sur un total de 1290 images de tailles diverses issues de l'instrument OMEGA. Une chaîne de traitement est programmée pour effectuer automatiquement les actions suivantes sur chaque image :

- Extraction de l'image de la base de données PSA (*Planetary Science Archive*) de l'ESA.
- Conversion de la mesure en réflectance, en particulier grâce à une étape de correction atmosphérique.
- Application de la méthode IPLS sur GPU sans pénalisation et avec contrainte SLO.

L'originalité de ce travail réside dans la bibliothèque spectrale utilisée ainsi que dans la prise en compte des incertitudes sur les données et les abondances estimées.

#### Construction de la bibliothèque spectrale

Les pôles de mélanges utilisés sont regroupés dans une bibliothèque spectrale utilisée dans l'algorithme IPLS. Celle-ci comprend 32 spectres de réflectance correspondant à des minéraux, glaces et gaz atmosphériques dont on connaît ou suppose la présence à la surface de Mars. Ces spectres sont représentés dans la figure 6.1. Mais cette bibliothèque comprend également 12 spectres artificiels ne correspondant à aucun matériau recherché. Ces spectres sont des constantes, des fonctions affines ou des portions de sinusoïdes. Il sont représentés dans la figure 6.2. Ils ont pour fonction de réduire la dépendance des détections à des processus physiques présent dans les images hyperspectrales mais non corrigés. Ces processus sont jugées comme non-pertinent dans l'analyse (contribution des aérosols, effet de taille de grain). Ils apportent essentiellement des contributions de basses fréquences dans les spectre de réflectance, perturbant fortement l'estimation de leurs abondances. Or, les variations de haute fréquences comme les bandes d'absorption sont les informations les plus importantes pour l'identification des matériaux. En ajoutant les spectres de la figure 6.2 à la bibliothèque spectrale, les variations de basses fréquences se traduisent par des abondances non-nulles des spectres artificiels, permettant une meilleur estimation des abondances des matériaux réels [Schmidt *et al.*, 2011].

#### Prise en compte de l'incertitude sur les données

Le niveau d'incertitude sur le spectre observé en un pixel donné peut être estimé et modélisé par une densité de probabilité Gaussienne de moyenne nulle et dont la matrice de variances-covariances est $\boldsymbol{C} \in \mathbb{R}^{L \times L}$ où $L$ est le nombre de bandes specrales acquises. La matrice $\boldsymbol{C}^{-1}$ étant symétrique définie positive et de taille modeste, elle peut être facilement factorisée en utilisant la décomposition de Cholesky

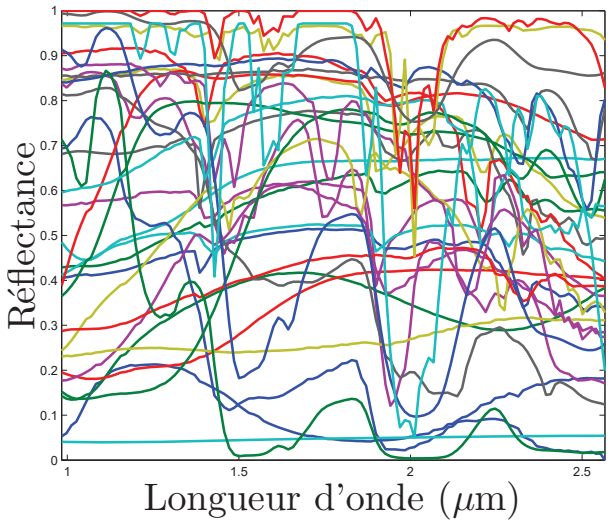



FIGURE 6.1 – 32 spectres de matériaux réels de la bibliothèque spectrale utilisée pour traiter les images de OMEGA.

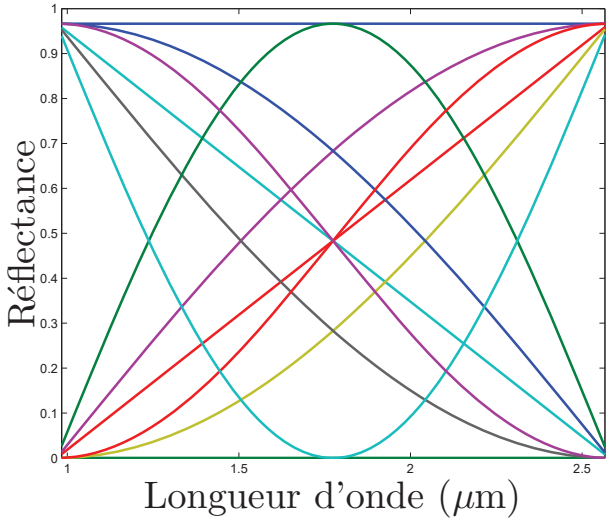



FIGURE 6.2 – 12 spectres artificiels de la bibliothèque spectrale utilisée pour traiter les images de OMEGA.

$$\boldsymbol{C}^{-1} = \boldsymbol{L}^{\mathrm{t}}\boldsymbol{L}, \tag{6.1}$$

avec $\boldsymbol{L} \in \mathbb{R}^{L\times L}$ triangulaire inférieure. Ainsi modélisée, l'incertitude sur les données peut être prise en compte en remplaçant le critère des moindre carrés pénalisé défini au chapitre 2 par un critère des moindres carrés pondérés pénalisés

$$\begin{aligned} F(\boldsymbol{A}) &= \sum_{n=1}^{N} (\boldsymbol{y}_n - \boldsymbol{S}\boldsymbol{a}_n)^{\mathrm{t}}\boldsymbol{C}^{-1}(\boldsymbol{y}_n - \boldsymbol{S}\boldsymbol{a}_n) + \beta R(\boldsymbol{A}) \\ &= (\boldsymbol{Y}' - \boldsymbol{S}'\boldsymbol{a}_n)_F^2 + \beta R(\boldsymbol{A}), \end{aligned} \tag{6.2}$$

avec $\boldsymbol{Y}' = \boldsymbol{Y}\boldsymbol{L}$ et $\boldsymbol{S}' = \boldsymbol{S}\boldsymbol{L}$. On voit alors qu'il suffit d'ajouter à la chaîne de traitement une étape de normalisation de la matrice d'observation $\boldsymbol{Y}$ et de la matrice des endmembers $\boldsymbol{S}$ en amont de l'estimation des abondances.

**Estimation de l'incertitude des abondances**

Au voisinage de la solution $\boldsymbol{a}^\star$, le critère F($\boldsymbol{a}$) où $\boldsymbol{a}$ est la version vectorisée de la matrice des abondances $\boldsymbol{A}$ peut être approximé par [Kalmikov et Heimbach, 2014] :

$$J(\boldsymbol{a}^\star) = F(\boldsymbol{a}^\star) + \frac{1}{2}(\boldsymbol{a} - \boldsymbol{a}^\star)^{\mathrm{t}}\nabla^2 F(\boldsymbol{a}^\star)(\boldsymbol{a} - \boldsymbol{a}^\star). \tag{6.3}$$

L'inverse du Hessien du critère est donc un indicateur de l'incertitude sur la solution du problème. Il représente la matrice de variances-covariances associée à la solution. En pratique, seule la diagonale de $\nabla^2 F(\boldsymbol{a}^\star)^{-1}$ est calculée. Elle représente la variance associée à chaque abondance estimée.

**Résultats**

Pour cette application uniquement, les tests sont effectués sur une station de travail comprenant un processeur Dual Core cadencé à 2.53 GHz, 4 GO de mémoire RAM, ainsi qu'une carte graphique Tesla C2050 (448 unité de calculs, 1.15 GHz, 3 GO de mémoire globale). Le tableau 6.1 donne les temps de calcul constatés avec l'implémentation Matlab présentée au chapitre 4 et l'implémentation CUDA présentée au chapitre 5. Le facteur de gain en temps de calcul est de 6.7. L'objectif de cette application n'est pas un traitement en temps réel des données de OMEGA, mais il est intéressant de noter que d'après [Bibring *et al.*, 2004], le temps d'acquisition d'une image varie entre 12 et 24 min. Avec le matériel utilisé, l'implémentation GPU permet donc un traitement des données en temps réel alors que l'implémentation CPU ne le permet pas.

Pour chaque image, 44 cartes d'abondances sont obtenues parmi lesquelles 32 correspondent à des matériaux réels. La figure 6.3 réunit les 1290 cartes d'abondances pour 4 de ces matériaux. Les cartes sont représentées sur une projection Mercator de la surface complète de la planète. On peut voir des concentrations de certains matériaux par région ou par lattitude. Ces informations permettent aux planétologues d'étudier la géologie de cette planète. Les images choisies pour cette expérience couvrent une partie importante de la surface de Mars. OMEGA ayant couvert 100% de la surface depuis sa mise en service, cette expérience montre qu'il serait possible d'utiliser la méthode IPLS pour estimer les cartes d'abondances sur planète entière.

| Implémentation de l'algorithme IPLS | CPU (Matlab) | GPU (CUDA) |
|---|---|---|
| Temps de traitement moyen par image | 20 min | 3 min |
| Temps de traitement total pour 1290 images | environ 20 jours | environ 3 jours |

TABLE 6.1 – Temps de calcul de l'algorithme IPLS pour 1290 images issues de Mars Express.

Il faut tout de même remarquer que la jonctions de certaines cartes portant sur le même matériau sur des régions voisines ou superposées n'est pas parfaite. Plusieurs facteurs sont en cause. Les variations naturelles de la composition de l'atmosphère peuvent expliquer une partie de ces incohérences car la base de données de spectres de référence ne prend pas en compte la présence de nuages, mais cela révèle surtout les limites de la modélisation effectuée, en particulier l'absence de prise en compte des angles d'incidence et de réflexion de la lumière.

### 6.1.3 Estimation des cartes d'abondances sur une image avec pénalisation spatiale

**Choix de l'image**

L'introduction de la pénalisation spatiale est encore trop coûteuse en temps de calcul pour être utilisée de façon systématique. Le bruit de mesure en général peu important ne justifie pas ce surcoût. Cependant, certaines images présentent un niveau de bruit particulièrement élevé dû à une faible illumination. C'est le cas de l'image ORB0068-5 (5ème image de la 68ème orbite de la sonde) disponible dans la base de données PSA (`www.rssd.esa.int/PSA`). L'image d'origine contient $128 \times 887$ pixels et 256 bandes spectrales. Cette application est réalisée sur un sous ensemble de $128 \times 500$ pixels et 114 bandes spectrales entre $0.984\mu m$ et $2.566\mu m$. Ce sous ensemble est représenté dans la figure 6.4 qui contient les mesures de réflectance effectuées dans trois bandes spectrales. La partie suppérieure de l'image étant peu lumineuse, le rapport signal à bruit y est faible.

**Résultats**

La bibliothèque spectrale utilisée est la même que dans l'exemple précédent. L'estimation des abondances est faite avec la contrainte SLO et une pénalisation de type $\ell_2 - \ell_1$ avec un paramètre de pénalisation $\beta = 10$ et un paramètre de la fonction de pondération $\delta = 0.1$ choisis de façon a obtenir des résultats visuellement satisfaisants. Le tableau 6.2 donne les temps de calcul relevés avec les implémentations CPU et GPU sans pénalisation, et GPU uniquement avec pénalisation, la version CPU avec pénalisation demandant un temps de calcul trop long pour être effectué. On voit que l'introduction de la pénalisation spatiale sur une image de grande taille augmente sensiblement le temps de calcul, au point d'être inenvisageable sans l'aide du GPU. Les cartes d'abondances des matériaux réels les plus présents résultants de ce traitement sont affichées dans la figure 6.5. On constate que le bruit de mesure de la figure 6.4 est amplifié sur certaines cartes d'abondances dû au mauvais conditionnement du problème lorsque la pénalisation n'est pas utilisée. Il est en revanche très atténué dans les cartes estimées sans la pénalisation spatiale. La prise en compte de l'*a priori* de régularité apporte dans le cas de cette image hyperspectrale un gain dans la qualité du résultat, révélant certains détails qui sont sans cela noyés dans le bruit.

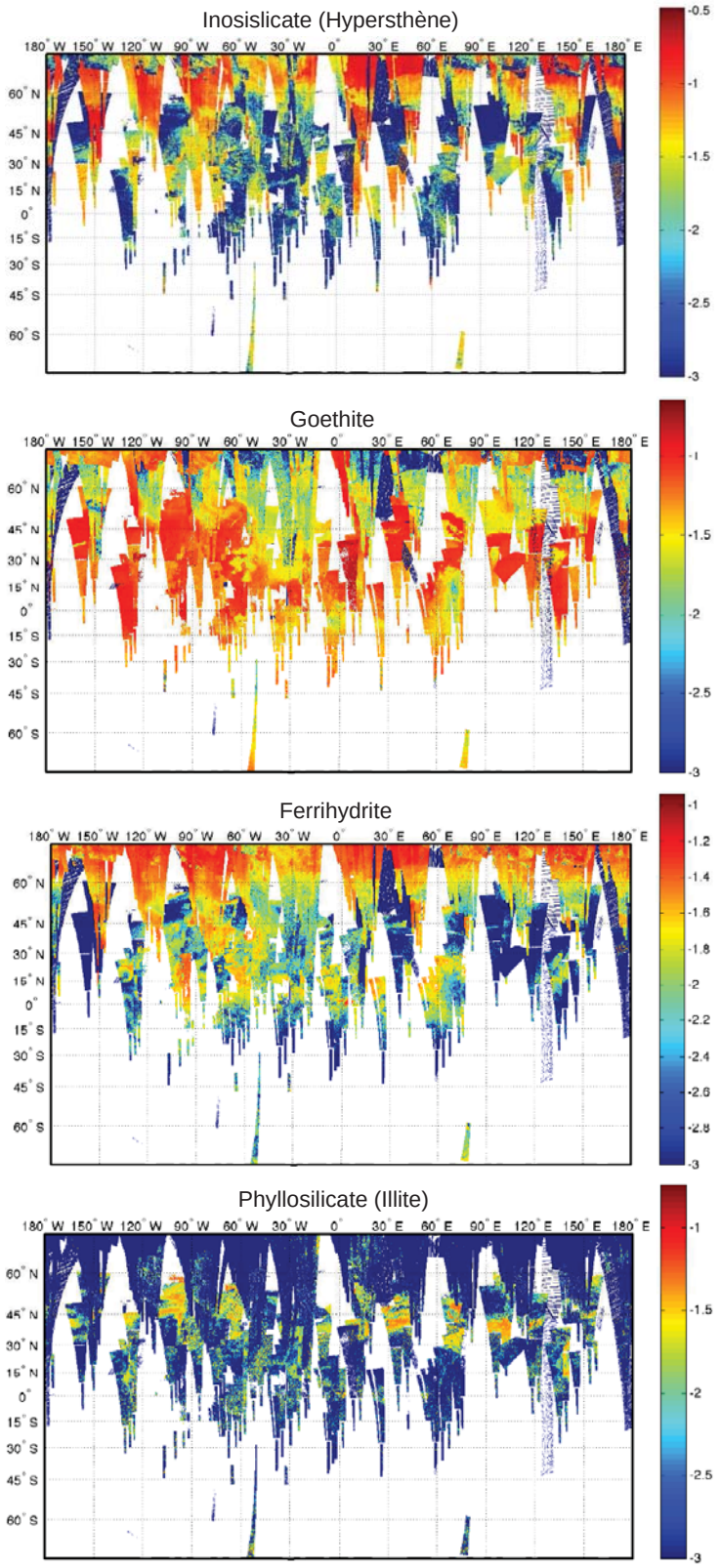



FIGURE 6.3 – Projection Mercator des 1290 cartes d'abondances pour 4 des 32 matériaux réels de la bibliothèque spectrale (échelle logarithmique).

