Analyse d'image hyperspectrale

Adrien Faivre

UBFC
UNIVERSITÉ
BOURGOGNE FRANCHE-COMTÉ

LmB

Thèse de Doctorat

présentée par

Adrien Faivre

pour obtenir le grade de
Docteur en Mathématiques et Applications de l'Université Bourgogne Franche-Comté

Analyse d'images hyperspectrales

Thèse soutenue le 14 décembre 2017, devant le jury composé de :

Jean Berney	Examinateur
Clément Dombry	Directeur de thèse
Florence Forbes	Rapporteur
Nicolas Gillis	Rapporteur
Christophe Guyeux	Examinateur
Cyprien Hufschmitt	Examinateur

ECOLE DOCTORALE
CARNOT - PASTEUR

Digital Surf
Surface Intelligence

cnrs

Table des matières

1 Contexte

Ce mémoire présente les travaux de thèses effectués dans le cadre d'une convention Cifre établie entre l'Association Nationale de la Recherche et de la Technologie, l'entreprise Digital Surf et le Laboratoire de Mathématiques de Besançon.

1.1 Problématique

Digital Surf est une entreprise bisontine développant des logiciels d'analyse d'image et d'analyse des surfaces pour microscopes et profilomètres depuis 1989. L'entreprise travaille en partenariat avec de nombreux fabricants d'instruments d'imagerie ou de mesure de surface. Le logiciel Mountains produit par l'entreprise équipe aujourd'hui les instruments d'une majorité des fabricants de profilomètre et de microscope. Dans sa déclinaison MountainsMap Hyperspectral, l'entreprise commercialise des solutions d'analyse d'images hyperspectrales pour la micro-spectroscopie. Ces images sont souvent présentées comme des cubes de données, car, contrairement à une image en noir et blanc où l'on ne dispose que d'une seule valeur par pixel, chaque pixel d'une image hyper-spectrale contient jusqu'à plusieurs centaines de valeurs permettant de décrire le spectre mesuré en ce point.

D'abord destinée à la télé-détection, l'hyper-spectral a progressivement gagné d'autres secteurs, comme la biochimie, la recherche pharmaceutique, ou la surveillance de la qualité des aliments pour n'en citer que quelques uns. Le terme d'analyse hyper-spectrale recoupe plusieurs méthodes très variées. Celles-ci s'étendent du dé-bruitage à la classification des spectres. Plus généralement, on peut dire que le terme regroupe un vaste ensemble de méthodes issues des statistiques multi-variées. Ces méthodes ont pour point commun leur vocation d'expliquer le cube, c'est à dire à le décomposer en plusieurs facteurs, en exprimant par exemple chaque spectre du cube comme le mélange d'un certain nombre de spectres de base.

Le logiciel propose habituellement des méthodes d'analyse suffisamment rapides pour être exécutées en quasi-temps réel sur un ordinateur standard. Les cubes hyperspectraux regroupant un nombre souvent important de données, il faudra donc veiller à ce que les techniques développées soient suffisamment rapides pour être calculées en un temps raisonnable.

Les hyper-cubes sont plus qu'une simple collection de spectres. Ils incluent aussi

les informations concernant la répartition spatiale de ces derniers. Ces données sont peu souvent prises en compte dans la littérature hyper-spectrale. Un des objectifs de la thèse est de mettre en valeur ces informations, en proposant des techniques de *clustering* et de décomposition sensibles à la répartition spatiale des spectres dans le cube.

1.2 Plan

Ce travail de thèse est donc un travail prospectif, essayant d'explorer au mieux le large éventail des diverses branches de l'analyse hyper-spectrale. Cela pose le risque de réduire ce mémoire à une simple énumération des différentes techniques englobées par le terme. Nous nous sommes cependant efforcés de garder une certaine cohésion, chaque chapitre renvoyant souvent à un autre, ou faisant allusion aux mathématiques abordées ailleurs dans le manuscrit. La cohérence de ce manuscrit provient d'ailleurs en majorité de l'apparition des mêmes outils à travers les différents chapitres. Les notions de programmation linéaire ou semi-définie, de décomposition d'une matrice ou d'un tenseur en valeurs propres, ou même de théorie des graphes réapparaissent régulièrement tout au long de ce mémoire.

Ce dernier se veut également accessible au lecteur moins versé dans les mathématiques. Il centralise en effet les explications de beaucoup des fonctionnalités d'analyse hyper-spectrale développées pour Mountains.

Le plan est organisé selon trois parties. La première partie est consacrée au *clustering*. Nous décrivons au premier chapitre le contexte de ce travail de thèse. Nous commençons par y exposer certaines des conclusions d'un stage effectué au préalable, concernant la classification des données par l'algorithme des *k-means*. Nous discutons ensuite au deuxième chapitre de la complexité algorithmique liée à ce problème, et introduisons différents outils nécessaires à la compréhension d'un des deux articles auquel cette thèse a donné lieu. Cet article présente une méthode de classification de données plus théorique, présentant certaines garanties d'efficacité, et constitue le troisième chapitre de ce mémoire. Il a été coécrit avec Stéphane Chrétien, et Clément Dombry, en langue anglaise. La seconde partie est consacrée au dé-mélange hyper-spectral. Le quatrième chapitre s'intéresse en effet à la factorisation en matrices positives, méthode clé pour dé-mélanger un cube. Il décrit les difficultés théoriques liées à cette décomposition, et comment on peut essayer d'y remédier. On y introduit ainsi certaines des notions mises en œuvre dans la seconde publication engendrée par ce travail de thèse. Celle-ci présente une méthode permettant de factoriser un cube mettant en valeur les informations de répartition spatiale des spectres. Elle a été coécrite avec Clément Dombry en anglais, et constitue le cinquième chapitre de cette thèse. La troisième et dernière partie regroupe les autres techniques abordées durant cette thèse. L'étude des méthodes de réduction non-linéaires de la dimension, ainsi que l'analyse en composantes indépendantes ou la régularisation spatiale du *clustering* y sont regroupées dans un sixième chapitre. Un dernier chapitre conclut ce mémoire en donnant un point de vue plus pratique de ce qu'a été ce travail de thèse.

1.3 Deux solutions déjà envisagées

Pour commencer ce chapitre d'introduction, nous décrivons rapidement les techniques de réduction de la dimension et de *clustering* développées préalablement à cette thèse. Certaines étaient en place avant même mon arrivée à Digital Surf, d'autres ont été mises en œuvre durant un stage effectué en 2014.

1.3.1 *Clustering* de données

La classification des hypercubes assigne chacun des pixels à une classe. C'est un problème assez courant en analyse hyper-spectrale, où l'exemple classique consiste à différencier sur une image aérienne les pixels représentant un certain type d'élément, par exemple une route, de ceux représentant un autre, comme des végétaux, les toits des maison, ou des voitures. Dans la majorité des cas auxquels le logiciel Mountains sera confronté, il ne s'agira pas d'images aériennes, mais d'images issues de microspectroscopie. Si les objets d'étude changent, les techniques restent similaires.

Le but est de minimiser la similarité entre classes, et/ou de maximiser la similarité à l'intérieur de chaque classe. Plusieurs critères quantifiant la qualité d'une classification existent, et mènent à des résultats différents. Un critère classique car naturel est celui des *k-means*. D'autres, très naturels eux aussi, sont issus de la théorie des graphes. Les *graph-cuts*, comme *min/max-cut*, ou ses versions régularisées, cherchent à séparer des communautés dans un graphe en coupant le moins d'arêtes possible.

L'algorithme de Loyd

Un exemple simple de technique de *clustering* est l'algorithme *k-means*.

Cet algorithme de classification des données permet, étant donnés $k \in \mathbb{N}$ centres initiaux, de partitionner l'ensemble $\Omega = \{x_1, \cdots, x_m\}$ des données en k *clusters*, i.e. en k sous ensembles disjoints de Ω. Cet algorithme essaye de minimiser la distance euclidienne entre chaque élément de Ω et le centre qui lui est assigné. Plus précisément, *k-means* s'efforce de minimiser le critère suivant, nommé critère SSE,

$$\min_{\mathcal{C}\in\mathcal{P}(\Omega)} \sum_{i=1}^{k} \sum_{x_j\in\mathcal{C}_i} \|x_j - c_i\|_2^2, \tag{1.1}$$

où $\mathcal{C} = (\mathcal{C}_1, \cdots, \mathcal{C}_k)$ sont k *clusters*, où c_i est le centre du *cluster* $\mathcal{C}_i$, et où $\mathcal{P}(\Omega)$ désigne l'ensemble des partitions possibles de Ω. La procédure est résumée dans l'algorithme 1.

Astuce de l'inégalité triangulaire

Plutôt que de recalculer à chaque itération les centres, on peut garder en mémoire la somme s_j de chacun des éléments attachés au centre j, ainsi que leur nombre n_j. On pourra ainsi recalculer chaque centre j en calculant

$$c_j = \frac{s_j}{n_j}. \tag{1.2}$$

Algorithme 1 *k-means*

Entrée : m éléments à classifier $\{x_1, \cdots, x_m\} \in (\mathbb{R}^n)^m$, et k centres initiaux $\{c_1, \cdots, c_k\} \in (\mathbb{R}^n)^k$ {Les k centres initiaux peuvent être choisis de diverses façons. On peut les sélectionner de façon complètement aléatoire, ou bien avec *k-means++* (2)}

Sortie : Minimise le critère SSE de la partition formée des k *clusters* définis comme ensembles des éléments les plus proches des k centres **c**

1: **while do**
2: Assigner chaque point x_i à son centre le plus proche c_j.
3: Recalculer chaque centre comme moyenne des x_i qui lui sont le plus proche.
4: **end while**

Il faudra cependant veiller à mettre à jour s et q chaque fois qu'un élément changera d'assignation.

Si l'on a choisi de travailler avec la distance euclidienne (ou tout du moins avec une 'vraie' distance respectant l'inégalité triangulaire), il est possible d'accélérer grandement l'algorithme. Si on sait que la distance entre un élément x_i et son centre assigné c_j est inférieure à un réel positif u_i, et que l'on connaît la distance minimale d_j entre c_j et chaque autre centre $\{c_1 \cdots c_{j-1}, c_{j+1}, \cdots c_k\}$, alors on peut affirmer, grâce à l'inégalité triangulaire, que si

$$u_i \leq \frac{d_j}{2}, \tag{1.3}$$

alors c_j est plus proche de x_i que tous les autres centres. Si l'on connaît également une borne inférieure l_i sur la distance entre chaque élément x_i et son centre le plus proche différent de son centre assigné c_j, on peut affirmer que si

$$u_i \leq l_i, \tag{1.4}$$

alors c_j est plus proche de x_i que tous les autres centres. Ces idées sont développées dans l'article Hamerly (2010).

Initialisation par *k-means++*

Le choix des centres initiaux est capital pour *k-means*. Il influe grandement sur les résultats. *k-means++* est un algorithme d'initialisation. Il permet d'éviter de choisir deux centres initiaux trop proches. Son principe est le suivant.

1. On choisit les centres initiaux de façon aléatoire, mais avec une loi de probabilité non-uniforme.

2. La probabilité de choisir un point comme centre initial est proportionnelle au carré de la distance de ce point aux centres déjà choisis.

Pour plus de détails sur les garanties de convergences qu'offre *k-means++* le lecteur pourra se référer à Arthur et Vassilvitskii (2007). Son implémentation est résumée dans l'algorithme 2. Il en existe une version parallélisable Bahmani *et al.* (2012), qui choisit $k' > k$ centres, puis sélectionne les meilleurs k centres parmi ces k' centres.

Son principal défaut est que, comme *k-means++*, il nécessite de trouver le centre le plus proche de chaque élément à chaque itération, ce qui prend beaucoup plus de temps quand le le nombre de centres augmente, à fortiori en présence de très nombreux éléments comme c'est généralement le cas pour les hypercubes.