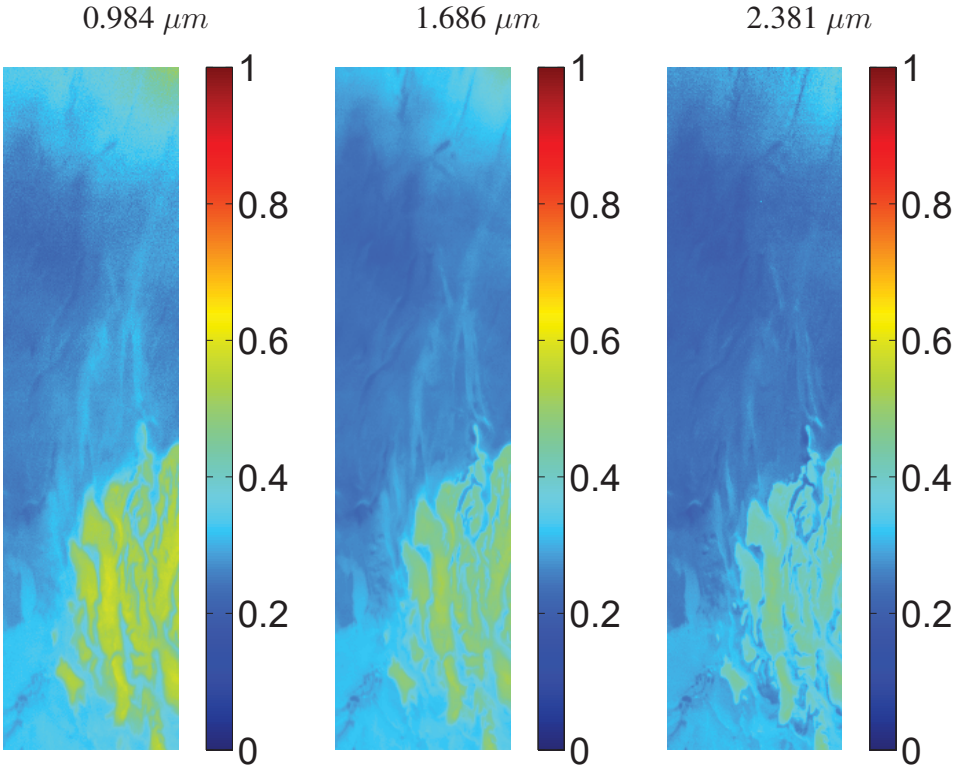



FIGURE 6.4 – Représentation partielle de l'image hyperspectrale ORB0068-5 : réflectances mesurées pour différentes longueurs d'onde.

| | |
|---|---|
| Temps de calcul sur CPU sans pénalisation | 136 s |
| Temps de calcul sur GPU sans pénalisation | 7.8 s |
| Temps de calcul sur GPU avec pénalisation | 38479 s |

TABLE 6.2 – Temps de calcul de l'algorithme IPLS pour traiter l'image ORB0068-5.

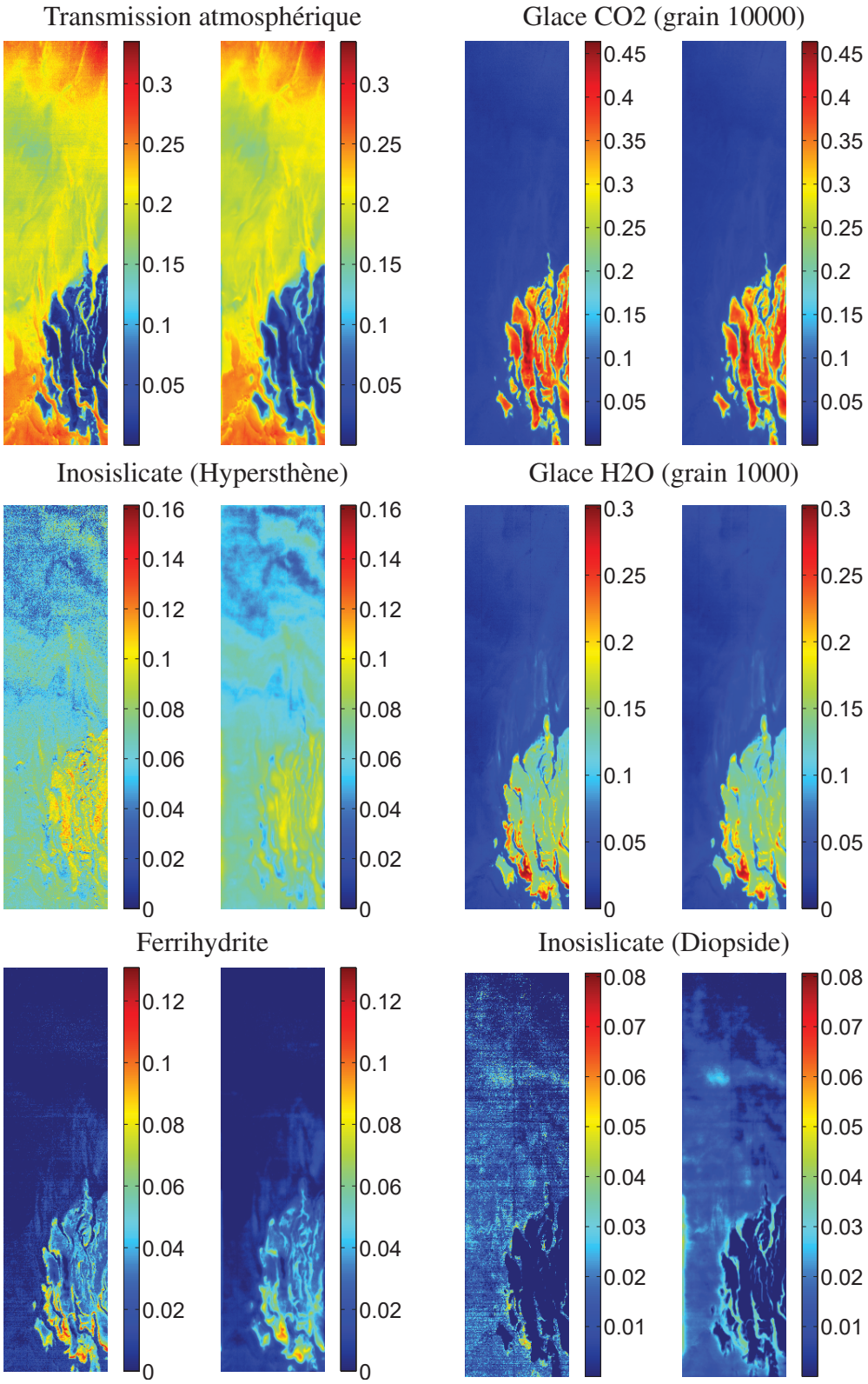



FIGURE 6.5 – Abondances estimées sans pénalisations (cartes de gauche) et avec pénalisation (cartes de droite) pour l'image ORB0068-5. Affichage pour les 6 matériaux réels dont les abondances sont les plus importantes.

## 6.2 Suivi de la végétation sur les dunes côtières de Vendée

**Présentation du projet**

Cette étude a été menée en collaboration avec Antoine Ba, doctorant au LPGN (*Laboratoire de Planétologie et Géodynamique*) de l'Université de Nantes, et Patrick Launeau, professeur à l'Université de Nantes, dans le cadre du programme RS2E (*Réseau de Suivi et de Surveillance Environnemental*) piloté par l'OSUNA (*Observatoire des Sciences de l'Univers Nantes Atlantique*).

Les dunes côtières sont un milieu protégé, certaines espèces de plantes qui les colonisent font parties de la flore protégée. Mais la surfréquentation des plages avec le tourisme grandissant et le non respect des dunes avec un piétinement fréquent entraîne la raréfaction des végétaux en haut de plage, cela cause une instabilité des dunes, provoquant une conquête des dunes à l'intérieur des terres ou bien leur érosion ce qui entraîne la disparition du sable, on parle alors d'amaigrissement des plages. Cette disparition du sable favorise l'érosion des plages et donc du trait de côte, grignotant le domaine continental. Le maintien des dunes par la végétation est primordial pour garder l'équilibre constant entre l'océan et le continent. La dynamique d'érosion est un phénomène naturel. La préservation des dunes doit se faire seulement de manière naturelle. La cartographie des zones dunaires se fait généralement par relevés des espèces présentent sur le terrain de manière manuelle. Les plantes étant des organismes vivants fixés, il est également possible d'utiliser la télédétection. L'imagerie hyperspectrale apporte dans ce domaine une simplification de la cartographie des espèces spécifiques à ce milieu de vie avec un gain de temps conséquent en couvrant une grande surface. Une campagne aéroportée a été effectuée en septembre 2013 avec la collaboration du LPGN, le LETG et FIT-Conseil sur toute la face Ouest Atlantique du département de la Vendée. L'équipe travaillant sur ce projet a pour mission d'exploiter conjointement des données issues d'un spectro-imageur et d'un LiDAR (*Light Detection And Ranging*) permettant d'obtenir des informations sur la topographie de la surface observée afin de faciliter le suivi l'évolution de la végétation dunaire.

Plusieurs spectro-imageurs sont conjointement utilisés. Ils capturent des images hyperspectrales dans des zones différentes du spectre lumineux. Celui qui est utilisé pour illustrer cette application est caractérisé par une résolution spectrale de 3.6 nm et couvre des longueurs d'onde allant de $0.410\mu m$ à $0.987\mu m$, soit 160 bandes spectrales, et une résolution spatiale variant entre 0.5 m et 1 m.

Deux approches sont envisagées pour le traitement des images hypespectrales de ce projet. La première consiste à former une bibliothèque spectrale à partir des spectres de réflectance des plantes d'intérêt mesurés directement sur le terrain, puis d'utiliser cette bibliothèque pour l'estimation de cartes d'abondances pour chaque type de plante. La deuxième approche consiste à estimer les pôles de mélange directement à partir de l'image hyperspectrale grâce à une méthode d'extraction des endmembers, de les utiliser pour l'estimation de cartes d'abondances, et de faire appel à un expert pour associer un type de plante à chaque couple endmember/carte.

Le choix retenu pour l'estimation des cartes d'abondances est la méthode IPLS, plus particulièrement la version implémentatée avec Matlab avec contraintes STO et sans pénalisation spatiale telle que présentée dans la section 4.2. Pour faciliter l'utilisation de cette méthode, une interface graphique a été réalisée dans Matlab. Elle permet de charger une image hyperspectrale ainsi qu'une bibliothèque spectrale ou bien de former une bibliothèque spectrale par extraction des endmembers de l'image par la méthode VCA ou NFINDR. Le format de fichier du logiciel ENVI utilisé pour la visualisation d'images hyperspectrales est pris en charge. L'interface permet l'utilisation de la méthode IPLS dans toutes les versions décrites dans le chapitre 4. Une

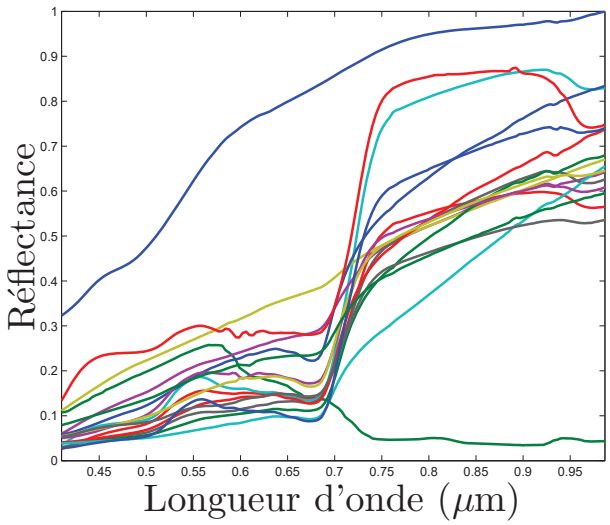



FIGURE 6.6 – Bibliothèque spectrale utilisée pour traiter les images de la côte vendéenne : 15 spectres de plantes + 1 spectre de sable + 1 spectre d'eau.

breve documentation a été rédigée pour faciliter la prise en main de l'interface. Elle est donnée dans l'annexe C.