Algorithme 2 *k-means++*, voir Arthur et Vassilvitskii (2007)

Entrée : $(x_i)_{i\in\{1,\cdots,m\}}$ les m éléments à partitionner {Nul besoin de centres initiaux.}

Sortie : $(c_i)_{i\in\{1,\cdots,k\}}$ les k centres initiaux pour *k-means* (1) {Renvoie k centres permettant d'initialiser *k-means*.}

$c_1 = x_1$
for $j \in \{1, \cdots, k-1\}$ **do**
  $s_1 = 0$
  for $j \in \{1, \cdots, m\}$ **do**
    $s_j = s_{j-1} + \min\left(\|x_j - c_1\|_2^2, \cdots, \|x_j - c_j\|_2^2\right)$
  end for
  Choisir aléatoirement $r \in [s_1, s_m]$ selon la loi uniforme
  Trouver $l \in \mathbb{N}$ tel que $r \in [s_l, s_{l+1}]$
  $c_j = x_l$
end for

Choix d'une norme

La similarité entre deux spectres est souvent définie différemment de la norme euclidienne. Une des autres mesures populaires est le *Spectral Angle Mapper* ou SAM, utilisé par exemple dans Park *et al.* (2007). Pour chaque *cluster* $\mathcal{C} = \{x_0, x_1, \cdots, x_l\}$, le minimum de la fonction

$$c \to \sum_{x_j \in \mathcal{C}} \|x_j - c\|_2^2 \tag{1.5}$$

est atteint quand c est l'isobarycentre de $\mathcal{C}$. Ce n'est pas forcément le cas avec une autre norme. C'est pourquoi *k-means* converge généralement bien avec la distance euclidienne classique, ce qui est très incertain quand on choisit d'autres distances.

Évaporation de *clusters*

Il peut aussi arriver que pendant le calcul des *clusters*, certains se retrouvent vidés de leurs éléments. Le résultat obtenu est alors très différent du résultat souhaité. De façon générale, la résultat obtenu par *k-means* peut être arbitrairement mauvais. La qualité du résultat dépend en effet fortement de l'initialisation choisie par l'algorithme.

1.3.2 Réduction de la dimension

On expose dans cette section certains outils de réduction de la dimension. Cette étape est souvent primordiale car les hypercubes ont tendance a être très volumineux. Il est donc souvent intéressant de les réduire ou de les compresser avant de

commencer des calculs. D'un certain point de vue, beaucoup des techniques présentées dans ce mémoire peuvent être perçues comme une réduction de la dimension. Certaines factorisations matricielles, comme la factorisation en matrices positives peuvent dans certains cas être considérées comme des techniques de réduction de dimension. On se concentrera dans cette section sur les outils classiques, et on laissera l'exposition des autres techniques aux chapitres suivants.

Analyse en composantes principales et décomposition en valeurs singulières

L'analyse en composantes principales (PCA) est un des outil de base en analyse hyper-spectrale. Projeter les données sur les directions orthogonales de variance maximale permet de réduire leur dimension, et ce de façon optimale dans un certain sens. La décomposition en valeurs singulières (SVD) tronquée y joue un rôle central, car elle permet de s'affranchir du calcul de la matrice de covariance, ainsi que de sa diagonalisation, permettant plus de stabilité numérique. Un des théorèmes fondamentaux de l'algèbre linéaire garantit l'existence de la SVD pour toute matrice $M \in \mathbb{R}^{m\times n}$ de rang k. Cette dernière s'écrit

$$M = USV^T, \tag{1.6}$$

avec $U \in \mathbb{R}^{m\times k}$ orthogonale, $S \in \mathbb{R}^{k\times k}$ diagonale, et $V \in \mathbb{R}^{n\times k}$ orthogonale. Une des façons les plus classiques de calculer une SVD est de recourir aux rotations de Givens. Une telle rotation s'écrit, pour $1 \leq i \leq n$, $1 \leq j \leq m, j \neq i$, et $\theta \in [0, 2\pi[$:

$$G(i,j,\theta) = \begin{pmatrix} 1 & \cdots & 0 & \cdots & 0 & \cdots & 0 \\ \vdots & \ddots & \vdots & & \vdots & & \vdots \\ 0 & \cdots & c & \cdots & -s & \cdots & 0 \\ \vdots & & \vdots & \ddots & \vdots & & \vdots \\ 0 & \cdots & s & \cdots & c & \cdots & 0 \\ \vdots & & \vdots & & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & \cdots & 0 & \cdots & 1 \end{pmatrix}, \tag{1.7}$$

avec $c = \cos(\theta)$, et $s = \sin(\theta)$. En multipliant à gauche et à droite la matrice de départ M par une suite de ces matrices de Givens, en prenant soin chaque fois de choisir θ de façon optimale, on arrive à diagonaliser M. Si appliquer une rotation de Givens à une matrice est très rapide, ce genre de méthode nécessite en revanche généralement un nombre important d'itérations avant d'obtenir une matrice suffisamment diagonale. La méthode est donc relativement lente, et difficilement applicable à un hypercube.

Algorithme aléatoire pour le calcul de la SVD

Il est possible d'accélérer grandement le calcul de la factorisation SVD en utilisant la méthode randomisée détaillée dans Halko *et al.* (2011). L'idée de cette méthode est de trouver une base de l'espace image de M, c'est à dire de trouver une matrice Q orthogonale (c'est à dire respectant $Q^TQ = I_d$), telle que $M = QQ^TM$, avec Q

ayant un nombre minimal de colonnes. On pose ensuite $\tilde{M} = QM$, et on calcule la SVD de $\tilde{M} = \tilde{U}SV^T$. Cette SVD est plus simple à calculer, la matrice $\tilde{M}$ étant plus petite que M. Pour obtenir la SVD de M il ne manque alors plus que le facteur U, que l'on obtient comme $U = Q^T\tilde{U}$.

Il reste à déterminer une matrice Q telle que $M = QQ^TM$. Supposons que le rang de M soit exactement $k > 0$. Étant donné une matrice M, on commence par générer aléatoirement k vecteurs $\{\omega_1, \cdots, \omega_k\}$ selon la loi normale centrée réduite. Comme ces vecteurs on été tirés aléatoirement, ils ont toutes les chances d'être linéairement indépendants, et de ne pas appartenir au noyau de M. On forme ensuite l'ensemble $\{y_1, \cdots, y_k\}$, où $y_i = M\omega_i$. Cet ensemble forme donc très probablement une base de l'espace image de M. Pour produire une base orthonormale de l'espace image de M, il n'y a plus qu'à ortho-normaliser l'ensemble des vecteurs y_i. Supposons maintenant que la matrice M soit légèrement perturbée, par exemple par du bruit dans la mesure, ou par des problèmes de précision numérique, ce qu'on écrit $\tilde{M} = M + E$, où M est la matrice de rang k non bruitée, et E est la matrice représentant les perturbations. L'introduction de la matrice E change la direction des k vecteurs $\{y_1, \cdots, y_k\}$, et peut donc empêcher que ces vecteurs engendrent l'intégralité de l'image de M. Pour éviter cela, on enrichit la base $\{y_1, \cdots, y_k\}$ de l'espace image de M de p nouveaux vecteurs. L'ensemble $\{y_1, \cdots, y_k, y_{k+1}, \cdots, y_{k+p}\}$ a nettement plus de chances de couvrir l'ensemble de l'espace image de M, rendant ainsi l'algorithme plus robuste. Pour retrouver une base orthonormale de M, on pratique alors une décomposition QR à pivot sur $\{y_1, \cdots, y_k, y_{k+1}, \cdots, y_{k+p}\}$.

Une fois la base Q de l'image de M calculée, on applique alors une SVD sur la matrice réduite, et on obtient ainsi les k premières valeurs singulières et vecteurs singuliers de $\tilde{M}$. Le détail de l'algorithme est présenté dans l'algorithme 3. Dans l'algorithme 3, le paramètre p, nombre de vecteurs aléatoires à générer en plus des k premiers, est remplacé par deux paramètres : i et l. Dans Halko *et al.* (2011), les auteurs conseillent une valeur par défaut pour chacun de ces paramètres.

- $i \leq 2$,
- $l = k + 2$.

On génère en tout $(i + 1)l$ vecteurs aléatoires.

Robustesse

L'analyse en composante principale classique cherche à résoudre

$$\min \|M - L\|_2 \text{ tel que } \operatorname{rank}(L) \leq k.$$

Bien que très employée, sa grande sensibilité aux valeurs aberrantes souvent présentes dans les données la rendent peu fiable. En effet, une seule valeur grossièrement erronée dans M peut rendre l'estimation de L très différente de sa valeur réelle. Dans Candès *et al.* (2011), il est suggéré d'adapter la définition de la PCA, en résolvant plutôt :

$$\min \|L\|_2 + \lambda\|S\|_1 \text{ tel que } L + S = M.$$

Algorithme 3 L'algorithme de décomposition SVD rapide, voir Halko *et al.* (2011)

Entrée : $M \in \mathbb{R}^{m\times n}$, $i, k, l \in \mathbb{N}$, $k < m$, $k < n$, $k \leq l$, $(i+2)k \leq n$. {M est la matrice dont on veut calculer la décomposition SVD, k est le nombre de valeurs singulières que l'on veut retrouver, i et l sont deux paramètres permettant d'augmenter la stabilité numérique du présent algorithme. Valeurs par défaut conseillées : $i \leq 2$, et $l = k + 2$.}

Sortie : $U \in \mathbb{R}^{m\times n}, S \in \mathbb{R}^{n\times n}, V \in \mathbb{R}^{n\times n}$, U, V orthogonales, S diagonale, telles que :

$$\left\|M - USV^T\right\|_F \leq \sqrt{(Ckn)^{1/(2i+1)} + \min(1, C/n)}\sigma_{k+1}, \tag{1.8}$$

où C est une constante. {L'inégalité ci-dessus est vraie dans l'immense majorité des cas pour $C = 10$, d'après l'article de N. Halko Halko *et al.* (2011). Nous avons constaté que c'était encore globalement le cas pour $C = 1$.}

1: On génère aléatoirement $G \in \mathbb{R}^{m\times(i+1)l}$.

2: On calcule $H = \left(MG \middle| MM^TMG \middle| \cdots (MM^T)^iMG\right) \in \mathbb{R}^{m\times(i+1)l}$.

3: On calcule $Q \in \mathbb{R}^{m\times(i+1)l}$, $R \in \mathbb{R}^{(i+1)l\times(i+1)l}$, $\Pi \in \mathbb{R}^{(i+1)l\times(i+1)l}$ telles que

$$H\Pi = QR, \tag{1.9}$$

avec Π une permutation, Q orthogonale, R triangulaire supérieure.

4: On calcule $T = M^TQ \in \mathbb{R}^{n\times(i+1)l}$.

5: On calcule $\tilde{V} \in \mathbb{R}^{n\times(i+1)l}, \tilde{S} \in \mathbb{R}^{(i+1)l\times(i+1)l}, W \in \mathbb{R}^{(i+1)l\times(i+1)l}$, telles que

$$T = \tilde{V}\tilde{S}W^T, \tag{1.10}$$

avec $\tilde{V}$ orthogonale, $\tilde{S}$ diagonale, W orthogonale.

6: On génère $\tilde{U} = QW \in \mathbb{R}^{m\times(i+1)l}$

7: On obtient $U \in \mathbb{R}^{m\times k}, S \in \mathbb{R}^{k\times k}, V \in \mathbb{R}^{n\times k}$ comme blocs supérieurs gauches de $\tilde{U}, \tilde{S}, \tilde{V}$

Contrairement à l'hypothèse d'un bruit faible N dans la PCA classique, les entrées de S peuvent être aussi grande que l'on veut, la seule hypothèse que l'on fait sur elles portent sur leur support, que l'on suppose sparse. Ce type de PCA, appelé PCA robuste ou RPCA, fait l'objet du paragraphe 6.1.

1.4 Dé-mélange hyper-spectral

Analyse hyper-spectrale est un terme regroupant plusieurs techniques. Nous avons déjà discuté dans ce chapitre de classification et de réduction de la dimension. Nous introduisons maintenant le dé-mélange hyper-spectral, l'une des notions centrales de ce travail de thèse.