**Exemple de résultat**

La figure 6.7 présente un exemple de résultat obtenu en utilisant la première approche (utilisation d'une bibliothèque spectrale contenant les 17 spectres de la figure 6.6 sur une image de taille $472 \times 387$ pixels. Les cartes obtenues montrent les zones dans lesquelles es différents types de végétation sont présents. Ces résultats ont été obtenus en 84 s sur CPU. Un essai sur GPU a été réalisé bien qu'il ne soit pas utilisé dans ce projet aboutissant à un temps de calcul de 3.95 s, soit un facteur de gain de 21.

Dans cette application, une difficulté majeure provient de la forte corrélation des différents spectres de la bibliothèque. Cela résulte par endroit en de fausses estimations. Les utilisateurs ayant une bonne connaissance du terrain analysé, il leur est possible de corriger les résultats *a posteriori*. L'image est découpés en différentes zones. Dans chaque zone, les abondances des plantes dont la présence est improbable sont affectées au plantes dont la présence est probable et dont le spectre est le plus proche. Cette procédure *ad hoc* permet d'obtenir des résultats conformes avec les relevés effectués sur le terrain.

Du point de vue de la préservation des dunes, les cartes les plus importantes sont celles correspondant à des mousses et lichens (hypnum, cladonia, pleurochaete et tortula). Ces plantes sont celles qui permettent de fixer le sable. Leur développement participe à la robustesse de la dune et donc au maintien du trait de côte.

## 6.3 Conclusion

Ce chapitre démontre l'applicabilité de la méthode IPLS développée dans cette thèse sur des projets réels. La possibilité de prendre en compte différents types de contraintes et un *a priori* de régularité spatiale, sa rapidité d'exécution, et son implémentation GPU en font une méthode avantageuse pour les utilisateurs d'images hyperspectrales dans différents domaines. L'analyse

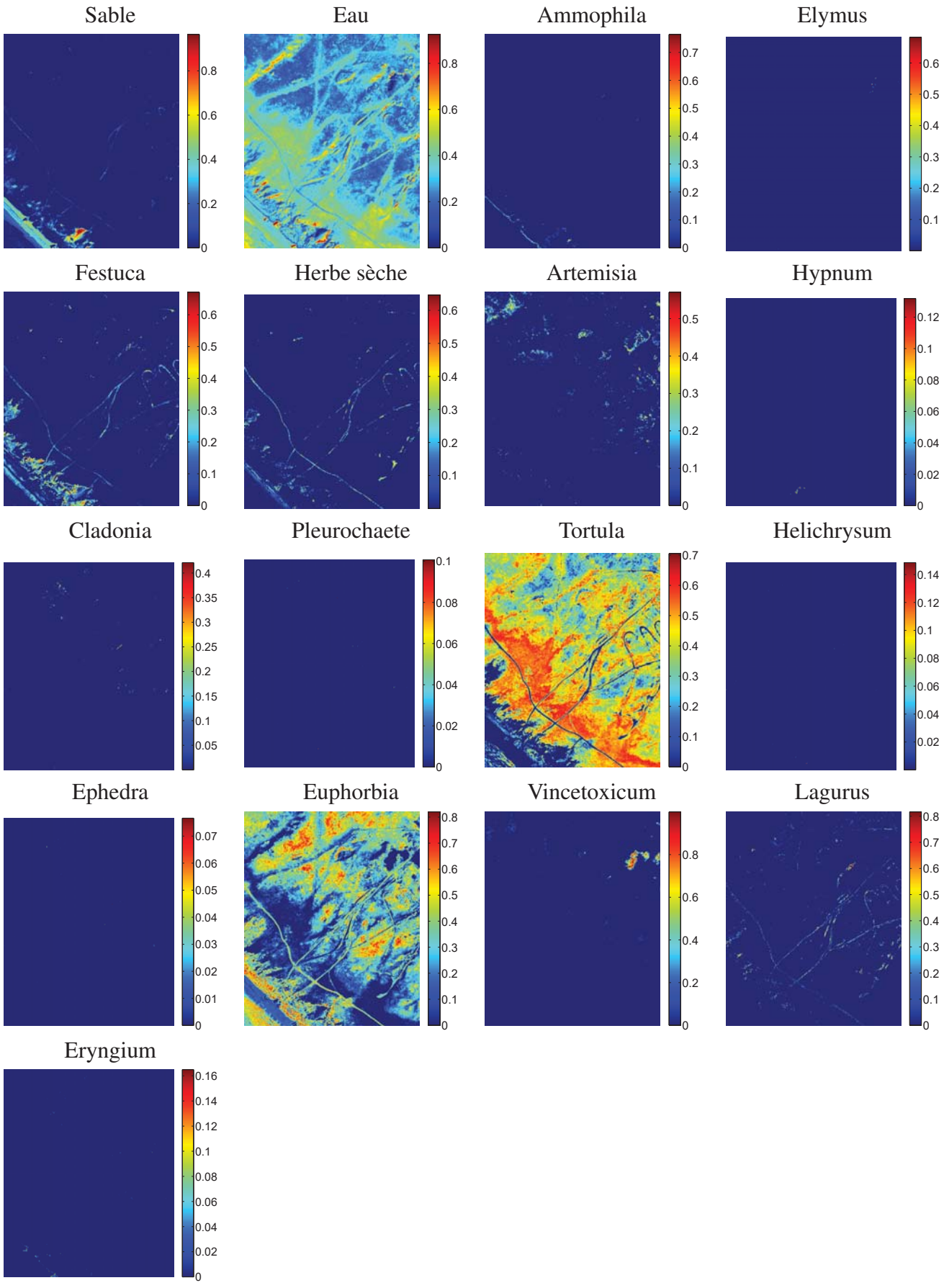



FIGURE 6.7 – Abondances estimées des 17 matériaux de la bibliothèque spectrale sur la côte de Noirmoutier.

s'est focalisée sur le temps de calcul et le gain apporté par l'utilisation des stratégies d'accélération proposées dans cette thèse.

La qualité des résultats dépend en grande partie du choix de la bibliothèque spectrale. Dans l'application concernant la géologie de la planète Mars, l'ajout dans la bibliothèque de plusieurs spectres artificiels permet une meilleurs estimation. Dans l'application concernant le suivi de la végétation côtière, une étape de post-traitement manuelle est effectuée afin de corriger les fausses détections dues à une bibliothèque contenant des spectres fortement corrélés.

# Chapitre 7

# Conclusion générale

## 7.1 Bilan

Cette thèse a exploré la question de l'estimation des cartes d'abondances qui se pose dans le domaine de l'imagerie hyperspectrale. Le modèle de mélange linéaire a été utilisé. Les spectres de réflectances mesurés en chaque pixel d'une image hyperspectrale sont considérés comme étant des combinaisons linéaires des spectres des composants de surface. L'inversion de ce modèle requière la prise en compte de contraintes liées à la physique du problème ainsi que d'informations *a priori*. Les contraintes de non-négativité des abondances et de somme égale à un ou de somme inférieure à un sont exploitées. Un *a priori* de régularité spatial est utilisé. Ce processus conduit à poser le problème d'estimation des cartes comme un problème d'optimisation convexe sous contraintes linéaires d'égalité et d'inégalité.

Plusieurs méthodes existantes sont détaillées et comparées : deux méthodes de contraintes actives (NNLS et FCLS), une méthode de Lagrangien augmenté (ADMM), et une méthode de points intérieurs (IPLS). L'analyse comparative de ces méthodes a été menée de façon à mettre en lumière leurs comportements face aux changements de chaque paramètre du modèle.

Des propositions ont été faites pour accélérer la méthode IPLS. Ces propositions concernent à la fois l'implémentation et la partie algorithmique. Côté implémentation, les choix effectués sous Matlab ont été discutés et améliorés. Concernant la partie algorithmique, l'étape de IPLS la plus coûteuse en temps de calcul porte sur la résolution d'un système linéaire de grande taille non-séparable. Elle est accélérée en l'effectuant de façon inexacte à l'aide d'une méthode itérative. Cette méthode permet non seulement une accélération de IPLS, mais assure également la séparabilité de ses variables, autorisant une implémentation parallèle efficace. La séparabilité et la convergence globale sont démontrés grâce à l'utilisation d'une approximation majorante séparable utilisée dans un algorithme de type Majoration-Minimisation ou de Gradient Conjugué préconditionné.

Une implémentation GPU a été réalisée en exploitant au maximum l'architecture matérielle présente. L'utilisation adéquat de cette ressource conduit à des gains de temps considérable, ce qui a été validé par l'application menée à grande échelle avec l'ESA concernant l'estimation de cartes d'abondances pour une grande partie de la surface de Mars. Une étude plus locale a également profité de la méthode proposée dans cette thèse pour réaliser la cartographie d'espèces végétales sur la côte vendéenne afin de préserver l'espace naturel côtier. A cette occasion une interface graphique mettant en œuvre la méthode IPLS a été créée. La méthode IPLS développée dans cette thèse a donc fait ses preuves aussi bien au niveau théorique que pratique et fait

désormais partie des outils disponibles pour les utilisateurs d'images hyperspectrales.

Ce qu'il faut retenir du travail présenté dans cette thèse est l'intérêt d'une approche consistant à agir en amont de l'implémentation matérielle en modifiant la méthode de traitement de manière à ce que l'algorithme résultant présente une structure fortement parallélisable tout en préservant des propriétés théoriques importantes.

## 7.2 Perspectives

Commençons par citer quelques évolutions techniques qui pourraient améliorer rapidement l'expérience des utilisateurs de la méthode IPLS sans demander d'étude théorique supplémentaire :

- **Prise en compte d'un *a priori* de parcimonie** : Dans de nombreuses applications, la plupart des pixels ne comptent pas plus de deux ou trois éléments. Lorsqu'une bibliothèque spectrale de taille importante est utilisée, on peut s'attendre à un grand nombre d'abondances proches de 0. Il est possible d'inclure cette information comme un *a priori* de parcimonie dans la méthode IPLS. Pour cela, on peut assortir le critère correspondant à un pixel d'un terme de pénalisation : $F(\boldsymbol{a}) = \|\boldsymbol{y} - \boldsymbol{S}\boldsymbol{a}\|_2^2 + \tau \sum_i \phi(\boldsymbol{a}_i)$ avec $\phi$ convexe. Cette pénalisation permet de favoriser les valeurs d'abondances faibles. En appliquant cette définition au abondances d'une image entière, la structure du système primal reste inchangée car la pénalisation est séparable par pixel. Elle sera donc bloc-diagonal si aucune pénalisation spatiale n'est prise en comptes, et on pourra utiliser l'approche MM ou PCG dans le cas de pénalisation spatiale. On peut ainsi combiner une pénalisation intra-pixel favorisant la parcimonie et une pénalisation inter-pixel favorisant la régularité.

- **Utilisation de la bibliothèque cuSPARSE** : L'implémentation GPU a été réalisée avec CUDA sans utiliser de bibliothèque. Or, il existe la bibliothèque cuSPARSE [Nvidia, 2014] permettant la gestion de matrices et vecteurs creux. Cela permettrait de porter sur GPU la version PCG+ICHOL qui est la plus performante sur CPU. De plus, la bibliothèque cuSPARSE contient entre autre une fonction de factorisation de Cholesky incomplète et une fonction implémentant l'algorithme PCG. Il serait intéressant de voir l'accélération apportée par cette bibliothèque alors que le calcul avec des matrices et vecteurs creux est moins adapté au GPU qu'avec des matrices et vecteurs pleins.

- **Prise en compte d'une distance inter-pixel variable** : Les images de Mars Express ont la particularité d'avoir une distance inter-pixel variable. En effet, l'orbite elliptique de la sonde impose un changement continuel d'altitude et donc de surface couverte par un pixel. Ainsi, la distance réelle au sol entre deux pixels voisin n'est pas la même sur l'ensemble de l'image. Connaître la distance entre chaque couple de pixels voisins permettrait d'ajuster le paramètre de régularisation spatiale $\beta$ en lui donnant des valeurs plus élevées là où la distance inter-pixel est faible.

Ce travail a également soulevé des questions plus théoriques ouvrant la voie à de nouvelles pistes de recherche :

- **Construction d'une meilleure approximation majorante** : dans le chapitre 4, l'approximation majorante quadratique séparable proposée pour le calcul des directions primales dans le cas pénalisé donne des résultats satisfaisants. Néanmoins il existe probablement d'autres propositions conduisant à un calcul plus rapide des directions primales.

La méthode IPLS pourrait être améliorée par une étude plus approfondie des techniques de majoration [Marshall *et al.*, 2010].

- **Calcul des directions primales par un autre algorithme** : les algorithmes MM et PCG ont été utilisés dans ce travail. Il existe bien d'autres algorithmes permettant de résoudre un système linéaire [Rugh, 1996], par exemple l-BFGS [Liu et Nocedal, 1989] ou encore les méthodes à sous-espace de gradiant [Shi et Shen, 2006].
- **Choix des hyperparamètres** : Comme exposé dans la section 2.4.5, la méthode IPLS compte un grand nombre d'hyperparamètres dont les valeurs sont fixées arbitrairement. Si ces valeurs semblent convenir à un grand nombre de situations, il serait préférable de pouvoir les calculer de façon automatique. L'estimation d'hyperparamètre est un domaine de recherche actif qui pourrait bénéficier à un algorithme tel qu'IPLS [Molina *et al.*, 1999; Bergstra et Bengio, 2012].
- **Modèle de mélange non-linéaire** : le choix du modèle de mélange linéaire est assez classique en imagerie hyperspectral, mais il n'est pas adapté à toutes les situations. Comme évoqué dans le chapitre 2, certains mélanges de matériaux ont par nature un spectre de réflectance non-linéaire. De plus en plus de travaux s'intéressent à ce sujet [Nascimento et Bioucas-Dias, 2009; Dobigeon *et al.*, 2014]. Les méthodes de points intérieur ne sont pas spécifiques au problèmes linéaires [Byrd *et al.*, 1999]. Dans ce contexte, il serait intéressant d'étudier l'applicabilité des méthodes primales-duales points intérieurs à l'estimation des abondances sous hypothèse de mélange non-linéaire.

# Annexe A

# Intérêt de la contrainte « somme inférieure ou égale à 1 »

La contrainte NN+SLO (« non-négativité » et « somme inférieure ou égale à 1 ») ne pose pas de difficulté technique et pourrait être prise en charge par toutes les méthodes étudiées dans cette thèse. Pourtant ce choix n'est jamais effectué dans la littérature. Deux raisons nous poussent à inclure cette possibilité dans l'algorithme IPLS.