1.4.1 Centres des classes et cartes de corrélations

A l'issu d'une classification d'un hyper-cube, chaque spectre appartient à une unique classe. Si l'on souhaite affiner ce résultat et savoir dans quelle mesure un spectre appartient à une classe et non à une autre, on peut chercher à calculer une distance entre ce spectre et un représentant de chacune des classes distinguées lors de la classification. L'algorithme de *k-means* permet par exemple de regrouper les spectres d'un cube en plusieurs classes homogènes définies par leur centre. On peut donc imaginer sélectionner le centre de chaque classe comme spectre de base, et calculer la distance euclidienne entre chaque spectre et ce spectre de base.

Ce n'est pourtant pas ce que l'on fait habituellement. Un des défauts de la méthode est que les centres des classes peuvent être relativement similaires les uns aux autres. C'est le cas sur la figure 1.1 par exemple, où les spectres extraits de cette façon sont très semblables. Il existe d'autres techniques, basées sur un modèle de mélange inspiré de considérations physiques, permettant de faire beaucoup mieux. Nous donnons à titre de comparaison figure 1.2 les spectres de bases calculés par l'algorithme 5, que nous décrirons chapitre 4.

1.4.2 L'hypothèse de Craig

Dans Craig (1994), l'auteur observe que les projections de nuages de spectres en deux dimensions constituent généralement des triangles. Il fait alors l'hypothèse que les spectres contenus dans un hypercube forment un simplexe, chaque spectre étant constitué d'un mélange de spectres de bases, correspondant aux sommets de ce simplexe. Une fois ces spectres de bases déterminés, on cherche à calculer les proportions constituant chacun des spectres observés.

Une myriade d'algorithmes existe pour tenter de résoudre le problème. On peut retenir globalement trois idées de base.

- Certains algorithmes cherchent à sélectionner k spectres parmi les spectres observés, formant un simplexe de volume maximal. La plupart du temps, on commence avec k spectres choisis aléatoirement, et on essaye d'augmenter le volume en remplaçant itérativement chaque spectre par un autre. C'est l'approche développée dans NFINDR, SIVM, et d'autres... Le problème correspondant, à savoir celui de déterminer le plus grand simplexe inscrit dans un

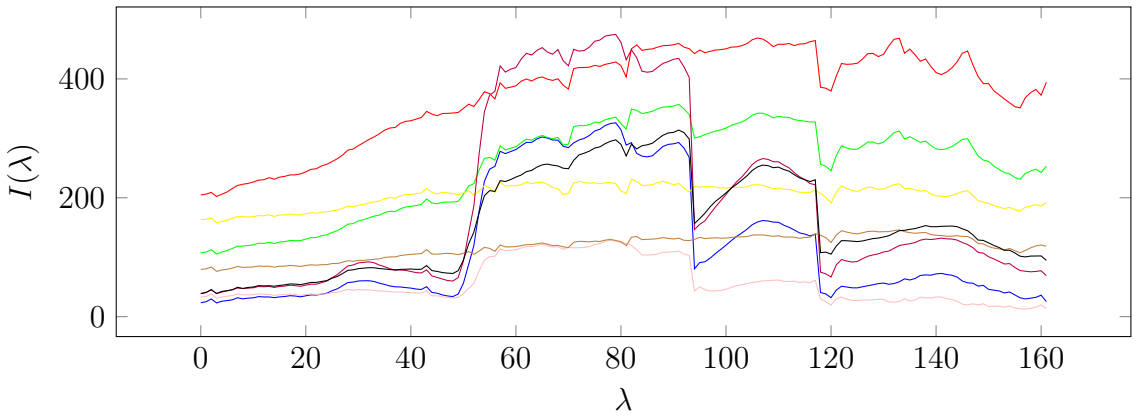


FIGURE 1.1 – Spectres de base par *k-means*

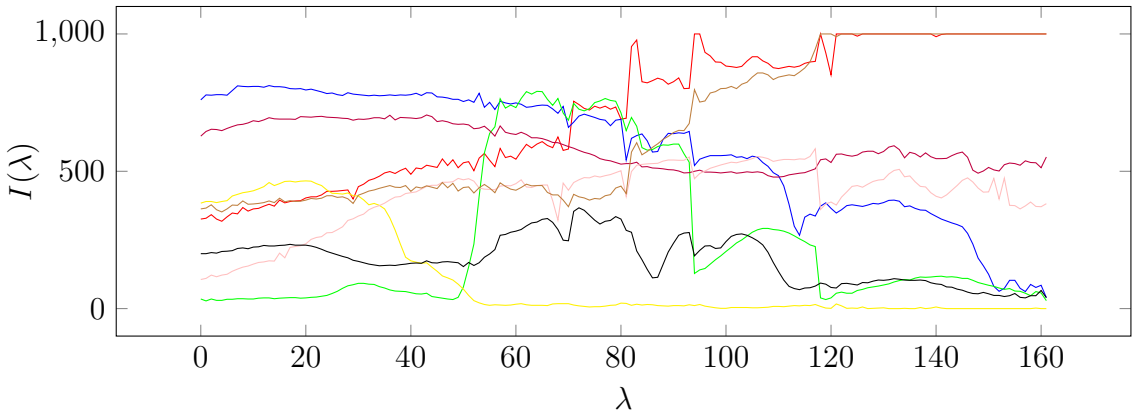


FIGURE 1.2 – Spectres de base par SPA

ensemble de points est particulièrement difficile. Le problème fait en effet partie d'une classe de problèmes complexes, nommée NP-difficile Packer (2002). Nous donnons une définition précise de cette classe plus tard dans ce mémoire, section 2.1.2.

- L'approche opposée consiste à chercher un simplexe de volume minimal contenant tous les spectres du cube. C'est la stratégie retenue par SISAL par exemple.

- Une troisième approche consiste à projeter l'ensemble des spectres du cube dans l'espace engendré par les endmembers déjà déterminés, et de retenir comme prochain spectre de base celui dont la projection est la plus extrême dans un certain sens. On y retrouve des algorithmes comme PPI, VCA, ou SPA.

1.4.3 Le mélange linéaire

Le modèle du simplexe correspond à un mélange linéaire. On considère que l'ensemble des spectres du cube observé $m_1, \cdots m_n \in \mathbb{R}_+^m$ peuvent se décomposer comme k spectres de bases $w_1, \cdots, w_k \in \mathbb{R}_+^m$ dans les proportions décrites par $h_1, \cdots, h_n \in \mathbb{R}_+^k$. En considérant que chaque spectre de base a une influence linéaire sur les spectres observés, on écrit matriciellement ce mélange comme

$$M = WH.$$

Comme ce produit est en pratique soumis à différents bruits, on préférera écrire le problème comme

$$\min_{W \in \mathbb{R}_+^{m \times k}, H \in \mathbb{R}_+^{k \times n}} \|M - WH\|_F .$$

Ce problème correspond à la factorisation en matrices non-négatives ou NMF. Les différentes approches envisagées dans le paragraphe précédent ne s'attaquent pas au problème de front : plutôt que de chercher à minimiser par rapport à W et H simultanément, elles cherchent dans un premier temps à déterminer W seul, sans chercher d'ailleurs systématiquement une matrice positive. Seulement ensuite essayent-elles de calculer H. On peut par ailleurs contraindre chaque colonne de H à appartenir au k-simplexe, de façon à mettre en avant l'idée de mélange.

1.4.4 Mélanges non-linéaires

Plusieurs articles, dont Keshava et Mustard (2002) ou Bioucas-Dias *et al.* (2012) par exemple, expliquent que ce type de mélange linéaire est plausible physiquement. Le but de cette thèse n'est pas de confirmer ou d'infirmer l'usage de tels modèles. On peut toutefois noter que dans la littérature consacrée à la spectroscopie Raman par exemple, les choses semblent en fait être un peu plus compliquées. Il existe apparemment plusieurs phénomènes agissant potentiellement sur le spectre d'un mélange de façon non-linéaire. Certains interactions entre les matériaux observées semblent en effet susceptibles de changer la forme d'un spectre (Hartmann *et al.*,

2008; Frommhold, 2006), en élargissant certains des 'pics' constituant le spectre, ou même en faisant apparaître de nouveaux.

Dans la littérature, il existe plusieurs articles consacrés à des modèles de mélange non-linéaire. On ne modélise plus le mélange comme un produit matriciel, mais comme une fonction non-linéaire de W et de H. Le choix de cette fonction non-linéaire dépend des domaines d'application et des auteurs des articles, et est généralement motivé par une modélisation du mélange. Comme cette thèse ne vise pas un type précis de spectroscopie, on préfère rester générique, et s'intéresser aux modèles dits post-non-linéaires, qui permettent de séparer la phase de dé-mélange de la phase de linéarisation. Les différentes méthodes de projection d'une variété non-linéaire vers un espace vectoriel sont explorées dans la section 6.2.

1.5 Autres problèmes d'analyse hyper-spectrale

D'autres méthodes encore figurent parmi les techniques de l'analyse hyper-spectrale. Nous en donnons quelques unes que nous n'avons pas abordées jusqu'ici.

Filtres

Les spectres comme les cubes que l'on observe sont généralement bruités. On évoque brièvement dans ce paragraphe quelques modèles de dé-bruitage. Les différents filtrages suivant s'appliquent à un spectre. Pour filtrer un cube, on peut en filtrer chaque spectre successivement.

- Le filtre moyen est un filtre simple à mettre en œuvre : on se contente de remplacer chaque élément par la moyenne des éléments de son voisinage.
- Le filtre médian, lui, remplace chaque élément par la médiane des éléments dans son voisinage.
- Le filtre Gaussien est similaire au filtre moyen, à la différence que l'on considère une moyenne pondérée avec des poids donnés par le noyau gaussien.
- Le filtre de Wiener permet, à partir de la variance du bruit et de la variance du signal, de dé-bruiter un signal de façon optimale. En pratique, il faudra tout de même estimer les différentes variances.
- Le filtre Savitsky-Golay ajuste un polynôme à chaque voisinage de points, puis remplace la valeur en ce point par la valeur du polynôme ajusté. C'est une technique de filtrage proche de la régression polynomiale locale en statistiques.

Il est aussi possible de filtrer le cube dans son ensemble. La plupart des algorithmes commencent par déplier le cube. Il est alors possible de filtrer la matrice obtenue par les méthodes suivantes.

- Le filtrage par SVD tronquée. On calcule la SVD de la matrice à filtrer, puis on écarte les valeurs singulières les plus faibles. Le résultat correspond à la matrice d'un certain rang la plus proche au sens des moindres carrés de la

matrice de départ. On peut aussi interpréter cette matrice comme issue d'une PCA, où l'on aurait supprimé le signal provenant des sous-espaces d'inertie trop faible.

- Une variation de la SVD tronquée est la minimum noise fraction ou MNF, très utilisée en analyse hyper-spectrale, décrite dans Green *et al.* (1988).

Il est aussi possible de voir le cube dans son ensemble. Les méthodes fonctionnant par voisinage restent valables, en veillant à sélectionner des voisinages en 3 dimensions. Il est aussi possible d'adapter le filtre par SVD tronquée, en utilisant une SVD d'ordre supérieur, décrite section 6.4.2.