D'une part, le niveau de réflectance peut varier d'un pixel à l'autre. Typiquement une surface à l'ombre enverra une radiance plus faible que la même surface au soleil. Si ce facteur n'est pas pris en compte, certaines valeurs de réflectance risquent d'être plus faibles que ce qu'elles devraient être. Dans ce cas il n'est pas possible d'estimer des abondances correctes avec la contrainte NN+STO. La contrainte SLO permet de relâcher la contrainte STO tout en gardant une borne maximale contrairement à la contrainte NN seule. Afin de tester cette idée, des images hyperspectrales sont générées d'après la procédure décrite dans la section 3.1. Les valeurs de références de la section 3.2 sont utilisées : 10 spectres de la bibliothèque USGS sont sélectionnés et mélangés dans une image de $100 \times 100$ pixels. Les valeurs de l'image hyperspectrale ainsi générée sont réduites pour simuler un manque de luminosité. Après application de la méthode IPLS avec l'un des trois types de contraintes testé, l'Erreur Quadratique Moyenne Normalisée (EQMN) est calculée. La figure A.1 montre que la contrainte NN+SLO est le meilleur choix dès lors que la réflectance mesurée d'un pixel est inférieure à 98 % de sa vrai valeur, alors que pour une luminosité de plus de 98 %, les abondances estimées avec les contraintes NN+SLO et NN+STO sont très proches. Ainsi dans une image réelle, les pixels les moins lumineux seront mieux estimés avec les contraintes NN+SLO.

D'autre part, il peut arriver qu'une image réelle comprenne un ou plusieurs éléments dont le spectre de réflectance ne se trouve pas dans la bibliothèque spectrale utilisée. Pour simuler ce cas, des images hyperspectrales sont générées toujours d'après la situation de référence, en particulier avec $P = 10$ endmembers. Ces images sont ensuite démixées avec IPLS en utilisant une bibliothèque spectrale incomplète contenant $P' \leqslant P$ endmembers. La qualité de reconstruction des spectres présents dans la bibliothèque utilisée pour les différents types de contraintes est illustrés par la figure A.2. Dans ce cas la différence entre les contraintes NN+SLO et NN+STO est faible. Néanmoins, dès que la bibliothèque compte un seul élément de moins que l'image, les contraintes NN+SLO sont préférables aux contraintes NN+STO.

Ces deux situations sont fréquentes lorsque l'on travaille avec des images réelles dont on ne connait pas tout par avance.

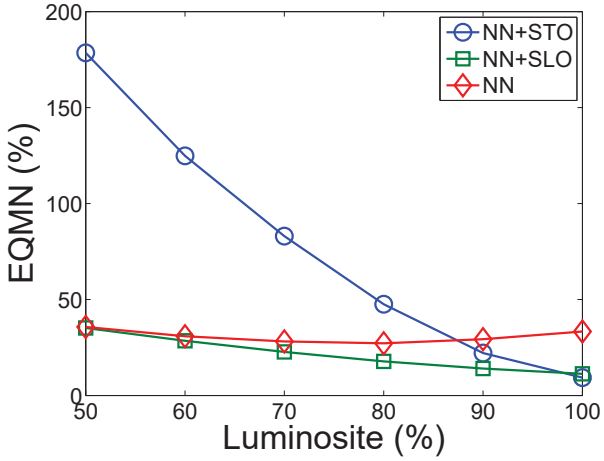



FIGURE A.1 – Erreur de reconstruction de la méthode IPLS pour différents types de contraintes lorsqu'une faible luminosité diminue les valeurs de réflectances mesurées.

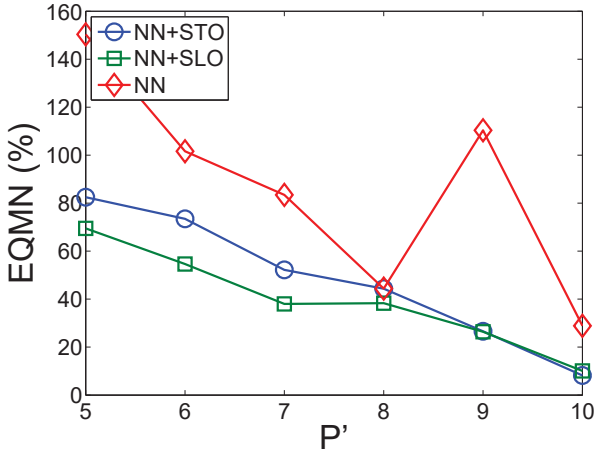



FIGURE A.2 – Erreur de reconstruction de la méthode IPLS pour différents types de contraintes lorsqu'une bibliothèque spectrale incomplète est utilisée.

# Annexe B

# Solving Systems of Linear Equations by GPU-based Matrix Factorization in a Science Ground Segment

Astronomy and Computing 3–4 (2013) 58–64

Contents lists available at ScienceDirect

Astronomy and Computing

journal homepage: www.elsevier.com/locate/ascom

Full length article

# Solving systems of linear equations by GPU-based matrix factorization in a Science Ground Segment

Maxime Legendre [a,*], Albrecht Schmidt [b], Saïd Moussaoui [a], Uwe Lammers [b]

[a] Institut de Recherche en Communications et Cybernétique de Nantes, 1 rue de la Noë, BP 92101, F-44321 Nantes Cedex 3, France
[b] European Space Astronomy Centre, PO Box 78, E-28691 Villanueva de la Cañada (Madrid), Spain

ARTICLE INFO

*Article history:*
Received 14 June 2013
Accepted 25 November 2013

*Keywords:*
Gaia mission
Linear solving
Graphics Processing Units (GPU)
Matrix factorization

ABSTRACT

Recently, Graphics Cards have been used to offload scientific computations from traditional CPUs for greater efficiency. This paper investigates the adaptation of a real-world linear system solver, which plays a central role in the data processing of the Science Ground Segment of ESA's astrometric Gaia mission. The paper quantifies the resource trade-offs between traditional CPU implementations and modern CUDA based GPU implementations. It also analyses the impact on the pipeline architecture and system development. The investigation starts from both a selected baseline algorithm with a reference implementation and a traditional linear system solver and then explores various modifications to control flow and data layout to achieve higher resource efficiency. It turns out that with the current state of the art, the modifications impact non-technical system attributes. For example, the control flow of the original modified Cholesky transform is modified so that locality of the code and verifiability deteriorate. The maintainability of the system is affected as well. On the system level, users will have to deal with more complex configuration control and testing procedures.



## 1. Introduction

### 1.1. Gaia mission

ESA's Gaia space mission (Lindegren et al., 2008) is designed to create an accurate and complete astrometric catalogue which will serve for decades to come: more than 1000 million objects (mostly stars in our Milky Way) down to apparent magnitude 20 will be repeatedly observed from the second Lagrange point (L2) during the satellite's nominal lifetime of 5 years. This will yield a raw data volume of about 100 TB from which each star's position, trigonometric parallax, and proper motion (for a smaller subset also radial velocity) will be determined to micro-arcsec accuracy (typically 25 $\mu$ as for parallax at 15th magnitude). The astrometric part of the catalogue will be complemented by astrophysical quantities derived from photometric and spectroscopic measurements that likewise take place onboard. The data processing and catalogue creation is carried out by the Gaia Data Processing and Analysis Consortium (DPAC) formed by around 400 individuals and six main data processing centres distributed across Europe. The used algorithms are diverse in nature but several core ones rely heavily on linear algebra. This paper investigates the trade-offs of moving from a traditional CPU-centric computing architecture to a more modern GPU architecture, for a particular algorithm of the Astrometric Global Iterative Solution.

### 1.2. Astrometric data processing: AGIS

The *Astrometric Global Iterative Solution* (AGIS) (Lindegren et al., 2012) is a central system in the Gaia Science Ground Segment and was therefore chosen as an interesting study case for this paper. It estimates the 5 astrometric parameters $\alpha$, $\delta$ (position), $\varpi$ (parallax), and $\mu_\alpha$, $\mu_\delta$ (proper motion) of each star through a maximum likelihood approach in which the optimal agreement between all astrometric measurements ($10^{12}$) and the unknown parameters of three needed models—namely, for the stars (S), the satellite's time-varying attitude (A), and the instrument's calibration (C)—are sought in a weighted least-square sense. A rigorous treatment of the problems would require the full diagonalization of a matrix of the order of $5 \cdot 10^9 \times 5 \cdot 10^9$ which is clearly intractable. Instead a number of approximations are made (see Lindegren et al., 2012 for details) which breaks the problem down into manageable pieces and for the astrometric parameters means that many $5 \times 5$ matrices have to be dealt with.

### 1.3. Problem statement

The main problem is to solve efficiently a high number $N$ of independent symmetric linear systems using a GPU based

* Corresponding author. Tel.: +33 677438252.
*E-mail address:* maxime.legendre@irccyn.ec-nantes.fr (M. Legendre).

2213-1337/$ – see front matter © 2013 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.ascom.2013.11.004

implementation. Let us note a linear system as

$$\boldsymbol{Ax} = \boldsymbol{b} \tag{1}$$

where $\boldsymbol{A} \in \Re^{n\times n}$ is a symmetric matrix, $\boldsymbol{b} \in \Re^n$ is a vector, and $\boldsymbol{x} \in \Re^n$ is the unknown variable. The size of the systems is noted $n$ and is set to $n = 5$ in this study.

In the literature, some methods have been developed to solve large scale systems using GPUs (Volkov and Demmel, 2008; Tomov et al., 2010), performing the parallelization over $n$. In this paper, we present an optimized solver for a high number of small size systems. As all the systems are independent, the parallelization will be performed over $N$.

The outline of the paper is as follows. In Section 2, two algorithms are considered to solve one given linear system. Section 3 discusses the implementation of these algorithms to solve a high number of systems on both CPU and GPU, focusing on the optimization of the GPU-based implementation. Section 4 presents the results of the optimizations proposed in terms of computation time. The issues raised by the integration of the modifications into a real-world system are discussed in Section 5; conclusions are presented in Section 6.

## 2. Linear solver algorithms

We do not consider in this paper iterative resolution algorithms such as Krylov subspace methods (Benzi, 2002). Instead, the resolution of a given linear system is performed in two steps: the LU decomposition of the matrix, followed by the solution of an upper and a lower triangular systems using forward and back substitution, respectively (see Golub and Van Loan, 1996).

Regarding the first step, a square matrix $\boldsymbol{A} \in \mathbb{R}^{n\times n}$ can always be decomposed into a product of a lower triangular matrix $\boldsymbol{L}$ and an upper triangular matrix $\boldsymbol{U}$. When $\boldsymbol{A}$ is symmetric and non-singular, the LU decomposition can be written $\boldsymbol{A} = \boldsymbol{LDL}^{\mathrm{t}}$, where $\boldsymbol{L}$ is a unit lower triangular matrix (consisting of only ones on its diagonal) and $\boldsymbol{D}$ is a diagonal matrix. When $\boldsymbol{A}$ is a symmetric positive-definite matrix, the LU decomposition becomes the Cholesky factorization $\boldsymbol{A} = \boldsymbol{LL}^{\mathrm{t}}$ where $\boldsymbol{L}$ is a (non-unit) lower triangular matrix.

In this study, the matrices to be computed are symmetric non-singular, but not necessarily positive definite. The first linear solver presented is based on the Cholesky factorization, modified in order to handle non definite positive matrices. The second one is based on the $\boldsymbol{LDL}^{\mathrm{t}}$ factorization.

### 2.1. Solver based on the modified Cholesky factorization used in AGIS

The linear solver based on a modified Cholesky factorization is the one currently in use in the Gaia processing pipeline. It is detailed in Lindegren et al. (2012). The factorization step is described in Algorithm 1. This algorithm overwrites the input matrix $\boldsymbol{A}$ so that the lower part of the output matrix $\boldsymbol{A}$ gives $\boldsymbol{L}$. When the matrix $\boldsymbol{A}$ is positive definite, it corresponds to the Cholesky factorization. Otherwise, the undefined values that appear during the computation are simply replaced by zeros.

The system to solve is then $\boldsymbol{LL}^{\mathrm{t}}\boldsymbol{x} = \boldsymbol{b}$, with $\boldsymbol{L} \in \mathbb{R}^{n\times n}$ lower triangular, $\boldsymbol{x} \in \mathbb{R}^n$, and $\boldsymbol{b} \in \mathbb{R}^n$. The inversion step is performed using forward and back substitution. If we note $\boldsymbol{u} = \boldsymbol{L}^{\mathrm{t}}\boldsymbol{x}$, the forward substitution consists in solving the triangular system $\boldsymbol{Lu} = \boldsymbol{b}$, and the back substitution gives the final solution by solving the triangular system $\boldsymbol{L}^{\mathrm{t}}\boldsymbol{x} = \boldsymbol{u}$. This inversion method is implemented using Algorithm 2 that overwrites the right hand side $\boldsymbol{b}$ of the equation with the solution $\boldsymbol{x}$.

It is to be noted that the result of this method is an approximation of the solution of Eq. (1) when the matrix $\boldsymbol{A}$ is not positive definite. However, as explained in Lindegren et al. (2012), this method has been proven to be accurate enough for the Gaia mission requirements.