Reconstruction par rapport à un dictionnaire

Une autre problématique assez courante en analyse hyper-spectrale consiste à décomposer un cube en fonction d'une librairie spectrale. Cette problématique est étudiée par exemple dans Iordache *et al.* (2012, 2010). Le problème revient alors à identifier si certains éléments de cette librairie sont présents dans le cube, et dans quelle quantité. On pourrait naïvement modéliser le problème comme un simple problème de moindres carrés positifs. Mais comme le nombre de spectres disponibles dans une librairie peut être comparable, ou même supérieur au nombre de spectres contenus dans le cube, il n'y aurait pas unicité de la solution. De plus ces solutions utiliseraient probablement tous les spectres de la librairie pour la reconstruction. On préfère donc habituellement un modèle économe en spectres utilisés de la librairie, de façon à rendre le résultat plus interprétable. Le modèle retenu est un modèle semblable à ce que l'on peut trouver en acquisition comprimée, à savoir

$$\min_{H \in \mathbb{R}_+^{k \times n}} \|M - WH\|_F^2 + \lambda \|H\|_1 ,$$

où $W \in \mathbb{R}_+^{m \times k}$ est la librairie spectrale contenant k spectres, H est la matrice décrivant l'utilisation des spectres de la librairie dans la reconstruction, et λ est le paramètre pénalisant l'utilisation d'un nombre trop important de spectres dans la reconstruction. On ne rentrera pas dans les détails dans ce paragraphe – rappelons juste que la pénalisation de la norme 1 est l'une des façons de privilégier les solutions sparses. Plus de détails sont fournis dans le chapitre 4.

Classification régularisée

Une fois le dé-mélange effectué, avec ou sans librairie, peut-on l'utiliser pour proposer une classification plus intéressante que *k-means* ? Il est en effet possible d'utiliser les proportions du mélange pour promouvoir une sorte de régularité spatiale. L'idée est de ne pas retenir comme classe d'un pixel celle du spectre de base dont il est constitué en majorité de façon systématique. Pour obtenir une continuité spatiale, il est utile d'examiner la constitution des voisins. Ce genre de considération amène à étudier un modèle courant en physique, le modèle d'Ising. Nous introduisons cette méthode plus en détail chapitre 6.

2

Problèmes de *clustering*

On introduit sommairement dans ce chapitre l'idée de complexité d'un problème, avant d'étudier certaines relaxations de problèmes difficiles. Le but n'étant pas un exposé précis des différentes notions de complexité, on ne donnera ici les définitions et théorèmes que de façon approximative, en essayant de donner suffisamment d'éléments pour appréhender les différentes relaxations. Pour une introduction rigoureuse à la complexité, au travers notamment des machines de Turing, un exposé précis et exhaustif est proposé dans Arora et Barak (2009).

2.1 Complexité des problèmes d'optimisation

Dans cette première section, nous introduisons les différentes notions de complexité utilisées tout au long de cette thèse.

2.1.1 Problèmes P et problèmes NP

Pour un problème donné, on peut généralement imaginer plusieurs algorithmes de résolution. Ces algorithmes n'ont à priori aucune raison de nécessiter le même temps de calcul, ou du même espace en mémoire. On peut donc souhaiter associer à ces algorithmes une mesure de complexité relatant leur temps de calcul ou leur consommation en mémoire. Le temps de calcul n'est pas un bon indicateur en soi. Un ordinateur plus puissant ira sûrement plus vite qu'un ordinateur plus lent. On préfère donc compter le nombre d'étapes élémentaires nécessaire à la complétion de l'algorithme. Cette définition rend la complexité en temps de calcul indépendante de la puissance d'un ordinateur. Elle reste cependant sensible à la définition d'une étape élémentaire, elle même tributaire du modèle de calcul adopté. Sans rentrer dans les détails, disons qu'il y a consensus dans la discipline sur ce qu'est un calcul élémentaire. On retient généralement comme modèle la machine de Turing, que l'on sait équivalente aux autres modèles importants. La thèse de Church-Turing laisse espérer que le modèle est suffisamment expressif pour être universel. Dans la suite de cette thèse, on s'intéressera principalement à la complexité en temps. On définira la complexité d'un problème comme la complexité de l'algorithme le plus efficace

permettant de le résoudre dans le pire des cas. Voir l'introduction de Arora et Barak (2009) pour plus de détails.

Une fois l'idée de complexité introduite, on peut imaginer différentes classes, caractérisées par leur temps de calcul. Dans ce paragraphe, on se contentera des problèmes de décision, où les problèmes sont vus comme des questions dont la réponse ne peut être que oui ou non. Les problèmes de décision pouvant être résolus en un temps polynomial constituent la classe P. Les problèmes dont la réponse peut être vérifiée en temps polynomial forment la classe NP. La classe P est donc inclue dans la classe NP. Si certains problèmes de NP admettent des algorithmes simples en temps polynomial, il en existe d'autres pour lesquels on ne connaît que des solutions dont le temps de calcul est exponentiel. Il semble qu'il existe des problèmes dans NP qui n'admettent aucun algorithme efficace. C'est la conjecture $P \neq NP$.

2.1.2 Problèmes NP complets, et NP difficiles

Les problèmes les plus difficiles de NP constituent la classe des problèmes NP-difficiles. Cette classe regroupe en effet l'ensemble des problèmes au moins aussi difficiles que les plus difficiles problèmes de NP. Pour définir cette classe plus précisément, il faut introduire la notion de réduction en temps polynomial. On dit alors qu'on a réduit le problème initial au problème que l'on s'est autorisé à appeler. Si un algorithme autorisé à en appeler un autre, ce en comptant cet appel comme une seule étape de calcul élémentaire, permet de résoudre un problème en temps polynomial, on dit que ce problème est réductible au problème résolu par l'algorithme appelé.

Cette notion de réductibilité permet d'envisager plus précisément la notion de problème NP-difficile. Un problème est NP-difficile si il permet de réduire tous les problèmes de NP. Autrement dit si un problème NP-difficile est dans P, alors $P = NP$. Si un problème NP-difficile est dans NP, on dira alors qu'il est NP-complet. Cette définition peut sembler étonnante, car elle sous-entend l'existence d'un (ou plusieurs) problème(s) de NP plus difficile(s) que tous les autres. L'existence de ces problèmes est garantie par le théorème de Cook-Levin (Cook, 1971).

Ce théorème porte sur la complexité de SAT, un problème où il faut assigner à des booléens une valeur leur permettant de satisfaire une formule. Ce genre de formule sera composée d'un certain nombre de variables, de conjonctions (dénotées $\cdot \wedge \cdot$), de disjonctions (notées $\cdot \vee \cdot$), et de négations ($\neg \cdot$). Dans SAT, on demande à la formule d'être sous forme normale conjonctive (CNF), c'est à dire que l'on ne s'intéressera qu'aux conjonctions de disjonctions de variables ou de leur négation. Par exemple

$$(b_1 \vee b_2 \vee \neg b_3) \wedge (b_2 \vee b_3 \vee \neg b_4)$$

est une 3-CNF, chaque clause contenant 3 variables ou leur négation. Le théorème de Cook-Levin affirme que SAT, le problème de décider si une CNF est satisfaisable, est NP-complet.

Une fois établie la difficulté de SAT, on peut démontrer que plusieurs autres problèmes sont difficiles en réduisant SAT à ces problèmes. Une fois ces problèmes démontrés difficiles, on peut les utiliser pour montrer que d'autres problèmes sont NP-difficiles. C'est le travail effectué par Karp, qui a montré que 21 problèmes clas-

siques était NP-difficiles. Un exemple couramment utilisé pour illustrer un problème NP-difficile est le problème du voyageur de commerce.

2.1.3 Aperçu du reste du zoo

Il existe beaucoup de classes de complexité, autres que P et NP. Le travail de Aaronson *et al.* (2015) en recense 417. Nous nous contentons d'introduire brièvement quelques-unes de ces classes dans ce paragraphe. Une classe de problèmes que l'on rencontrera souvent par la suite est la classe des problèmes NPO, pour problème d'optimisation NP. Ces problèmes ne sont pas des problèmes de décision, où la réponse attendue serait oui ou non, mais des problèmes d'optimisation, où l'on se demande plutôt quelle est la meilleure solution à un problème. Ainsi, décider si une matrice a un rang inférieur à k est dans NP (et même dans P), mais calculer le rang d'une matrice est dans NPO. Dans la littérature, comme dans cette thèse, on parle souvent de problème d'optimisation NP-difficile, même si ce n'est pas exact. On sous-entend en fait que le problème de décision correspondant est NP-difficile. Il existe encore bien d'autres classes de complexité, y compris à l'intérieur de NP. Certains problèmes NP-difficiles admettent par exemple des algorithmes produisant des solutions approximatives de façon efficace. On distingue plusieurs types d'approximation, en fonction du ratio solution proposée/solution optimale garanti. Les problèmes pour lesquels on dispose d'algorithmes en temps polynomial pouvant atteindre la solution optimale à une constante multiplicative fixe sont dans APX. Parmi ces problèmes, on distingue la classe des PTAS, problèmes pour lesquels il existe un algorithme en temps polynomial pour tout ratio inférieur à 1. Quand on fait face à un problème NP-dur, l'approximation est l'une des stratégies efficaces de résolution. On en donnera un exemple dans le paragraphe 2.2.2. Une autre stratégie est de distinguer des instances faciles d'un problème NP-dur. La complexité d'un problème est en effet définie dans le pire des cas. Souvent, il existe des instances faciles, voire triviales d'un problème.

2.1.4 Complexité de *k-means*

On a déjà parlé de *k-means* dans le premier chapitre. On a mentionné plusieurs problèmes concernant cet algorithme, notamment sa dépendance à la façon dont on l'initialise. Un obstacle supplémentaire au problème des *k-means* est sa complexité. On peut en effet montrer que *k-means* est NP-difficile (Aloise *et al.*, 2009). On peut se demander si cette difficulté concerne la majorité des instances naturelles de *k-means*, ou si il existe une partie importante de ces problèmes pour lesquels il existe un algorithme efficace. On se concentrera sur l'étude d'un autre type de *clustering* par la suite. On commencera par montrer dans le paragraphe 2.2.2 une relaxation classique d'un autre problème de *clustering* : *max-cut*. On montrera ensuite dans le paragraphe 2.4 que pour des problèmes de *clustering* où les données sont issues d'une certaines densité de probabilité, il existe des algorithmes efficaces.

2.2 Le problème *max-cut*

Max-cut est un problème NP-difficile classique. C'est un exemple de problème pour lequel il n'existe pas d'algorithme polynomial approximant arbitrairement bien la meilleure solution. Le problème admet cependant une relaxation à travers un SDP dont la solution proposée sera en moyenne assez bonne.

2.2.1 Description

Soit un graphe $G = (V, E)$, avec une fonction de coût $a : E \to \mathbb{R}^+$. Le but de *max-cut* est de partitionner les sommets en deux groupes, de façon à maximiser la somme des poids des arêtes liant ces deux groupes. Posons A la matrice des coûts définie pour $1 \leq i, j \leq n$ comme

$$a_{i,j} = \begin{cases} a\left((v_i, v_j)\right), \text{ si } (v_i, v_j) \in E, \\ 0 \text{ sinon.} \end{cases}$$

On peut alors écrire *max-cut* comme

$$\max_{x \in \{-1,1\}^n} \sum_{1 \leq i,j \leq n} a_{i,j} \frac{1 - x_i x_j}{2}.$$

On peut ensuite ré-écrire ce problème comme un programme quadratique binaire,

$$\max_{x \in \{-1,1\}^n} \frac{1}{2} x^T L x,$$

où L est le Lagrangien correspondant à A, défini comme $L = \text{diag}(A1_n) - A$.