**Algorithm 1** Modified Cholesky factorization

**Require:** $\boldsymbol{A} \in \mathbb{R}^{n\times n}$ symmetric

```
for i = 1 to n do
  ϵ ← a_ii × 10^-12
  for j = 1 to i do
    for k = 1 to j − 1 do
      a_ij ← a_ij − a_j,j−k × a_i,j−k
      if i ≠ j then
        if a_jj ≠ 0 then
          a_ij ← a_ij / a_jj
        else
          a_ij = 0
        end if
      else
        if a_ii > limit then
          a_ii ← √a_ii
        else
          a_ii = 0
        end if
      end if
    end for
  end for
end for

return lower part of A
```

**Algorithm 2** Inversion of the system $\boldsymbol{LL}^{\mathrm{t}}\boldsymbol{x} = \boldsymbol{b}$

**Require:** $\boldsymbol{L} \in \mathbb{R}^{n\times n}$ lower triangular, $\boldsymbol{b} \in \mathbb{R}^n$

```
for i = 1 to n do
  if L_ii > 0 then
    for j = 1 to i − 1 do
      b_i = b_i − L_ij b_j
    end for
    b_i = b_i / L_ii
  else
    b_i = 0
  end if
end for

for i = n to 1 do
  if L_ii > 0 then
    for j = i + 1 to n do
      b_i = b_i − L_ji b_j
    end for
    b_i = b_i / L_ii
  else
    b_i = 0
  end if
end for

return b
```

### 2.2. Solver based on the $\boldsymbol{LDL}^{\mathrm{t}}$ factorization

In this paper, we propose another solver that appears to be both more accurate and slightly faster than the one currently used by AGIS. The $\boldsymbol{LDL}^{\mathrm{t}}$ factorization handles all the symmetric non-singular matrices. It is performed by Algorithm 3, which overwrites the input matrix $\boldsymbol{A}$ with its $\boldsymbol{LDL}^{\mathrm{t}}$ decomposition. The lower part of the output matrix $\boldsymbol{A}$ gives the lower part of $\boldsymbol{L}$, and its diagonal gives $\boldsymbol{D}$.

The system to be solved is now $\boldsymbol{LDL}^{\mathrm{t}}\boldsymbol{x} = \boldsymbol{b}$, with $\boldsymbol{L} \in \mathbb{R}^{n\times n}$ unit lower triangular, $\boldsymbol{D} \in \mathbb{R}^{n\times n}$ diagonal, $\boldsymbol{x} \in \mathbb{R}^n$, and $\boldsymbol{b} \in \mathbb{R}^n$.

The inversion of such a system is realized by Algorithm 4 inspired by the Part 3.1 in Golub and Van Loan (1996). It is done in three steps: a forward substitution, a diagonal matrix inversion, and a back substitution. If we note $\boldsymbol{u} = \boldsymbol{L}^{\mathrm{t}}\boldsymbol{x}$ and $\boldsymbol{v} = \boldsymbol{D}\boldsymbol{u}$, then the three steps mentioned consist in solving successively the triangular system $\boldsymbol{L}\boldsymbol{v} = \boldsymbol{b}$, the diagonal system $\boldsymbol{D}\boldsymbol{u} = \boldsymbol{v}$, and the triangular system $\boldsymbol{L}^{\mathrm{t}}\boldsymbol{x} = \boldsymbol{u}$. In terms of memory management, at the end of Algorithm 4 the vector $\boldsymbol{b}$ is replaced by the solution $\boldsymbol{x}$.

**Algorithm 3** LDLT factorization

**Require:** $\boldsymbol{A} \in \mathbb{R}^{n\times n}$ symmetric non-singular

```
for k = 1 to n − 1 do
  for j = k + 1 to n do
    v ← a_jk / a_kk
    for i = j to n do
      a_ij ← a_ij − a_ik × v
    end for
    a_jk ← v
  end for
end for

return lower part of A
```

**Algorithm 4** Inversion of the system $\boldsymbol{L}\boldsymbol{D}\boldsymbol{L}^{\mathrm{t}}\boldsymbol{x} = \boldsymbol{b}$

**Require:** $\boldsymbol{L} \in \mathbb{R}^{n\times n}$ unit lower triangular, $\boldsymbol{D} \in \mathbb{R}^{n\times n}$ diagonal, $\boldsymbol{b} \in \mathbb{R}^{n}$

```
for i = 2 to n do
  for j = 1 to i − 1 do
    b_i = b_i − L_ij b_j
  end for
end for

for i = 1 to n do
  b_i = b_i / D_ii
end for

for i = n − 1 to 1 do
  for j = i + 1 to n do
    b_i = b_i − L_ji b_j
  end for
end for

return b
```

## 3. GPU implementation

Figs. 1 and 2 illustrate the basic difference between a CPU and a GPU in terms of hardware architecture. A CPU is composed of a control unit that interprets the instructions, a cache memory that stores the data to be processed, and a computing unit that realizes the operations. Nowadays, most of the CPUs contain two or four computing units that can be used in parallel. A GPU contains hundreds of computing units that can run in parallel. They are organized into several groups that share the same control unit and cache memory. A detailed comparison can be found in Owens et al. (2008).

The program developed to solve the linear systems is written in C for the CPU part and in CUDA for the GPU part. CUDA (short for *Compute Unified Device Architecture*) is a programming model created by the GPU manufacturer Nvidia (2012). It is based on a programming language, C for CUDA, created as an extension of the C language.

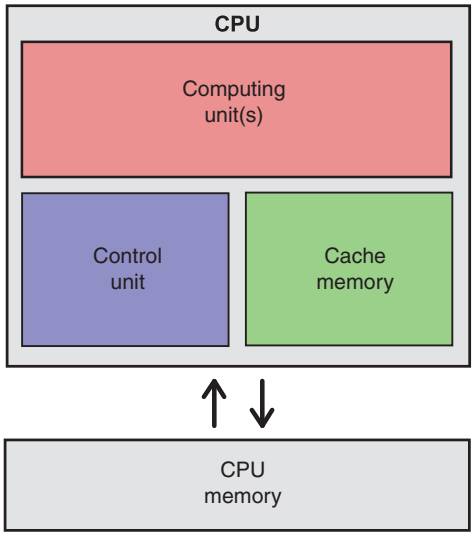



**Fig. 1.** Simplified CPU architecture.

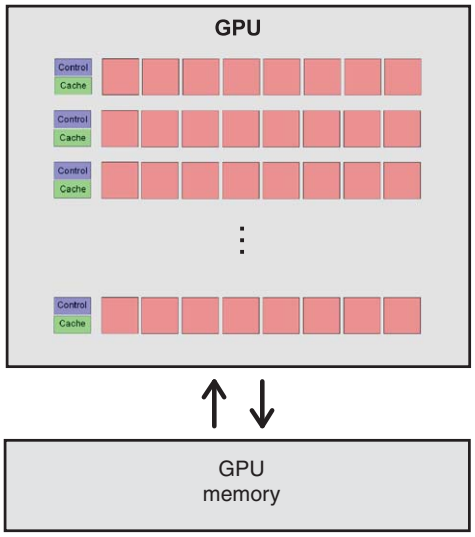



**Fig. 2.** Simplified GPU architecture.

The CPU based implementation is not parallelized for a multicore usage because it is used to have a sequential reference time to be compared to the GPU computing time.

### 3.1. Implementation on CPU

First, the input data are read from a database. Fig. 3 illustrates the way data are stored in the CPU memory. The lower part of every input matrix is stored in lexicographic order as a single precision value. These linear representations are gathered in a single table. The vectors are stored in another table with the same pattern.

The $N$ linear systems are then solved sequentially using Algorithms 1 and 2, or Algorithms 3 and 4. The resulting vectors are stored in a table with the same organization as the input vectors. The computing time is measured only during this stage.

The resulting table is then stored in a database.

In fact, this simple implementation will serve as a reference for comparison with the following ones.

### 3.2. Implementation on the GPU

The computing capacity of a GPU is theoretically much higher than the one of a CPU of the same price range (see the details of

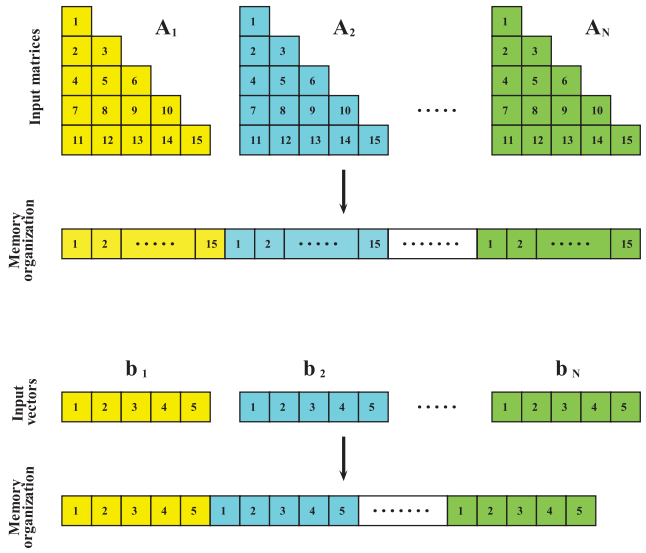



**Fig. 3.** Data storage by concatenation of vectors and lexicographically vectorized matrices.

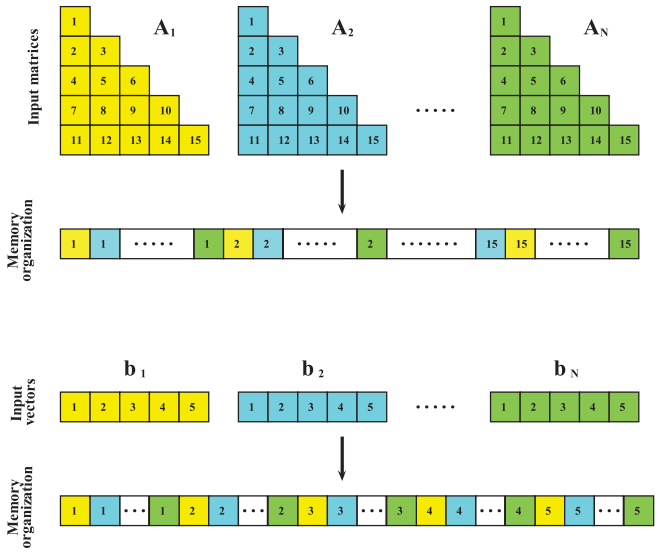



**Fig. 4.** Element-wise organization.

the workstation used in Section 4). But the memory access latency can substantially reduce its performance. In this part, we discuss several memory management issues to use the potential of a GPU computing to the maximum.

For the two linear solvers presented in Section 2, several implementations have been done to explore how computation time is affected not only by the arithmetic complexity of the method used, but also by the pattern of memory accesses and transfers between GPU and CPU. The four versions hereafter are presented in an increasing order of optimization, each one being an improvement of the previous one.

### 3.2.1. Version 1: CPU-like implementation

The first GPU implementation is quite close to the CPU one. First, the data are loaded in the CPU memory with the same organization, then transferred to the GPU memory. On the GPU, as many threads as systems to solve are launched. Each thread runs the resolution on a different matrix–vector couple. The CUDA scheduler automatically runs as many threads in parallel as the device can support. When all the threads have terminated, the results are transferred to the CPU memory, and finally saved in an output database. For comparison with the CPU implementation, the measured time includes the computing part and the transfers between CPU and GPU memories.

### 3.2.2. Version 2: element-wise organization

The GPU works as a SIMD (Single Instruction Multiple Data) tool (Owens et al., 2008; Nvidia, 2012). This means that at a given time, several threads are executing the same instruction on some data from different memory locations. They are all reading or writing the same element of different matrix or vector. However, it is asserted in Kirk et al. (2010) that coalesced memory transactions must be used when possible. Indeed, it is much faster to access contiguous addresses in the GPU memory than random ones. That is why all the elements of the same index, for all the matrices or vectors, should be stored in contiguous addresses. This organization is illustrated in Fig. 4. In practice, the scheduler can allocate several threads at the same time on the same computing unit in order to hide memory access latency (Kirk et al., 2010). Despite this internal optimization, latency remains a major issue in version 1 and justifies the solution adopted in version 2, as shown in the Section 4.

Thus, this version of the program imports the data in the CPU memory according to the new organization. The tables are then transferred to the GPU memory, computed as in version 1. The result is transferred back to the CPU memory, and written in a database with the original organization.

For this version to be gainful, it is important to ensure that the time saved is larger than the time needed for reordering the data while loading and saving it. In the application presented in 4, no additional cost has been observed during these stages.

### 3.2.3. Version 3: pinned memory usage

In version 2, the time required by the computation is reduced, while the memory transfers time stays unchanged. Using the pinned memory is a way to speed up the transfers. Usually, when a variable is stored in the CPU memory, the operating system has the possibility to swap it to a secondary storage device if needed. During a transfer from the CPU to the GPU memory, a copy of the variable is first made in the pinned CPU memory, which means that it cannot be moved. Then this temporary variable is copied into the GPU memory. CUDA enables the programmer to allocate directly the variables in the pinned memory so that the first copy is not performed any more. Hence, the only difference between Versions 2 and 3 is the type of CPU memory used.

### 3.2.4. Version 4: data splitting

This version uses the capability of the graphic card to concurrently copy memory and run a kernel using Cuda Streams (Nvidia, 2012). The original data are split into two groups of matrix–vector couples. These two groups are independently stored in separate variables, using the same strategy described in Section 3.2.3. In this way it is possible to compute the first group while transferring the second to the GPU, and to compute the second while transferring the results of the first to the CPU. The transfer rate is a bit slower because less data are transferred at a time, but the computation is partially hidden because it is done during transfers. The advantage of this method is illustrated in Fig. 5.