2.2.2 Relaxation par SDP

L'idée de Goemans et Williamson (1995) est de plonger le problème dans $\mathbb{R}^n$,

$$\max_{x_i \in \mathbb{R}^n, \|x_i\| = 1, 1 \leq i \leq n} \sum_{i<j} a_{i,j}(1 - x_i^T x_j).$$

C'est un programme semi-défini, que l'on peut réécrire sous une forme plus classique

$$\max_{X \in \mathbb{S}_n^+} \text{tr}\, LX \text{ tel que } \text{diag}(X) = I_n.$$

Une fois le problème SDP résolu, on construit une partition $\{S, T\}$ en choisissant aléatoirement un vecteur v unitaire de $\mathbb{R}^n$, et en posant

$$S = \left\{ i \in \{1, \cdots, n\}, x_i^T v > 0 \right\}, T = \left\{ i \in \{1, \cdots, n\}, x_i^T v < 0 \right\}.$$

On obtient une coupe dont l'espérance du poids est supérieure à $0.878 a_{\max}$ avec $a_{\max}$ le poids de la coupe maximale. On peut trouver plus de détails dans Goemans et Williamson (1995), où dans Lovász (2003). Si un SDP peut être résolu en temps polynomial en sa taille (par des algorithmes de point intérieur), on ne peut pas

en pratique dépasser une taille de l'ordre de quelques milliers. De plus, garder en mémoire une matrice de taille $n \times n$ devient rapidement très contraignant. Un théorème de Pataki (1998) assure qu'il existe une solution X à tout SDP à m contraintes vérifiant

$$\frac{\operatorname{rank} X\,(\operatorname{rank} X + 1)}{2} > m.$$

Pour *max-cut* il existe donc une solution de rang $r = \lceil \sqrt{2n} \rceil$. On peut donc restreindre l'espace des recherches à ces matrices. L'astuce de Bürer-Monteiro consiste à remplacer une contrainte type $X \succeq 0, \operatorname{rank} X = k$ par un problème non-linéaire équivalent, où X est remplacé par $YY^T, Y \in \mathbb{R}^{n\times k}$. Dans le cas de *max-cut*, cela revient à résoudre le problème non-linéaire suivant

$$\max_{Y\in\mathbb{R}^{n\times k}} \operatorname{tr}\left(LYY^T\right) \text{ tel que } \operatorname{diag} YY^T = I_n. \tag{2.1}$$

Trouver le minimum global de ce genre de problème non-linéaire n'est généralement pas aisé. En revanche, dans le cas du problème (2.1), un théorème de Boumal *et al.* (2016) assure que tout point critique d'ordre 2 est un minimum global.

2.3 Détection de communautés dans les graphes aléatoires

La relaxation de *max-cut* est générique, en ce qu'elle concerne tous les graphes. Elle n'exclut aucunement que certaines classes de graphes admettent des solutions exactes en temps polynomial. On peut penser à un graphe cyclique avec des poids uniformes par exemple. L'exemple n'est certes pas très intéressant, mais on peut se demander si il existe des classes de graphes plus pertinentes pour lesquelles on dispose d'une solution en temps polynomial. C'est le cas par exemple pour certains modèles de graphes aléatoires, pour lesquels nous allons maintenant introduire une relaxation.

2.3.1 Graphes aléatoires

Dans cette partie, nous allons discuter de certains modèles de graphes aléatoires, et de ce qu'on peut dire de *max-cut* sur ces graphes. Un des modèles de graphe courant est celui de Erdös et Renyi, $G(n, p)$ sur n sommets. Une arête entre deux sommets existe avec une probabilité de p, et ce indépendamment de toutes les autres. Erdös et Rényi ont étudié diverses propriétés de ces graphes, comme leur connectivité. Ces graphes ne tendent pas naturellement à former des communautés ou *clusters*, ce qui les rend peu intéressants pour étudier le comportement de *max-cut*.

Nous allons donc nous intéresser à une généralisation de ce modèle, le modèle stochastique par bloc, ou SBM. Dans ce modèle, on spécifie en plus du nombre n de sommets

- une partition C de $\{1, \cdots, n\}$ en k communautés,
- une matrice P symétrique $k \times k$ de probabilités.

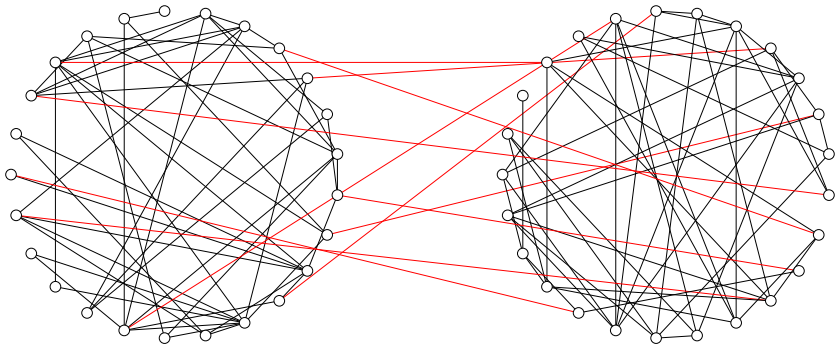


FIGURE 2.1 – Deux communautés

Une arête entre deux sommets i et j existe avec la probabilité définie dans P_{C_i,C_j}. Quand P est égale à pE_k, on retombe sur le modèle d'Erdös et Rényi. Quand en revanche la matrice P vérifie $p_{i,i} > p_{i,j}, i \neq j$, des communautés apparaissent. Nous nous restreindrons dans la suite de ce chapitre à un modèle ou les probabilités d'apparition d'une arête intra-communauté sont toutes égales à p, et où les probabilités d'apparition d'une arête extra-communauté sont toutes égales à q.

Deux régimes sont habituellement étudiés, le régime sparse, où

$$p = \frac{a}{n}, q = \frac{b}{n},$$

avec a et b deux constantes positives, et le régime relativement dense

$$p = \frac{a \log n}{n}, q = \frac{b \log n}{n}.$$

Dans le régime sparse, certains noeuds vont se retrouver isolés avec une grande probabilité, et il est donc souvent impossible de recouvrer parfaitement et globalement les communautés.

2.3.2 Détection des communautés

Récemment Guédon et Vershynin (2016) ont étudié la détection de communautés sur les graphes aléatoires issus de ce modèle, où la probabilité d'apparition d'une arête intra-communauté est donnée par $p = \frac{a}{n}$ et celle d'apparition d'une arête entre communautés est quant à elle donnée par $q = \frac{b}{n}$. L'article explique comment la solution d'un certain programme SDP permet de retrouver les différentes communautés d'un SBM avec forte probabilité à partir de l'observation d'un graphe issu de ce modèle.

On donne ici le canevas de la preuve proposée par les auteurs, car il sert de guide à la section suivante. L'article commence par donner une description de l'inégalité de Grothendieck, l'une des clés de la démonstration.

Inégalité de Grothendieck

Nous donnons ici une description de la version de l'inégalité utilisée dans l'article. Cette dernière énonce que, si une matrice $B \in \mathbb{R}^{n\times n}$ vérifie

$$\forall s_i, t_i \in \{-1, 1\}, \left|\sum_{i,j} b_{i,j} s_i t_j\right| \leq 1, \tag{2.2}$$

alors

$$\forall x_i, y_i \in B_2^n, \sum b_{i,j} x_i^T y_j \leq K_G, \tag{2.3}$$

où B_2^n désigne la boule unité en dimension n pour la norme euclidienne. L'article en propose une écriture différente, faisant apparaître un SDP, ainsi qu'une norme matricielle notée $\|\cdot\|_{\infty\to 1}$.

Cette norme est habituellement définie comme

$$\|B\|_{\infty\to 1} = \max_{\|x\|_\infty = 1} \|Bx\|_1 .$$

On remarque que la condition (2.2) équivaut à

$$\|B\|_{\infty\to 1} \leq 1.$$

L'équation (2.3) revient quant à elle à

$$\max_{x_i, y_i \in B_2^n} \operatorname{tr}(BX^TY) \leq K_G.$$

Pour faire apparaître un SDP, l'article restreint l'ensemble sur lequel porte le maximum, pour obtenir

$$\max_{Z\in\mathbb{S}_n, \operatorname{diag}(Z)\preceq I_n} \operatorname{tr}(BZ) \leq K_G.$$

On arrive ainsi à la forme finale utilisée dans l'article

$$\max_{\substack{X\in\mathbb{S}_n^+,\\ \operatorname{diag}(X)\preceq I_n}} \operatorname{tr}(BX) \leq K_G \|B\|_{\infty\to 1}. \tag{2.4}$$

Grâce à cette inégalité, l'article parvient à démontrer l'exactitude de la démarche proposée.

Approche permettant de retrouver les communautés

Nous décrivons maintenant les différents éléments intervenant dans la procédure décrite dans l'article pour retrouver les communautés d'un modèle SBM à partir de l'observation d'une instance de graphe. Nous esquissons ensuite une partie de la preuve proposée par l'article pour établir l'exactitude de la méthode.

Dans la suite de ce chapitre, la matrice d'adjacence du graphe observé sera notée A, et son espérance $\bar{A}$. Pour simplifier l'exposition, nous nous contenterons de deux communautés C_1 et C_2 dans ce paragraphe, et nous dénoterons par $\bar{z}$ l'indicateur de ces dernières. Celui-ci est défini comme

$$\bar{z}_i = \begin{cases} 1 \text{ si } i \in C_1 \\ -1 \text{ si } i \in C_2. \end{cases}$$

L'article démontre qu'il est possible de récupérer les communautés du SBM en résolvant le problème SDP suivant

$$\max_{\substack{Z\in\mathbb{S}_n^+,\\ \operatorname{diag}(Z)\preceq I_n}} \operatorname{tr}\left[\left(A-\frac{p-q}{2}1_n1_n^T\right)Z\right], \tag{2.5}$$

où 1_n désigne un vecteur colonne de taille n dont tous les éléments valent 1. On peut en effet démontrer que la solution à ce problème SDP, notée $\hat{Z}$ par la suite, est proche de $\bar{z}\bar{z}^T$. La démonstration se fait en deux temps. Dans un premier temps, l'article établit une borne sur $\|A-\mathbb{E}A\|_{\infty\to 1}$. À partir de cette borne, et grâce à l'inégalité de Grothendieck, l'article parvient ensuite à borner $\|\hat{Z}-\bar{z}\bar{z}^T\|_F^2$.

La première borne, portant sur la différence entre A et son espérance, est établie par l'inégalité de Bernstein. Nous ne donnons pas plus de détails sur cette première partie de la preuve, car cette inégalité exige l'indépendance des variables étudiées, condition non vérifiée dans le cadre que nous étudierons par la suite, section 2.4. Nous nous contentons dans un premier temps de montrer comment l'article obtient la seconde borne à partir de la première. Plus précisément, nous exposons la démonstration du théorème suivant.

Théorème 2.3.1. *Soit $\hat{Z}$ une solution au problème SDP 2.5. Alors*

$$\|\hat{Z}-\bar{z}\bar{z}^T\|_F^2 \leq \frac{8K_g}{p-q}\left(\left\|A-\bar{A}\right\|_{\infty\to 1}+(1-p)n\right) \tag{2.6}$$

Démonstration. On suppose dans cette preuve que les deux communautés sont définies comme

$$C_1=\{1,\cdots,\frac{n}{2}\} \text{ et } C_2=\{\frac{n}{2}+1,\cdots,n\}.$$

On note A la matrice d'adjacence du graphe issu de ce SBM, et $\bar{A}$ son espérance. On considère que le SBM génère une boucle sur chaque sommet du graphe. Cela implique que $A_{i,i}=1$, et $\bar{A}_{i,i}=1$. L'espérance de A vérifie donc

$$\bar{A}=\begin{pmatrix} pE_{n/2} & qE_{n/2}\\ qE_{n/2} & pE_{n/2}\end{pmatrix}+(1-p)I_n,$$

où $E_n=1_n1_n^T$ désigne une matrice de taille $n\times n$ remplie de 1. On pose ensuite $B=A-\frac{p-q}{2}E_n$, ainsi que

$$Z_R=\begin{pmatrix} E_{n/2} & -E_{n/2}\\ -E_{n/2} & E_{n/2}\end{pmatrix} \text{ et } R=\frac{p-q}{2}Z_R.$$

On peut remarquer que l'espérance de B et la matrice R sont liées par l'équation $\bar{B}=R+(1-p)I_n$, et que Z_R est solution de

$$\max_{\substack{Z\in\mathbb{S}_n^+,\\ \operatorname{diag}(Z)\preceq I_n}} \operatorname{tr} RZ.$$

Grâce à l'inégalité de Grothendieck, nous allons maintenant montrer que $\hat{Z}$, la solution au problème SDP

$$\max_{\substack{Z\in\mathbb{S}_n^+,\\ \operatorname{diag}(Z)\preceq I_n}} \operatorname{tr}(BZ),$$

est proche de Z_R. En effet, d'après cette inégalité, pour tout $Z \succeq 0, Z \preceq I_n$,

$$|\operatorname{tr}(B-R)Z| \leq K_g \|B-R\|_{\infty\to 1}.$$

On a donc

$$\begin{aligned}\operatorname{tr} R\hat{Z} &\geq \operatorname{tr} B\hat{Z} - K_g \|B-R\|_{\infty\to 1} \\ &\geq \operatorname{tr} BZ_R - K_g \|B-R\|_{\infty\to 1} \\ &\geq \operatorname{tr} RZ_R - 2K_g \|B-R\|_{\infty\to 1}.\end{aligned}$$

Ainsi

$$\operatorname{tr} Z_R\hat{Z} \geq \operatorname{tr} Z_R Z_R - \frac{4K_g}{p-q}\|B-R\|_{\infty\to 1}.$$

Il reste à borner $\|B-R\|_{\infty\to 1}$. Pour ce, on utilise l'inégalité triangulaire.

$$\begin{aligned}\|B-R\|_{\infty\to 1} &\leq \left\|B-\bar{B}\right\|_{\infty\to 1} + \left\|\bar{B}-R\right\|_{\infty\to 1} \\ &\leq \left\|A-\bar{A}\right\|_{\infty\to 1} + (1-p)\|I_n\|_{\infty\to 1} \\ &\leq \left\|A-\bar{A}\right\|_{\infty\to 1} + (1-p)n.\end{aligned}$$

On obtient finalement,

$$\begin{aligned}\|\hat{Z}-Z_R\|_F^2 &= \|\hat{Z}\|_F^2 + \|Z_R\|_F^2 - 2\operatorname{tr}\hat{Z}Z_R \\ &\leq 2\|Z_R\|_F^2 - 2\operatorname{tr}\hat{Z}R, \\ &\leq \frac{8K_g}{p-q}\left(\left\|A-\bar{A}\right\|_{\infty\to 1} + (1-p)n\right).\end{aligned}$$

□

Cette dernière équation dépend d'une borne sur $\left\|A-\bar{A}\right\|_{\infty\to 1}$, borne que nous allons établir dans la prochaine section, dans un cadre différent de celui envisagé dans ce paragraphe.

2.4 *Clustering* de gaussiennes

Nous allons ici exposer les étapes principales de la relaxation SDP au *clustering* de gaussiennes que nous avons proposée dans Chrétien *et al.* (2016). Le cadre est assez différent de celui de la détection de communautés engendrées par SBM. Pas de graphes aléatoires ici, juste l'observation d'un mélange de deux gaussiennes. Pour simplifier l'exposition, nous nous contenterons de deux *clusters* dans ce paragraphe. On généralisera ensuite de 2 à k gaussiennes dans le chapitre suivant.

On observe n tirages selon la densité

$$x_i \sim \mathcal{N}(\mu_k, \Sigma_k) \quad \text{si } i \in \mathcal{C}_k,$$

où C_1, C_2 forment une partition de $\{1, \cdots, n\}$, $\mu_1, \mu_2 \in \mathbb{R}^d$ sont les moyennes des gaussiennes, et $\Sigma_1, \Sigma_2 \in \mathbb{S}_d$ leurs matrices de covariance. À défaut de matrice d'adjacence, on construit la matrice A comme

$$A = \left(f(\|x_i - x_j\|_2)\right)_{1\leq i,j\leq n}, \tag{2.7}$$

où $f : [0, +\infty) \to [0, 1]$ est une fonction d'affinité. Un choix courant est l'affinité gaussienne

$$f(h) = e^{-\frac{h^2}{2\lambda^2}}. \tag{2.8}$$

C'est le choix retenu pour la suite de cette section.

Cette matrice nous permet d'adapter la méthode de Guédon et Vershynin (2016) exposée au paragraphe précédent. Le théorème 2.3.1 reste en effet valide, en posant

$$\begin{cases} p = \mathbb{E}A_{i,j}, \text{ pour } i, j \text{ dans le même } \textit{cluster} \text{ et } i \neq j, \\ q = \mathbb{E}A_{i,j}, \text{ pour } i, j \text{ dans deux } \textit{clusters} \text{ différents et } i \neq j. \end{cases}$$

Il reste donc à borner $\|A - \mathbb{E}A\|_{\infty \to 1}$. Le calcul de cette borne est l'une des contributions principales de Chrétien *et al.* (2016). La technique de démonstration est très différente de celle utilisée dans Guédon et Vershynin (2016). Les entrées de la matrice A ne sont en effet plus indépendantes les unes des autres.

Pour établir cette borne, on commence par calculer $\bar{A}$.

Calcul de $\bar{A}$

La différence $x_i - x_j$ suit une loi normale centrée en $\mu = \mu_{k_i} - \mu_{k_j}$ de variance $\Sigma = \Sigma_{k_i} + \Sigma_{k_j}$. Il faut donc calculer $\mathbb{E}\exp(-\frac{z^T z}{2\lambda^2})$, pour $z \sim \mathcal{N}(\mu, \Sigma)$. Pour ce, on fait apparaître la transformée de Laplace de z.

$$\begin{aligned}
&\int_{\mathbb{R}^d} \exp(-\frac{z^T z}{2\lambda^2}) \left((2\pi)^d \det \Sigma\right)^{-\frac{1}{2}} \exp\left(-\frac{1}{2}(z-\mu)^T \Sigma^{-1}(z-\mu)\right) \mathrm{d}x \\
&= \det(\Sigma)^{-\frac{1}{2}} \int_{\mathbb{R}^d} \exp\left(-\frac{(z-\mu)^T(I_d + \lambda^2\Sigma^{-1})(z-\mu) + 2\mu^T z - \mu^t\mu}{2\lambda^2}\right) (2\pi)^{-\frac{d}{2}} \mathrm{d}x \\
&= \det(\frac{1}{\lambda^2}\Sigma + I_d)^{-\frac{1}{2}} \exp\left(\frac{\mu^t\mu}{2\lambda^2}\right) \exp\left(-\frac{\mu^T\mu}{\lambda^2} + \frac{\mu^T\left(\frac{1}{\lambda^2}I_d + \Sigma^{-1}\right)^{-1}\mu}{2\lambda^4}\right) \\
&= \lambda^d \det(\Sigma + \lambda^2 I_d)^{-\frac{1}{2}} \exp\left(-\frac{\mu^T\left(\Sigma + \lambda^2 I_d\right)^{-1}\mu}{2}\right)
\end{aligned}$$

On en déduit, pour $x_i \in C_k, x_j \in C_l$, $i \neq j$,

$$\bar{A}_{i,j} = \lambda^d \det(\Sigma_k + \Sigma_l + \lambda^2 I_d)^{-\frac{1}{2}} \exp\left(-\frac{(\mu_k - \mu_l)^T\left(\Sigma_k + \Sigma_l + \lambda^2 I_d\right)^{-1}(\mu_k - \mu_l)}{2}\right).$$

Pour $i = j$, on a évidemment $\bar{A}_{i,j} = 1$.

Exemple 2.4.1. *Si on prend deux gaussiennes centrées réduites de dimension* 2 *séparées de* δ*, on a*

$$\bar{A}_{i,j} = \frac{\lambda^d}{\lambda^2 + 2}\left(1_{i \sim j} + 1_{i \nsim j} \exp(-\frac{\delta^2}{4 + 2\lambda^2})\right)$$

Concentration de A autour de $\bar{A}$

On démontre ensuite la concentration de A autour de $\bar{A}$ par une inégalité de concentration pour les fonctions lipschitziennes de gaussiennes centrées réduites. On commence donc par montrer que la fonctionnelle $F_{u,v}$ définie ci-dessous est lipschitzienne. On pose

$$F_{uv}(x_1, \cdots, x_n) = \sum_{i,j=1}^{n} u_i v_j \left(f(\|x_i - x_j\|_2) - \bar{A}_{i,j} \right), \tag{2.9}$$

de façon à avoir

$$\|A - \bar{A}\|_{\infty \to 1} = \max_{u,v \in \{-1,1\}^n} F_{uv}.$$

La fonction F_{uv} ne dépend pas de variables gaussiennes centrées réduites. On introduit donc

$$g_i : y \to \Sigma_{k_i}^{\frac{1}{2}} y_i + \mu_{k_i},$$

pour tout $1 \leq i \leq n$, ainsi que

$$\tilde{F}_{uv}(y_1, \cdots, y_n) = \sum_{i,j=1}^{n} u_i v_j \left(f(\|g_i(y_i) - g_j(y_j)\|_2) - \bar{A}_{i,j} \right),$$

qui elle permet d'écrire $\|A - \bar{A}\|_{\infty \to 1}$ à partir de gaussiennes centrées réduites.

Pour $y_1, \cdots y_n \in \mathbb{R}^d$ et $y'_1, \cdots y'_n \in \mathbb{R}^d$, on a

$$\begin{aligned}
\left| \tilde{F}_{uv}(y_1, \cdots, y_n) - \tilde{F}_{uv}(y'_1, \cdots, y'_n) \right| &\leq \ell \sum_{1 \leq i,j \leq n} \left[\|g_i(y_i) - g_j(y_j)\|_2 + \|g_i(y'_i) - g_j(y'_j)\|_2 \right] \\
&\leq \ell \sum_{1 \leq i,j \leq n} \left[\|g_i(y_i) - g_i(y_i)'\|_2 + \|g_j(y_j) - g_j(y'_j)\|_2 \right] \\
&= 2n\ell \sum_{i=1}^{n} \|\Sigma_{k_i}^{1/2}(y_i - y'_i)\|_2 \\
&\leq 2n\ell \sum_{i=1}^{n} \rho(\Sigma_{k_i})^{1/2} \|y_i - y'_i\|_2 \\
&\leq 2\ell\sigma n^{3/2} \|(y_1, \ldots, y_n) - (y'_1, \ldots, y'_n)\|_F,
\end{aligned}$$

où $\rho(\Sigma_{k_i})$ désigne le rayon spectral de Σ_{k_i}, l est la constante de Lipschitz de f, et

$$\sigma^2 = \frac{1}{n} \sum_{i=1}^{n} \rho(\Sigma_{k_i}).$$

La dernière inégalité s'obtient grâce à Cauchy-Schwartz. Comme $l = 1/\sqrt{e\lambda^2}$, la constante de Lipschitz de $\tilde{F}_{u,v}$ est

$$L = \frac{2\sigma n^{3/2}}{\sqrt{e\lambda^2}}.$$

Il ne reste plus qu'à utiliser deux inégalités de concentrations, tirées de Boucheron *et al.* (2013). La première est une inégalité type Log-Sobolev.

Théorème 2.4.1. *Soient $Y_1, \ldots, Y_n$ des vecteurs aléatoires gaussiens indépendants sur $\mathbb{R}^d$ centrés réduits. Supposons que $F : \mathbb{R}^{n\times d} \to \mathbb{R}$ soit lipschitz de constante L. Alors la variable aléatoire $F = F(Y_1, \ldots, Y_n)$ vérifie*

$$\mathbb{E}[\exp(\theta\,(F - \mathbb{E}F))] \leq \exp(L^2\theta^2/2) \quad \textit{pour tout } \theta \in \mathbb{R},$$

ainsi que

$$\mathbb{P}(|F - \mathbb{E}F| > t) \leq 2\exp\left(-t^2/(8L^2)\right) \quad \textit{pour tout } t > 0.$$

Le second permet de borner l'espérance du maximum de variables sous-gaussiennes.