## 4. Experimental results

### 4.1. Experiment conditions

The systems to solve are defined by a set of $N = 250531$ symmetric non-singular matrices of size $n \times n = 5 \times 5$, and

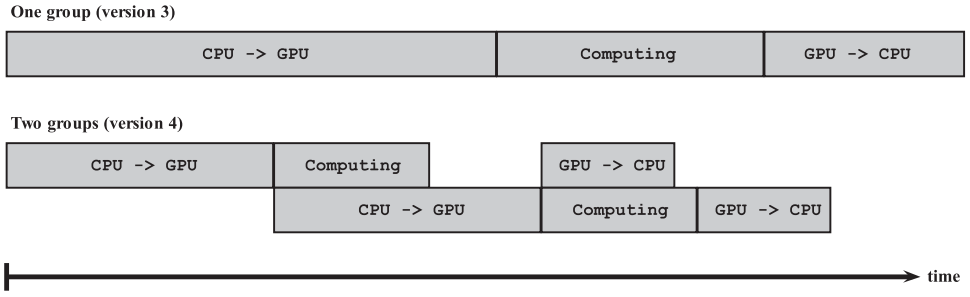



**Fig. 5.** Advantage of splitting data into two groups using CUDA Streams.

**Table 1**
Computing times and gain factors of all the implementations realized for the linear solver based on the *modified Cholesky* factorization in *single* precision.

| CPU time | 145 ms | | | |
|---|---|---|---|---|
| *GPU time* | *Version* 1 | *Version* 2 | *Version* 3 | *Version* 4 |
| Transfer CPU to GPU | 7.7 ms | 7.7 ms | 3.4 ms | n.a. |
| Computing | 23.7 ms | 3.8 ms | 3.8 ms | n.a. |
| Transfer GPU to CPU | 2.4 ms | 2.4 ms | 1.0 ms | n.a. |
| *Total* | **33.8** ms | **13.9** ms | **8.2** ms | **6.7** ms |
| *Gain factor* | **4.3** | **10.4** | **17.7** | **21.6** |

**Table 2**
Computing times and gain factors of all the implementations realized for the linear solver based on the *modified Cholesky* factorization in *double* precision.

| CPU time | 145 ms | | | |
|---|---|---|---|---|
| *GPU time* | *Version* 1 | *Version* 2 | *Version* 3 | *Version* 4 |
| Transfer CPU to GPU | 15.0 ms | 15.0 ms | 6.7 ms | n.a. |
| Computing | 22.4 ms | 5.9 ms | 5.9 ms | n.a. |
| Transfer GPU to CPU | 4.4 ms | 4.4 ms | 1.7 ms | n.a. |
| *Total* | **41.8** ms | **25.3** ms | **14.3** ms | **11.4** ms |
| *Gain factor* | **3.5** | **5.7** | **10.1** | **12.7** |

**Table 3**
Computing times and gain factors of all the implementations realized for the linear solver based on the $\boldsymbol{LDL}^t$ factorization in *single* precision.

| CPU time | 136 ms | | | |
|---|---|---|---|---|
| *GPU time* | *Version* 1 | *Version* 2 | *Version* 3 | *Version* 4 |
| Transfer CPU to GPU | 7.7 ms | 7.7 ms | 3.4 ms | n.a. |
| Computing | 21.6 ms | 3.5 ms | 3.5 ms | n.a. |
| Transfer GPU to CPU | 2.4 ms | 2.4 ms | 1.0 ms | n.a. |
| *Total* | **31.7** ms | **13.6** ms | **7.9** ms | **6.2** ms |
| *Gain factor* | **4.4** | **10.0** | **17.2** | **21.9** |

**Table 4**
Computing times and gain factors of all the implementations realized for the linear solver based on the $\boldsymbol{LDL}^t$ factorization in *double* precision.

| CPU time | 136 ms | | | |
|---|---|---|---|---|
| *GPU time* | *Version* 1 | *Version* 2 | *Version* 3 | *Version* 4 |
| Transfer CPU to GPU | 15.0 ms | 15.0 ms | 6.7 ms | n.a. |
| Computing | 21.2 ms | 5.5 ms | 5.5 ms | n.a. |
| Transfer GPU to CPU | 4.4 ms | 4.4 ms | 1.7 ms | n.a. |
| *Total* | **40.6** ms | **24.9** ms | **13.9** ms | **11.1** ms |
| *Gain factor* | **3.3** | **5.5** | **9.8** | **12.3** |

the same number of vectors of length $n$, provided by the Gaia Science Ground Segment, corresponding to simulations of the data expected from the real mission. As mentioned in the introduction, the real data measurements will result in more than $10^9$ systems to solve at each iteration of an optimization process. So this example is a very small subset of the expected real data.

During this study, the following workstation is used:

- *DELL Precision T7400*
- CPU: 2x Intel Xeon X5472 (3.00 GHz)
- DRAM: 16 GB
- OS: Linux, Ubuntu 10.04.

This computer embeds the following *Nvidia Tesla C1060* GPU card whose technical documentation is available online.[1] Its main features are:

- 1 T GPU (240 processor cores)
- 1.296 GHz Processor core clock
- 4 GB GDDR3 Global memory
- IEEE 754 single and double precision floating point
- Ultra-fast memory access with 102 GB/s peak bandwidth per GPU.

### 4.2. Results

The two linear solving methods presented in Section 2 are both implemented according to the five implementation strategies presented in Section 3 (one CPU version and four GPU versions). Single and double precision representations are considered. All the computation times and gain factors between CPU and GPU are reported in Tables 1–4. Multiple runs exhibited a variability of several milliseconds for the CPU time, and less than 0.1 ms for the GPU time. All the results presented contain average values for 10 runs. Detailed execution time is not available for version 4 due to the use of CUDA Streams.

We can first notice that every version of the GPU implementation is faster than the CPU one, the most optimized version having a gain factor of 12.3–21.9 depending on the conditions.

The solver based on the $\boldsymbol{LDL}^t$ factorization is faster than the one based on the modified Cholesky factorization. Indeed, although the $\boldsymbol{LDL}^t$ and Cholesky factorizations have exactly the same arithmetic complexity, the verification steps added in the modified Cholesky algorithm slow it down. However, the gain factors observed when using a GPU are very similar for the two methods tested.

In version 4 of the GPU implementation, the data transferred have a total size of 25 MB in single precision (resp. 50 MB in double precision) and they are transferred during less than 7 ms (resp. 12 ms). This corresponds to a transfer rate higher than 3.5 GB/s. The graphic card is connected to the CPU through a PCIe x16 (gen2) connection, that has a maximal transfer rate of 8 GB/s (Specification Board, 0000). So the limiting factor, which is the transfer rate, cannot be much improved because it is already close to its theoretical maximum. Thus this version is the best we can have when the limiting factor is the transfer rate.

### 4.3. Single vs. double precision

Recent graphics cards like the one used for this study support double precision computing, but the gain factor observed is lower

[1] http://www.nvidia.com/object/personal-supercomputing.html.

**Table 5**
Average relative errors for the linear solver using the modified Cholesky factorization.

| | | |
|---|---|---|
| Single precision | CPU version | AvRE = 6.2 |
| | GPU versions | AvRE = 6.2 |
| Double precision | CPU version | AvRE = 6.2 |
| | GPU versions | AvRE = 6.2 |

**Table 6**
Average relative errors for the linear solver using the $\boldsymbol{LDL}^{\mathrm{t}}$ factorization.

| | | |
|---|---|---|
| Single precision | CPU version | AvRE $= 4.7 \times 10^{-7}$ |
| | GPU versions | AvRE $= 8.2 \times 10^{-7}$ |
| Double precision | CPU version | AvRE $= 1.0 \times 10^{-15}$ |
| | GPU versions | AvRE $= 9.4 \times 10^{-16}$ |

than in single precision for two reasons. First, the transfer time is doubled because the data need twice as much memory space. Second, the computing part also slows down because it is realized by double precision cores which are fewer than single precision cores (30 double precision and 240 single precision cores on the Tesla C1060). However, this difference should be bigger in the case of a computation bound execution (the ratio between single and double computation capability is about 8; (Nvidia, 2012)). This means that this algorithm is memory bound, the memory access latency is the bottleneck of the computing parts. The CPU, on the other hand, shows the same computing time regardless of the precision because it is designed for double precision computing, and is just partially used when processing single precision numbers (SSE instructions allowing parallel single precision computation using one double precision processor are not used).

To check the difference in terms of result accuracy, an independent program has been created. It reads the original data and the solutions, and computes the average relative error defined as

$$\mathrm{AvRE} = \frac{1}{N}\frac{1}{n}\sum_{k=1}^{N}\sum_{i=1}^{n}\left|\frac{[\boldsymbol{A}_k\boldsymbol{x}_k]_i - [\boldsymbol{b}_k]_i}{[\boldsymbol{b}_k]_i}\right|.$$

These errors are reported in Tables 5 and 6. For the method using the modified Cholesky algorithm, all of the matrices used appeared to be non positive definite. That results in a deterioration of the average relative error for all precisions. For the method using the $\boldsymbol{LDL}^{\mathrm{t}}$ algorithm, the reconstruction error is improved a lot by using double precision, but it also affects the GPU computing time.

### 4.4. Shared memory usage

A GPU contains different levels of memory. The versions presented above only use the global memory (GPU memory in Fig. 2), which is the largest but also the slowest. According to Kirk et al. (2010), using the shared memory can speedup the computing time in many cases. The shared memory (Cache in Fig. 2) is faster but has a limited size. In this case, the idea is to modify the kernel of version 4 in such a way that to solve one system, the corresponding data are first transferred from the global to the shared memory, then computed, and finally the result is transferred from the shared to the global memory. In this way, the data are read from or written to the global memory only once, and all the intermediate memory accesses are fast because they are realized with a cache memory.

This idea has been implemented, but shows unsatisfactory performance. This is due to the limited size of the shared memory. The scheduler always runs as many threads in parallel as the device can support. In this case, the shared memory size sets this limit. The scheduler has to reduce the number of threads running at the same time. So although the threads run faster, the overall computing time increases. Using the shared memory would be more adapted in the case of a kernel requiring more memory accesses per data and/or involving less data.

New generations of Nvidia GPUs contain three times more shared memory than the Tesla C1060 generation (Nvidia, 2012). Using the shared memory on the devices could be profitable. But as long as the limiting factor is the transfer rate, such improvement will result in a minor speedup.

## 5. Integration in the existing data processing pipeline

The integration of GPU-based solutions into an existing system has been studied mainly from a systems engineering perspective. While it is clear that GPU technology brings a boost in performance at relatively low cost for the hardware, it is less clear what other costs there are to consider. We report some aspects that concern the functioning of a ground segment as a whole. The *verification* infrastructure has to resemble the production infrastructure more closely than would be required for a wholly Java-based solution where programming semantics are more machine-independent. The programming staff has to be *trained* in a wider variety of skills and techniques. When the core components of a system are designed by a committee, the added complexity of a system might pose a challenge to the structure of *responsibilities*.

## 6. Conclusion

This study has examined the improvements in performance that can be expected by running a part of Gaia's AGIS—namely, the LU-decomposition of the $5 \times 5$ matrices—on GPUs. First, an algorithm both more accurate and faster than the one currently used in AGIS has been proposed. Second, an optimization of the GPU implementation has been realized. The achievable gain factor is dependent on the chosen algorithm and desired floating point precision; however, with a range of roughly 12–22 is clearly significant in all considered cases. Those figures being very hardware dependent, the most important part of this paper is the optimization processed realized, and not the gain factor itself. In particular, the new generations of Nvidia GPUs may result in higher speed ups. This, however, does not mean that AGIS as a whole would be speeded up by these factors as the decomposition is not the most time consuming part of the system. At the moment AGIS is entirely written in Java (O'Mullane et al., 2011) and as such can be ported seamlessly to any platform that can run a Java Virtual Machine. In this light, the usage of GPUs still remains an attractive option but it is clear that is would come at the price of much diminished portability. This disadvantage may largely disappear with the introduction of native GPU support planned for Java 9 (Oracle lays out long-range Java intentions, 2013). As this analysis validates the usefulness of GPUs on one stage of a large scale optimization problem, further work will study the applicability of such a solution to the entire problem.

## References

Benzi, M., 2002. Preconditioning techniques for large linear systems: a survey. J. Comput. Phys. 182 (2), 418–477.

Golub, G., Van Loan, C., 1996. Matrix Computations, Vol. 3. Johns Hopkins Univ. Pr.

Kirk, D., Wen-mei, W., Hwu, W., 2010. Programming Massively Parallel Processors: A Hands-on Approach. Morgan Kaufmann.

Lindegren, L., Babusiaux, C., Bailer-Jones, C., Bastian, U., Brown, A.G., Cropper, M., Høg, E., Jordi, C., Katz, D., Van Leeuwen, F., et al., 2008. The gaia mission: science, organization and present status. Proc. Int. Astron. Union 248, 217.

Lindegren, L., Lammers, U., Hobbs, D., O'Mullane, W., Bastian, U., Hernández, J., 2012. The astrometric core solution for the gaia mission. Astron. Astrophys. 538.

Nvidia, C., 2012. NVIDIA CUDA C Programming Guide Version 4.2.

O'Mullane, W., Lammers, U., Lindegren, L., Hernández, J., Hobbs, D., 2011. Implementing the gaia astrometric global iterative solution (AGIS) in java. Exp. Astron. 31, 215–241.

Oracle lays out long-range Java intentions. 2013. http://openjdk.java.net/projects/sumatra/.

Owens, J., Houston, M., Luebke, D., Green, S., Stone, J., Phillips, J., 2008. GPU computing. Proc. IEEE 96 (5), 879–899.

Specification Board. Tesla C1060 computing processor board, NVIDIA Corporation, Santa Clara, USA.