Théorème 2.4.2. *Soient $Z_1, \cdots, Z_N$ des variables aléatoires réelles sous-gaussiennes de facteur ν, i.e. respectant*

$$\mathbb{E}[\exp(\theta Z_i)] \leq \exp(\nu\theta^2/2) \quad \textit{pour tout } \theta \in \mathbb{R}.$$

Alors

$$\mathbb{E}\left[\max_{i=1,\cdots,N} Z_i\right] \leq \sqrt{2\nu \log N}.$$

En appliquant la première inégalité (théorème 2.4.1) dite de Tsirelson-Ibragimov-Sudakov, on obtient

$$\mathbb{E}\left[\exp(\theta\tilde{F}_{uv})\right] \leq \exp\left(L^2\theta^2/2\right) \quad \text{for all} \quad \theta \in \mathbb{R}.$$

Pour cela, il faut remarquer que $\mathbb{E}F_{u,v} = 0$. Cela permet d'établir que $F_{u,v}$ est sous-gaussienne. On peut donc appliquer la seconde inégalité (théorème 2.4.2), dite de Dudley, pour obtenir

$$\begin{aligned}
\mathbb{E}\left[\|A - \bar{A}\|_{\infty\to 1}\right] &= \mathbb{E}\left[\max_{u,v\in\{-1,1\}^n} F_{uv}\right] \\
&\leq \sqrt{2L^2 \log 2^n} \\
&= 2l\sigma n^2\sqrt{2\log 2}.
\end{aligned}$$

La deuxième partie du théorème 2.4.1 nous donne

$$\begin{aligned}
\mathrm{P}\left(|\|A - \bar{A}\|_{\infty\to 1} - \mathrm{E}\|A - \bar{A}\|_{\infty\to 1}| > t\right) &= \mathrm{P}\left(|\max_{u,v\in\{-1,1\}^n} F_{uv} - \mathrm{E}\max_{u,v\in\{-1,1\}^n} F_{uv}| > t\right) \\
&\leq 2\exp\left(-\frac{t^2}{8L^2}\right).
\end{aligned}$$

Il reste alors à tout regrouper. Pour $t > 2\sqrt{2\log 2}\ell\sigma$, on a ainsi

$$\mathrm{P}(\left\|A - \bar{A}\right\|_{\infty\to 1} > tn^2) \leq 2\exp\left(-\frac{\left(t - 2\sqrt{2\log 2}\ell\sigma\right)^2}{32\ell^2\sigma^2}n\right). \tag{2.10}$$

Conclusion

En insérant la borne (2.10) dans l'équation (2.6), on obtient finalement :

$$\|\hat{Z} - Z_R\|_F^2 \leq \frac{8K_g}{p-q}\left(2\ell\sigma\sqrt{2\log 2}n^2\right) + o(n^2).$$

Cette équation montre que la matrice $\hat{Z}$ obtenue en résolvant le SDP (2.5) est très proche de $Z_R = \bar{z}^T\bar{z}$, et valide donc la méthode de *clustering* par SDP développée dans cette section.

Exemple 2.4.2. *On poursuit l'exemple 2.4.1. Dans ce cas, on a* $p = \frac{\lambda^2}{2+\lambda^2}$, $\sigma^2 = 1$ *et*

$$p - q = \left[\frac{\lambda^2}{2+\lambda^2}\right]^{\frac{d}{2}}\left(1 - \exp\left(-\frac{\delta^2}{4+2\lambda^2}\right)\right).$$

En collectant les inégalités établies précédemment, on obtient

$$\|\hat{Z} - Z_R\|_F^2 \leq \frac{16\sqrt{2\log 2}K_G}{\sqrt{e}} \cdot \frac{\lambda^2+2}{\lambda^2} \cdot \frac{1}{\lambda\left(1 - \exp\left(-\frac{\delta^2}{4+2\lambda^2}\right)\right)}n^2 + o(n^2).$$

Il semble difficile de trouver une expression simple du meilleur λ *possible. La borne proposée dans cet exemple est fonction de* δ *et* λ*. La figure 2.2 montre les courbes décrites par la borne sur* $\|\hat{Z} - Z_R\|_F^2$ *en fonction de* λ^2*, et ce pour quelques valeurs de* δ*. On remarque que la borne ne devient intéressante que pour de grandes valeurs de* δ*.*

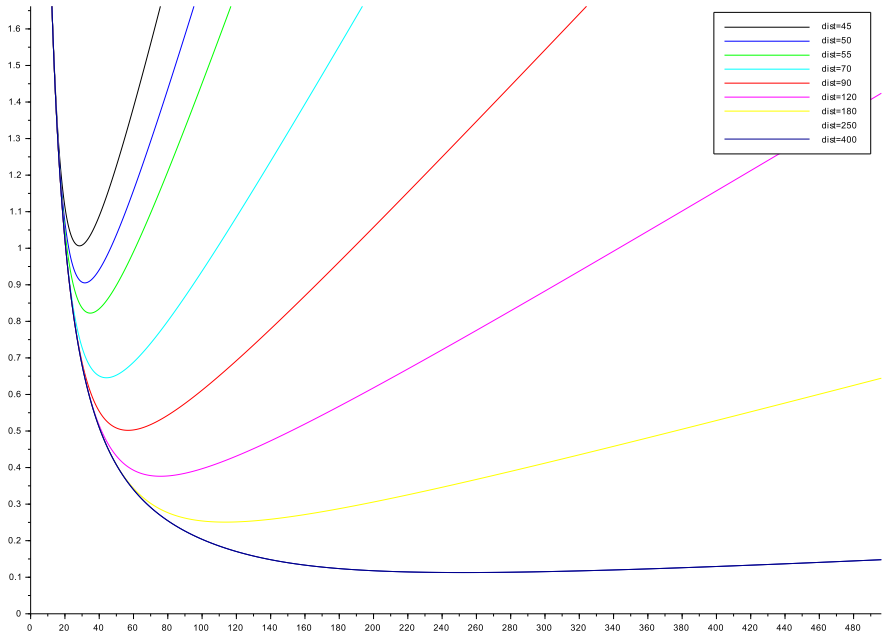


FIGURE 2.2 – Borne obtenue sur $\frac{\|\hat{Z}-Z_R\|_F^2}{n^2}$ en fonction de λ^2 pour différentes distances entre μ_1 et μ_2.

3

The Guedon-Vershynin Semi-Definite Programming approach to low dimensional embedding for unsupervised clustering

Ce chapitre est issu d'un article cosigné avec Stéphane Chrétien et Clément Dombry, proposant une approche au *clustering* de gaussiennes présentant certaines garanties théoriques. La méthode étend les résultats obtenus au chapitre précédent pour deux *clusters* à un nombre quelconque de *clusters*. L'article a été soumis à *Statistics and Computing*.

Noname manuscript No.
(will be inserted by the editor)

The Guedon-Vershynin Semi-Definite Programming approach to low dimensional embedding for unsupervised clustering

Stéphane Chrétien · Clément Dombry · Adrien Faivre

Received: date / Accepted: date

Abstract This paper proposes a method for estimating the cluster matrix in the Gaussian mixture framework via Semi-Definite Programming. Theoretical error bounds are provided and a (non linear) low dimensional embedding of the data is deduced from the cluster matrix estimate. The method and its analysis is inspired by the work by Guédon and Vershynin on community detection in the stochastic block model. The adaptation is non trivial since the model is different and new Gaussian concentration arguments are needed. We propose furthermore an eigenvalue optimization approach for solving the semi-definite program and computing the embedding. This results in an efficient and scalable algorithm taking only the pairwise distances as input. The performance of the method is illustrated via Monte Carlo experiments and comparisons with other embeddings from the literature.

1 Introduction

Low dimensional embedding is a key to many modern data analysis procedures. The main underlying idea is that the data is better understood after extracting the main features of the sample. Based on a compressed description from a few extracted features, the observations can then be projected, visualized or clustered more reliably and efficiently. The main embedding techniques available nowadays are PCA [16] and its robust version [8], random embeddings [15], Laplacian Eigenmap [4], Maximum Variance Unfolding/Semi-Definite embedding [28], ... The first two techniques in this list are linear embeddings methods, whereas the other are nonlinear in nature.

Our main objective in the present paper is to propose a technique for a low dimensional representation which aims at preparing the data for unsupervised clustering at the same time. Combining the goals of projecting and clustering is not new. This is achieved in particular by spectral clustering [25], [3, Chapter 3]. The SemiDefinite embedding technique in [17] is also motivated by clustering purposes. Spectral clustering is based on a Laplacian matrix constructed from the pairwise distances of the samples and whose second eigenvector, called the Fiedler vector, is proved to separate the data into two clusters using the normalized cut criterion. The main idea in such methods is to approximately preserve the pairwise distances. A frequent way to illustrate non-linear low dimensional embedding such as Diffusion Maps is shown in Figure 1.

S. Chrétien
National Physical Laboratory
Hampton Road
Teddington TW11 0LW, UK E-mail: stephane.chretien@npl.co.uk

C. Dombry
Univ. Bourgogne Franche-Comté
Laboratoire de Mathématiques de Besançon, UMR CNRS 6623
16 route de Gray, 25030 Besançon cedex, France. E-mail: clement.dombry@univ-fcomte.fr

A. Faivre
DigitalSurf
16 rue Lavoisier, 25000 Besançon
Besançon, France E-mail: afaivre@digitalsurf.fr

(a) Original 3D Cluster

(b) Mapped data using Diffusion Maps

Fig. 1: The mapping of a 3D cluster using Diffusion Maps from the Matlab package drtoolbox https://lvdmaaten.github.io/drtoolbox/

In this paper, we develop an original method for low dimensional embedding and clustering. Our starting point is the method by Guedon and Vershynin [10] initially developed for community detection in the stochastic block model (SBM). The SBM considers a random graph based on a set of vertices partitioned into clusters and with random edges between vertices, all edges are independent and the probabilities of edges depend only on the cluster structure. It is assumed that these probabilities are larger within clusters than across clusters and the aim is to detect the clusters, see Abbe et al. [1], Heimlicher et al. [11], Mossel et al. [19]. Guedon and Vershynin show that the cluster matrix can be estimated via Semi-Definite Programming (SDP) with an explicit control of the error rate. We adapt the method to deal with the Gaussian Cluster Model (GCM) and show that the cluster matrix in the GCM can be estimated by solving an SDP with an affinity matrix as input that depends only on the pairwise distances between observations. Contrary to the adjacency matrix from the SBM, the affinity matrix from the GCM has non independent entries, making the analysis and the adaptation non trivial. Furthermore, we show that the estimated cluster matrix yields not only a clustering of the data, but also a natural associated embedding. Indeed, quite similarly to spectral clustering, the eigenvectors of the estimated cluster matrix provide a meaningful embedding. Contrary to the standard embedding methods discussed in the paragraph above, the embedding does not try to preserve pairwise distances but rather to estimate the cluster matrix. The intuition for the cluster matrix is supported by Remark 1.6 in [10] which we now quote: *It may be convenient to view the cluster matrix as the adjacency matrix of the cluster graph, in which all vertices within each community are connected and there are no connections across the communities. This way, the Semi-Definite program takes a sparse graph as an input, and it returns an estimate of the cluster graph as an output. The effect of the program is thus to "densify" the network inside the communities and "sparsify" it across the communities.*

One advantage of the approach is that efficient algortihm for SDP are available, even in high dimension. Let us discuss briefly the issues related with the optimization problem associated with clustering algorithms. One major issue is that the usual objective function to optimize is non convex. For instance, in the classical mixture model approach of clustering (see McLachlan and Peel [18] for an overview of finite mixture models), the algorithm of choice has long been the EM algorithm by Dempster et al. [9]. The log-likelihood function of e.g. Gaussian mixture model is non convex and, even worse, exhibits degenerate behavior, see Biernacki and Chrétien [6]. Nonparametric methods for clustering such as K-means, K-means ++ and generalizations have been used extensively in computer science, see Jain [14] for a review. Similarly, the objective function to minimize is not convex. As a result, one can not certify that the algorithm has converged to an interesting stationary point. In practice, the algorithm is run from different starting points and the popularity of the method is based on its satisfactory average practical performance. To bypass this hurdle, recent literature focuses on clustering methods based on convex optimization. ClusterPath [13] is such a method that has been studied and extended in [23], [21] and

[27]. One drawback of the approach is the presence of tuning parameters without robust rule to select them, even though in practise they seem reasonably easy to tune.