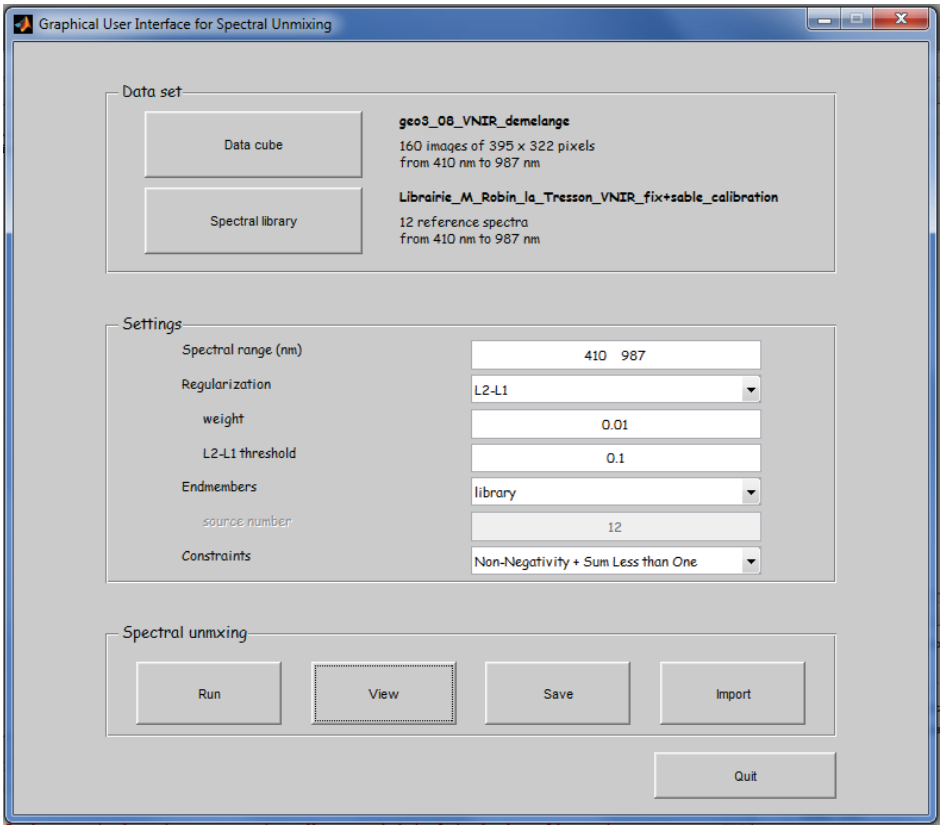
Tomov, S., Nath, R., Ltaief, H., Dongarra, J., 2010. Dense linear algebra solvers for multicore with GPU accelerators. In: 2010 IEEE International Symposium on Parallel & Distributed Processing, Workshops and Ph.D. Forum, IPDPSW. IEEE, pp. 1–8.

Volkov, V., Demmel, J., 2008. LU, QR and Cholesky Factorizations Using Vector Capabilities of GPUs, Tech. Rep. UCB/EECS-2008-49. EECS Department, University of California, Berkeley, pp. 2008–2049.

# Annexe C

# Documentation de l'interface graphique

# Graphical User Interface for Spectral Unmixing




## 1 – Data set

### Data cube

Browse the header file (*.hdr*) corresponding to a data cube generated by ENVI. The data file, with the same name and without extension, must exist in the same folder.

### Spectral library

Same as *Data cube* loading. Format of the data: "number of samples" in *Data cube* becomes "number of endmembers" in *Spectral library*, "number of lines" becomes "number of bands", "number of bands" is set to 1. Needed only if *Settings->Endmembers* is set to *library*.

## 2 – Settings

### Spectral range (nm)

By default: largest common spectral range between the data and the library (if a library is used). The range can be reduced manually.

### Regularization
**no**: No spatial regularization

**L2**: Spatial regularization of type L2. The differences of abundance between neighboring pixels are penalized with a quadratic function. It reduces the noise. It requires more computing time than the non-regularized case.

**L2-L1**: Spatial regularization of type L2-L1. The differences of abundance between neighboring pixels are penalized with a quadratic-linear function (quadratic for small differences, linear for large differences). It reduces the noise while preserving the edges. It requires more computing time than the L2-regularized case.

### weight
Regularization strength. Increasing this value reduces the noise, but increases the computation time.

### L2-L1 threshold
Threshold between small abundance differences between neighboring pixels, considered as noise, and large differences, considered edges. A small value results in a higher computing time.

### Endmembers
**library**: uses spectral library loaded in the part Data set

**vca**: uses endmembers extracted from the data cube by the VCA method.

**nfindr**: uses endmembers extracted from the data cube by the NFINDR method.

### source number
set the number of endmembers extracted from the data cube if vca of nfindr is chosen.

### Constraints
Sets the abundance constraints used for unmixing.

## 3 – Spectral unmixing

### Run
Runs the unmixing algorithm IPLS (Interior Points Least Squares), possibly preceded by VCA or NFINDR algorithm.

### View
Plots the endmembers and their corresponding abundance maps obtained after unmixing. It also displays the spatial and spectral SNR calculated as the ratio Signal/Residual.

### Save
Saves results in a *.mat* file. This file can be opened in Matlab for further analysis.

### Import
Imports results previously saved to display it with the *View* button.

# Bibliographie


ARDOUIN, J., LÉVESQUE, J. et REA, T. (2007). A demonstration of hyperspectral image exploitation for military applications. *In IEEE International Conference on Information Fusion*, pages 1–8. IEEE. 17

ARMAND, P., BENOIST, J. et DUSSAULT, J. (2012). Local path-following property of inexact interior methods in nonlinear programming. *Computational Optimization and Applications*, 52(1):209–238. 70, 75, 76, 77

ARMAND, P., BENOIST, J. et ORBAN, D. (2013). From global to local convergence of interior methods for nonlinear optimization. *Optimization Methods and Software*, 28(5):1051–1080. 41

ARMAND, P., GILBERT, J. et JAN-JÉGOU, S. (2000). A feasible BFGS interior point algorithm for solving convex minimization problems. *SIAM Journal on Optimization*, 11(1):199–222. 39, 41, 43

ASANOVIC, K., BODIK, R., CATANZARO, B., GEBIS, J., HUSBANDS, P., KEUTZER, K., PATTERSON, D. A., PLISHKER, W., SHALF, J. et WILLIAMS, S. (2006). The landscape of parallel computing research : A view from berkeley. Rapport technique, Technical Report UCB/EECS-2006-183, EECS Department, University of California, Berkeley. 82

BERGSTRA, J. et BENGIO, Y. (2012). Random search for hyper-parameter optimization. *The Journal of Machine Learning Research*, 13(1):281–305. 113

BERTERO, M. et BOCCACCI, P. (1998). *Introduction to inverse problems in imaging*. CRC press. 19

BIBRING, J., SOUFFLOT, A., BERTHÉ, M., LANGEVIN, Y., GONDET, B., DROSSART, P., BOUYÉ, M., COMBES, M., PUGET, P. et SEMERY, A. (2004). OMEGA : Observatoire pour la minéralogie, l'eau, les glaces et l'activité. *In Mars Express : The Scientific Payload*, volume 1240, pages 37–49. 99, 102

BIOUCAS-DIAS, J. et FIGUEIREDO, M. (2010). Alternating direction algorithms for constrained sparse regression : Application to hyperspectral unmixing. *In 2nd Workshop on Hyperspectral Image and Signal Processing : Evolution in Remote Sensing (WHISPERS)*, pages 1–4. IEEE. 29, 37, 38

BIOUCAS-DIAS, J. et PLAZA, A. (2011). An overview on hyperspectral unmixing : geometrical, statistical, and sparse regression based approaches. *In IEEE International Geoscience and Remote Sensing Symposium (IGARSS)*, pages 1135–1138. IEEE. 24

BIOUCAS-DIAS, J., PLAZA, A., DOBIGEON, N., PARENTE, M., DU, Q., GADER, P. et CHANUSSOT, J. (2012). Hyperspectral unmixing overview : Geometrical, statistical, and sparse regression-based approaches. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 5(2):354–379. 18

BOURGUIGNON, S., MARY, D. et SLEZAK, E. (2012). Processing MUSE hyperspectral data : Denoising, deconvolution and detection of astrophysical sources. *Statistical Methodology*, 9(1):32–43. 18

BOYD, S., PARIKH, N., CHU, E., PELEATO, B. et ECKSTEIN, J. (2011). Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends® in Machine Learning*, 3(1):1–122. 35

BOYD, S. et VANDENBERGHE, L. (2004). *Convex optimization*. Cambridge university press. 29

BRO, R. et DE JONG, S. (1997). A fast non-negativity-constrained least squares algorithm. *Journal of chemometrics*, 11(5):393–401. 29, 31

BYRD, R., HRIBAR, M. et NOCEDAL, J. (1999). An interior point algorithm for large-scale nonlinear programming. *SIAM Journal on Optimization*, 9(4):877–900. 113

CHANG, C. (2007). *Hyperspectral data exploitation : theory and applications*. John Wiley & Sons. 14, 24

CHANG, C. et HEINZ, D. (2000). Constrained subpixel target detection for remotely sensed imagery. *IEEE Transactions on Geoscience and Remote Sensing*, 38(3):1144–1159. 34

CHOUZENOUX, E., LEGENDRE, M., MOUSSAOUI, S. et IDIER, J. (2014). Fast constrained least squares spectral unmixing using primal-dual interior-point optimization. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 7(1):59–69. 47, 64

CHOUZENOUX, E., MOUSSAOUI, S., LEGENDRE, M. et IDIER, J. (2013). Algorithme primal-dual de points intérieurs pour l'estimation pénalisée des cartes d'abondances en imagerie hyperspectrale. *Traitement du signal*, 30(1-2):35–59. 47, 79

CLARK, R., SWAYZE, G., GALLAGHER, A., KING, T. et CALVIN, W. (1993). The U.S. geological survey digital spectral library : version 1 : 0.2 to 3.0 $\mu m$. *U.S. Geological Survey, Denver, CO, Open File Rep. 93-592*. 24, 48

DEMBO, R. et STEIHAUG, T. (1983). Truncated-newton algorithms for large-scale unconstrained optimization. *Mathematical Programming*, 26(2):190–212. 70

DIAZ, J., MUNOZ-CARO, C. et NINO, A. (2012). A survey of parallel programming models and tools in the multi and many-core era. *IEEE Transactions on Parallel and Distributed Systems*, 23(8):1369–1386. 84

DOBIGEON, N., MOUSSAOUI, S., COULON, M., TOURNERET, J. et HERO, A. (2009). Joint bayesian endmember extraction and linear unmixing for hyperspectral imagery. *IEEE Transactions on Signal Processing*, 57(11):4355–4368. 24

DOBIGEON, N., TOURNERET, J., RICHARD, C., BERMUDEZ, J., MCLAUGHLIN, S. et HERO, A. (2014). Nonlinear unmixing of hyperspectral images : Models and algorithms. *Signal Processing Magazine, IEEE*, 31(1):82–94. 113

ECKSTEIN, J. et BERTSEKAS, D. (1992). On the douglas—rachford splitting method and the proximal point algorithm for maximal monotone operators. *Mathematical Programming*, 55(1-3):293–318. 38

EL-BAKRY, A., TAPIA, R. A., TSUCHIYA, T. et ZHANG, Y. (1996). On the formulation and theory of the newton interior-point method for nonlinear programming. *Journal of Optimization Theory and Applications*, 89(3):507–541. 42

FAUVEL, M., BENEDIKTSSON, J., CHANUSSOT, J. et SVEINSSON, J. (2008). Spectral and spatial classification of hyperspectral data using svms and morphological profiles. *IEEE Transactions on Geoscience and Remote Sensing*, 46(11):3804–3814. 18

FIACCO, A. et MCCORMICK, G. (1968). *Nonlinear programming : sequential unconstrained minimization techniques*, volume 4. Society for Industrial and Applied Mathematics (SIAM). 39

FORSGREN, A., GILL, P. et WRIGHT, M. (2002). Interior methods for nonlinear optimization. *SIAM review*, 44(4):525–597. 41

FRISCH, K. (1955). The logarithmic potential method of convex programming. *Memorandum, University Institute of Economics, Oslo*. 39

GABAY, D. et MERCIER, B. (1976). A dual algorithm for the solution of nonlinear variational problems via finite element approximation. *Computers & Mathematics with Applications*, 2(1):17–40. 35

GLOWINSKI, R. et MARROCO, A. (1975). Sur l'approximation, par éléments finis d'ordre un, et la résolution, par pénalisation-dualité d'une classe de problèmes de dirichlet non linéaires. *Revue Française d'Automatique, Informatique, et Recherche Opérationelle*, 9(R2):41–76. 35

GOLUB, G. et ORTEGA, J. (2014). *Scientific computing : an introduction with parallel computing*. Elsevier. 82

GOLUB, G. et VAN LOAN, C. (1996). *Matrix computations*, volume 3. Johns Hopkins Univ Pr. 88

GORDON, R. et HERMAN, G. (1971). Reconstruction of pictures from their projections. *Communications of the ACM*, 14(12):759–768. 19

GRAY, R. (2006). *Toeplitz and circulant matrices : A review*. Now Publishers Inc. 74

GREEN, R., CONEL, J. et ROBERTS, D. (1993). Estimation of aerosol optical depth, pressure elevation, water vapor, and calculation of apparent surface reflectance from radiance measured by the airborne visible/infrared imaging spectrometer (AVIRIS) using a radiative transfer code. *In Optical Engineering and Photonics in Aerospace Sensing*, pages 2–11. International Society for Optics and Photonics. 17

HADAMARD, J. (1902). Sur les problèmes aux dérivées partielles et leur signification physique. *Princeton University Bulletin*, 13(49-52):28. 27

HEINZ, D. et CHANG, C. (2001). Fully constrained least squares linear spectral mixture analysis method for material quantification in hyperspectral imagery. *IEEE Transactions onGeoscience and Remote Sensing*, 39(3):529–545. 29, 34

HENROT, S. (2013). *Déconvolution et séparation d'images hyperspectrales en microscopie*. Thèse de doctorat, Université de Lorraine. 18

HUCK, A., GUILLAUME, M. et BLANC-TALON, J. (2010). Minimum dispersion constrained nonnegative matrix factorization to unmix hyperspectral data. *IEEE Transactions on Geoscience and Remote Sensing*, 48(6):2590–2602. 24

HUNTER, D. et LANGE, K. (2004). A tutorial on MM algorithms. *The American Statistician*, 58(1):30–37. 70, 71, 72

IBMC, I. (1981). Color/graphics adapter. Rapport technique, Tech. Rep. 83

IDIER, J. (2013). *Bayesian approach to inverse problems*. John Wiley & Sons. 18, 28, 57

IORDACHE, M., BIOUCAS-DIAS, J. et PLAZA, A. (2012). Total variation spatial regularization for sparse hyperspectral unmixing. *IEEE Transactions on Geoscience and Remote Sensing*, 50(11):4484–4502. 29, 38, 56

JANSSON, P. (2014). *Deconvolution of images and spectra*. Courier Corporation. 19

JIA, S. et QIAN, Y. (2009). Constrained nonnegative matrix factorization for hyperspectral unmixing. *IEEE Transactions on Geoscience and Remote Sensing*, 47(1):161–173. 24

JOHNSON, C., SEIDEL, J. et SOFER, A. (2000). Interior-point methodology for 3-D PET reconstruction. *IEEE Transactions on Medical Imaging*, 19(4). 75

KALMIKOV, A. et HEIMBACH, P. (2014). A hessian-based method for uncertainty quantification in global ocean state estimation. *SIAM Journal on Scientific Computing*, 36(5):S267–S295. 102

KARMARKAR, N. (1984). A new polynomial-time algorithm for linear programming. *In Proceedings of the sixteenth annual ACM symposium on Theory of computing*, pages 302–311. ACM. 29, 39

KESHAVA, N. et MUSTARD, J. (2002). Spectral unmixing. *IEEE Signal Processing Magazine*, 19(1):44–57. 11, 18, 24, 25, 26

KIRK, D., WEN-MEI, W. et HWU, W. (2010). *Programming massively parallel processors : a hands-on approach*. Morgan Kaufmann. 84, 87, 94

KNIGHT, E. et KVARAN, G. (2014). Landsat-8 operational land imager design, characterization and performance. *Remote Sensing*, 6(11):10286–10305. 14

LACAR, F., LEWIS, M. et GRIERSON, I. (2001). Use of hyperspectral imagery for mapping grape varieties in the barossa valley, south australia. *In IEEE International Geoscience and Remote Sensing Symposium IGARSS*, volume 6, pages 2875–2877. IEEE. 17

LAWSON, C. et HANSON, R. (1974). *Solving least squares problems*, volume 161. SIAM. 29

LEGENDRE, M., MOUSSAOUI, S., CHOUZENOUX, E. et IDIER, J. (2014). Primal-dual interior-point optimization based on majorization-minimization for edge-preserving spectral unmixing. *In IEEE International Conference on Image Processing*. 64

LEGENDRE, M., MOUSSAOUI, S., SCHMIDT, F. et IDIER, J. (2013a). Implémentation parallèle d'une méthode d'estimation des cartes d'abondances en imagerie hyperspectrale. *In Colloque GRETSI pour le Traitement du Signal et des Images*, pages CD–ROIM. 81

LEGENDRE, M., MOUSSAOUI, S., SCHMIDT, F. et IDIER, J. (2013b). Parallel implementation of a primal-dual interior-point optimization method for fast abundance maps estimation. *In Workshop on Hyperspectral Image and Signal Processing : Evolution in Remote Sensing (WHISPERS)*. 81

LEGENDRE, M., SCHMIDT, A., MOUSSAOUI, S. et LAMMERS, U. (2013c). Solving systems of linear equations by GPU-based matrix factorization in a science ground segment. *Astronomy and Computing*, 3:58–64. 81, 87

LIN, C. (2007). Projected gradient methods for nonnegative matrix factorization. *Neural computation*, 19(10):2756–2779. 29

LINDEGREN, L., BABUSIAUX, C., BAILER-JONES, C., BASTIAN, U., BROWN, A. G., CROPPER, M., HØG, E., JORDI, C., KATZ, D., VAN LEEUWEN, F. *et al.* (2008). The gaia mission : science, organization and present status. *Proceedings of the International Astronomical Union*, 248:217. 88

LINDEGREN, L., LAMMERS, U., HOBBS, D., O'MULLANE, W., BASTIAN, U. et HERNÁNDEZ, J. (2012). The astrometric core solution for the gaia mission. *Astronomy & Astrophysics*, 538. 88

LIU, D. et NOCEDAL, J. (1989). On the limited memory bfgs method for large scale optimization. *Mathematical programming*, 45(1-3):503–528. 113

MANOLAKIS, D., MARDEN, D. et SHAW, G. (2003). Hyperspectral image processing for automatic target detection applications. *Lincoln Laboratory Journal*, 14(1):79–116. 18

MARSHALL, A., OLKIN, I. et ARNOLD, B. (2010). *Inequalities : Theory of majorization and its applications : Theory of majorization and its applications*. Springer Science & Business Media. 113

MEHROTRA, S. (1992). On the implementation of a primal-dual interior point method. *SIAM Journal on optimization*, 2(4):575–601. 39

MICROIMAGES, I. (2012). Introduction to hyperspectral imaging with TNTmips. 11, 17

MOLINA, R., KATSAGGELOS, A. et MATEOS, J. (1999). Bayesian and regularization methods for hyperparameter estimation in image restoration. *IEEE Transactions on Image Processing*, 8(2):231–246. 113

MOUSSAOUI, S., CHOUZENOUX, E. et IDIER, J. (2012). Spectral unmixing using constrained penalized least squares estimation and primal-dual interior point optimization. *In Proc. 4th IEEE Workshop on Hyperspectral Image and Signal Processing : Evolution in Remote Sensing*. 70

MOUSSAOUI, S., HAUKSDOTTIR, H., SCHMIDT, F., JUTTEN, C., CHANUSSOT, J., BRIE, D., DOUTÉ, S. et BENEDIKTSSON, J. (2008). On the decomposition of mars hyperspectral data by ICA and bayesian positive source separation. *Neurocomputing*, 71(10):2194–2208. 24

NASCIMENTO, J. et BIOUCAS-DIAS, J. (2009). Nonlinear mixture model for hyperspectral unmixing. *In SPIE Europe Remote Sensing*, pages 74770I–74770I. International Society for Optics and Photonics. 113

NASCIMENTO, J. M. et DIAS, J. B. (2005). Vertex component analysis : A fast algorithm to unmix hyperspectral data. *IEEE Transactions on Geoscience and Remote Sensing*, 43(4): 898–910. 24

NASRABADI, N. (2014). Hyperspectral target detection : An overview of current and future challenges. *IEEE Signal Processing Magazine*, 31(1):34–44. 18

NOCEDAL, J. et WRIGHT, S. (1999). *Numerical optimization*, volume 2. Springer New York. 29, 32, 36, 39, 41, 75, 78

NVIDIA, C. (2008). Tesla c1060 specification board version 3. 84

NVIDIA, C. (2012). NVIDIA CUDA C programming guide version 4.2. 85

NVIDIA, C. (2014). NVIDIA cuSPARSE library version 6.0. 96, 112

ORTEGA, J. et RHEINBOLDT, W. (2000). *Iterative solution of nonlinear equations in several variables*, volume 30. Siam. 70

OWENS, J., HOUSTON, M., LUEBKE, D., GREEN, S., STONE, J. et PHILLIPS, J. (2008). GPU computing. *Proceedings of the IEEE*, 96(5):879–899. 83

PARENTE, M. et PLAZA, A. (2010). Survey of geometric and statistical unmixing algorithms for hyperspectral images. *In 2nd Workshop on Hyperspectral Image and Signal Processing : Evolution in Remote Sensing (WHISPERS)*, pages 1–4. IEEE. 29

PETERSEN, K. et PEDERSEN, M. (2006). The matrix cookbook. *Technical University of Denmark*. 74

PLAZA, A. et CHANG, C.-I. (2005). An improved N-FINDR algorithm in implementation. *In Proc. of SPIE Vol*, volume 5806, page 299. 24, 54

PLAZA, A., PLAZA, J., PAZ, A. et SANCHEZ, S. (2011). Parallel hyperspectral image and signal processing [applications corner]. *IEEE Signal Processing Magazine*, 28(3):119–126. 82

RICHARDS, J. (1999). *Remote sensing digital image analysis*, volume 3. Springer. 14

RUGH, W. (1996). *Linear system theory*, volume 2. prentice hall Upper Saddle River, NJ. 113

SANCHEZ, S., RAMALHO, R., SOUSA, L. et PLAZA, A. (2012). Real-time implementation of remotely sensed hyperspectral image unmixing on GPUs. *Journal of Real-Time Image Processing*, pages 1–15. 82

SANDERS, J., KANDROT, E. et JACOBONI, E. (2011). *CUDA par l'exemple*. Pearson. 85

SCHMIDT, F. (2007). *Classification de la surface de Mars par imagerie hyperspectrale OMEGA. Suivi spatio-temporel et étude des dépôts saisonniers de CO2 et H2O*. Thèse de doctorat, Université Joseph-Fourier-Grenoble I. 18

SCHMIDT, F. (2014). Echanges de volatils entre surface et atmosphère sur mars. *Paris-Sud XI*. 15

SCHMIDT, F., BOURGUIGNON, S., LE MOUËLIC, S., DOBIGEON, N., THEYS, C. et TRÉGUIER, E. (2011). Accuracy and performance of linear unmixing techniques for detecting minerals on omega/mars express. *In Hyperspectral Image and Signal Processing : Evolution in Remote Sensing (WHISPERS), 2011 3rd Workshop*, pages 1–4. 17, 100

SCHMIDT, F., LEGENDRE, M. et LE MOUËLIC, S. (2014). Minerals detection for hyperspectral images using adapted linear unmixing : Linmin. *Icarus*, 237:61–74. 99

SCHOTT, J. (2007). *Remote sensing*. Oxford University Press. 14

SETOAIN, J., PRIETO, M., TENLLADO, C. et TIRADO, F. (2008). GPU for parallel on-board hyperspectral image processing. *International Journal of High Performance Computing Applications*, 22(4):424–437. 82

SETTLE, J. et DRAKE, N. (1993). Linear mixing and the estimation of ground cover proportions. *International Journal of Remote Sensing*, 14(6):1159–1177. 29

SHI, Z. et SHEN, J. (2006). A new class of supermemory gradient methods. *Applied mathematics and computation*, 183(2):748–760. 113

STARCK, J. et MURTAGH, F. (2007). *Astronomical image and data analysis*. Springer Science & Business Media. 17

TARANTOLA, A. (2005). *Inverse problem theory and methods for model parameter estimation*. siam. 18

THOMPSON, C., HAHN, S. et OSKIN, M. (2002). Using modern graphics architectures for general-purpose computing : a framework and analysis. *In Proceedings of the 35th annual ACM/IEEE international symposium on Microarchitecture*, pages 306–317. IEEE Computer Society Press. 83

TILLING, A., O'LEARY, G., FERWERDA, J., JONES, S., FITZGERALD, G. et BELFORD, R. (2006). Remote sensing to detect nitrogen and water stress in wheat. *The Australian Society of Agronomy*. 17

Van der MEER, F., Van der WERFF, H., van RUITENBEEK, F., HECKER, C., BAKKER, W., NOOMEN, M., van der MEIJDE, M., CARRANZA, E., de SMETH, J. et WOLDAI, T. (2012). Multi-and hyperspectral geologic remote sensing : A review. *International Journal of Applied Earth Observation and Geoinformation*, 14(1):112–128. 17

VENKATASUBRAMANIAN, S. (2003). The graphics card as a stream computer. *In SIGMOD-DIMACS workshop on management and processing of data streams*, volume 101, page 102. 83

WINTER, M. (1999). N-FINDR : an algorithm for fast autonomous spectral end-member determination in hyperspectral data. *In SPIE's International Symposium on Optical Science, Engineering, and Instrumentation*, pages 266–275. International Society for Optics and Photonics. 24, 48

ZENG, G. (2001). Image reconstruction—a tutorial. *Computerized Medical Imaging and Graphics*, 25(2):97–103. 19

ZYMNIS, A., KIM, S., SKAF, J., PARENTE, M. et BOYD, S. (2007). Hyperspectral image unmixing via alternating projected subgradients. *In Conference Record of the Forty-First Asilomar Conference on Signals, Systems and Computers (ACSSC)*, pages 1164–1168. IEEE. 29

l'université nantes angers le mans
COMMUNAUTÉ D'UNIVERSITÉS ET ÉTABLISSEMENTS

Centrale Nantes

# Thèse de Doctorat

Maxime LEGENDRE

## Accélération algorithmique et matérielle des méthodes d'estimation de cartes d'abondances en imagerie hyperspectrale

## Algorithmic and hardware acceleration of abundance maps estimation methods in hyperspectral imaging

**Résumé**
L'imagerie hyperspectrale consiste en une mesure du spectre de réflectance en chacun des pixels d'une image. Cette technique de mesure est utilisée pour la télédétection aéroportée, en astrophysique ou encore en microscopie. Le traitement du grand volume de données que représente une image hyperspectrale nécessite à la fois des méthodes présentant un coût de calcul maîtrisé et un besoin mémoire raisonnable. Le traitement proposé dans cette thèse a pour objectif l'estimation de cartes d'abondances (proportions de plusieurs constituants dans chaque pixel de l'image) par minimisation d'un critère de type moindres carrés sous des contraintes de positivité et de somme à un, additionné d'un terme de pénalisation pour assurer une régularité spatiale des cartes. Les travaux réalisés ont pour objectif la réduction du temps de calcul d'une méthode d'optimisation de type points-intérieurs. Des modifications algorithmiques basées sur la notion d'approximation majorante séparable sont proposées. Il en résulte une méthode à la fois plus rapide et plus adaptée aux outils de calcul parallèle. Une implémentation sur processeurs de cartes graphiques (GPU) est réalisée et appliquée à grande échelle pour traiter un grand nombre d'images hyperspectrales issues de la mission d'exploration spatiale Mars Express. La méthode développée est également utilisée dans un projet de suivi de la végétation sur la côte atlantique française.

**Mots clés**
Imagerie hyperspectrale, optimisation sous contraintes, points-intérieurs, calcul parallèle, processeur de carte graphique (GPU).

**Abstract**
Hyperspectral imaging consists in collecting the reflectance spectrum for each pixel of an image. This measurement technique is used in airborne remote sensing, astrophysics, or microscopy. Processing the large data volume of a hyperspectral image requires a method with both restrained computational cost and limited memory usage. The method proposed in this thesis aims at estimating the abundance maps (component's proportions in each image pixel) by constrained least squares criterion minimization with the addition of a penalization term to ensure the maps spatial regularity. The work done intends to reduce the computing time of an interior point optimization method. Algorithmic modifications based on separable majorization are proposed. It results in a method both faster and more adapted to parallel computing tools. An implementation on Graphics Processing Units (GPU) is achieved and applied in a large scale experiment where a high number of hyperspectral images from Mars Express exploration mission are processed. The developed method is also used in a vegetation monitoring project on the french atlantic coast.

**Key Words**
Hyperspectral imaging, constrained optimization, interior-points, parallel computing, Graphics Processing Unit (GPU).