To summarize, the main contributions of the paper are the following. We show that in the Gaussian Cluster Model, the cluster matrix can be estimated by solving an SDP program. Error bounds are available as well as an upper bound for the missclassification rate for the resulting clustering. We discuss how a low dimensional embedding can be deduced from the estimated cluster matrix. The method and its analysis are adapted from Guédon and Vershynin [10]. Because the affinity matrix associated to Gaussian clustering does not have independent entries (unlike the adjacency matrix in the stochastic block model), the adaptation is non trivial and we need to introduce concentration inequalities for Gaussian measures, see e.g. the monograph by Boucheron et al. [7]. Besides, we propose a simple and scalable algorithm to solve the Semi-Definite Program based on eigenvalue optimization in the spirit of Helmberg and Rendl [12].

The paper is organized as follows. The SDP approach for estimating the cluster matrix, the associated embedding and the main theoretical results are presented in Section 2, while the proofs are postponed to Section 3. The algorithmic considerations for the resolution of the SDP are discussed in Section 4, where an efficient algorithm is described as well as a practical method for selecting the unknown tuning parameter. Section 5 is devoted to the presentation of simulation results, demonstrating the potential of the proposed method. Some technical background on Gaussian concentration and Grothendieck inequality is provided in Appendices A and B.

2 Main results

2.1 Framework: the Gaussian Cluster Model

The mathematical framework is the following. We assume that we observe a data set $x_1, \dots, x_n \in \mathbb{R}^d$ over a population of size n. The population is partitioned into K clusters $\mathcal{C}_1, \dots, \mathcal{C}_K$ of size $n_1, \dots, n_K$ respectively, i.e. $n = n_1 + \cdots + n_K$. We assume the standard Gaussian Cluster Model for the data: the observations x_i are independent with

$$x_i \sim \mathcal{N}(\mu_k, \Sigma_k) \quad \text{if } i \in \mathcal{C}_k \tag{1}$$

with $\mu_k \in \mathbb{R}^d$ the cluster mean and $\Sigma_k \in \mathbb{R}^{d\times d}$ the cluster covariance matrix.

The clustering problem aims at recovering the clusters $\mathcal{C}_k$, $1 \leqslant k \leqslant K$, based on the data x_i, $1 \leqslant i \leqslant n$, only. For each $i = 1, \dots, n$, we will denote by k_i the index of the cluster to which i belongs. The notation $i \sim j$ will mean that i and j belong to the same cluster. The cluster matrix $\bar{Z}$ is the $n \times n$ matrix defined by

$$\bar{Z}_{i,j} = \begin{cases} 1 & \text{if } i \sim j \\ 0 & \text{otherwise} \end{cases}, \quad 1 \leqslant i, j \leqslant n. \tag{2}$$

It determines entirely the the clusters and, up to a reordering of the points, it is a block-diagonal matrix with a block of ones for each cluster.

Note that the Gaussian Cluster Model slightly differs from the usual Gaussian mixture model where the data set consists in independent observations from the Gaussian mixture $\sum_{k=1}^K \pi_k \, \mathcal{N}(\mu_k, \Sigma_k)$, with $(\pi_k)_{1\leqslant k\leqslant K}$ the mixture distribution. In the Gaussian mixture model, the cluster sizes $(n_1, \dots, n_k)$ are random with multinomial distribution of size n and probability parameters $(\pi_1, \dots, \pi_K)$.

2.2 Embedding associated with the estimated cluster matrix

We will define in the next section an estimate $\hat{Z}$ of the cluster matrix $\bar{Z}$. We discuss now how a low dimensional embedding of the data set can be associated with the estimate $\hat{Z}$. The main idea is to use the fact that the cluster matrix $\bar{Z}$ defined by (2) has a very specific eigenstructure: denoting by $\mathcal{C}_1$, …, $\mathcal{C}_K$ the index set of each cluster and by $1_{\mathcal{C}_k} \in \{0,1\}^n$ the indicator vector of cluster $\mathcal{C}_k$, we have

$$\bar{Z} = \sum_{k=1}^K 1_{\mathcal{C}_k} 1_{\mathcal{C}_k}^t$$

and we deduce that

- the rank of $\bar{Z}$ is K,
- the nonzero eigenvalues of $\bar{Z}$ are $|\mathcal{C}_1|, \ldots, |\mathcal{C}_K|$ with associated eigenvectors $1_{\mathcal{C}_1}/\sqrt{|\mathcal{C}_1|}, \ldots, 1_{\mathcal{C}_K}/\sqrt{|\mathcal{C}_1|}$.

We assume in the sequel that the cluster sizes are all different so that all non-zero eigenvalues have multiplicity one. The clusters can hence be recovered from the eigenstructure of the matrix $\bar{Z}$: the label of the sample point x_i is the index of the only eigenvector whose i-th component is non zero. Indeed, all other eigenvectors associated with a non-zero eigenvalue have i-th component equal to zero.

The estimate $\hat{Z}$ of the matrix $\bar{Z}$ can be used in practice to embed the data into the space $\mathbb{R}^K$ by associating each data point x_i to the vector consisting of the i-th coordinate of the K first eigenvectors of $\hat{Z}$. Given this embedding, if we can prove that $\hat{Z}$ accurately estimates the cluster matrix $\bar{Z}$, one can then apply any clustering method of choice to recover the clustering pattern of the original data. The next section gives a method for computing an estimator $\hat{Z}$ of $\bar{Z}$ using the SDP approach by Guedon and Vershynin.

2.3 Guedon and Vershynin's Semi-Definite Program

We now turn to the estimation $\hat{Z}$ of the cluster matrix using Guedon and Vershynin's Semi-Definite Programming based approach. Whereas Vershynin and Guédon [10] were interested in analyzing the Stochastic Block Model for community detection, we propose a study of the Gaussian Cluster Model and therefore prove that their approach has a great potential applicability in embedding of general data sets beyond the Stochastic Block Model setting.

Based on the data set $x_1, \ldots, x_n$, we construct the affinity matrix A by

$$A = \big(f(\|x_i - x_j\|_2)\big)_{1 \leqslant i,j \leqslant n} \tag{3}$$

where $\|\cdot\|_2$ denotes the Euclidean norm on $\mathbb{R}^d$ and $f : [0, +\infty) \to [0, 1]$ a non-increasing affinity function. A popular choice is the Gaussian affinity

$$f(h) = e^{-(h/h_0)^2}, \quad h \geqslant 0, \tag{4}$$

and other possibilities are

$$f(h) = e^{-(h/h_0)^a}, \quad f(h) = (1 + (h/h_0))^{-a}, \quad f(h) = (1 + e^{h/h_0})^{-a} \quad \cdots$$

Before stating the Semi-Definite Program, we introduce some matrix notations. The usual scalar product between matrices $A, B \in \mathbb{R}^{n \times n}$ is denoted by $\langle A, B \rangle = \sum_{1 \leqslant i,j \leqslant n} A_{ij} B_{ij}$. The notations $1_n \in \mathbb{R}^n$ and $1_{n \times n} \in \mathbb{R}^{n \times n}$ stand for the vector and matrices with all entries equal to 1. For a symmetric matrix $Z \in \mathbb{R}^{n \times n}$, the notation $Z \succeq 0$ means that Z the quadratic form associated to Z is non-negative while the notation $Z \geqslant 0$ means that all the entries of Z are non-negative.

With these notations, we define $\hat{Z}$ as a solution of the Semi-Definite Program

$$\text{maximize } \langle A, Z \rangle \quad \text{subject to} \quad Z \in \mathcal{M}_{opt} \tag{5}$$

with $\mathcal{M}_{opt}$ the set of symmetric matrices $Z \in \mathbb{R}^{n \times n}$ such that

$$\begin{cases} Z \succeq 0 \\ Z \geqslant 0 \\ \operatorname{diag}(Z) = 1_n \\ \langle Z, 1_{n \times n} \rangle = \lambda_0 \end{cases}. \tag{6}$$

For further reference, note that a semi-definite positive matrix Z with non-negative entries and unit diagonal must have all entries in $[0, 1]$, so that $\mathcal{M}_{opt} \subset [0, 1]^{n \times n}$.

The heuristic justifying that $\hat{Z}$ can be seen as an estimate of the cluster matrix $\bar{Z}$ is the following Lemma.

Lemma 1 *Consider the expected affinity matrix*

$$\bar{A} = \Big(\mathrm{E} f(\|x_i - x_j\|_2)\Big)_{1 \leqslant i,j \leqslant n}. \tag{7}$$

and assume

$$p = \inf_{i \sim j} \bar{A}_{i,j} > q = \sup_{i \not\sim j} \bar{A}_{i,j}. \tag{8}$$

Then, the cluster matrix $\bar{Z}$ defined by Eq. (2) is the unique solution of

$$\textit{maximize} \quad \langle \bar{A}, Z \rangle \quad \textit{subject to} \quad Z \in \mathcal{M}_{opt} \tag{9}$$

with $\lambda_0 = \sum_{k=1}^K n_k^2$.

The intuition behind condition (8) is that the average distance (or more precisely the average affinity) between two points within a same cluster is smaller than the average distance between two points from different clusters. This corresponds to the intuitive notion of clusters. Note that a similar condition appears in [10]. In the case of the Gaussian affinity function (4), we provide in Section 2.5 explicit formulas for the expected affinity matrix that can be used to check condition (8).

The SDP (5) appears as an approximation of the SDP (9) since the affinity matrix A can be seen as a noisy observation of the unobserved matrix $\bar{A}$. Concentration arguments together with Grothendieck theorem allow to prove that $A \approx \bar{A}$ in the sense of the ℓ^∞/ℓ^1-norm (see Proposition 1 below). In turn, this implies $\widehat{Z} \approx \bar{Z}$ in the sense of ℓ^1-norm in $\mathbb{R}^{n^2}$ (see Theorem 1 below) so that the SDP program (5) provides a good approximation $\widehat{Z}$ of the cluster matrix $\bar{Z}$. Note that in practice, λ_0 is unknown and must be estimated, see comment in section 5.1.5.

Proof (Proof of Lemma 1) This corresponds to Lemma 7.1 in [10] and is proved simply as follows. Since $\mathcal{M}_{opt} \subset [0,1]^{n \times n}$, we can transform the SDP problem (9) into the simpler problem

$$\text{maximize} \quad \langle \bar{A}, Z \rangle \quad \text{subject to} \quad Z \in [0,1]^{n \times n} \text{ and } \langle Z, 1_{n \times n} \rangle = \lambda_0.$$

In order to solve this second problem, the mass λ_0 has to be assigned to the λ_0 entries where $\bar{A}_{ij}$ is maximal. Thanks to (8), this corresponds exactly to the cluster matrix $\bar{Z}$. One can then check a posteriori that $\bar{Z} \in \mathcal{M}_{opt}$ so that in fact the original SDP problem (9) has been solved. □

2.4 Theoretical error bounds

Our main result is a non asymptotic upper bound for the probability that $\widehat{Z}$ differs from $\bar{Z}$ in ℓ^1-distance, that is an upper bound for

$$\left\| \widehat{Z} - \bar{Z} \right\|_1 = \sum_{1 \leqslant i,j \leqslant n} |\widehat{Z}_{i,j} - \bar{Z}_{i,j}|.$$

Theorem 1 *Consider the Gaussian Cluster Model (1) and assume that the affinity function f is ℓ-Lipschitz and that condition (8) is satisfied. Let*

$$t_0 = 8\sqrt{2 \log 2} K_G \sigma \ell / (p - q)$$

where $K_G \leqslant 1.8$ denotes the Grothendieck constant and $\sigma^2 = \frac{1}{n} \sum_{k=1}^K n_k \rho(\Sigma_k)$ with $\rho(\Sigma_k)$ the largest eigenvalue of the covariance matrix Σ_k.
Then, for all $t > t_0$,

$$\mathrm{P}\left(\left\| \widehat{Z} - \bar{Z} \right\|_1 > n^2 t \right) \leqslant 2 \exp\left(- \left(\frac{t - t_0}{c} \right)^2 n \right), \quad c = \frac{16\sqrt{2} K_G \ell \sigma}{p - q}. \tag{10}$$

Moreover, there exists a subset $\tau \subset \{1, \dots, n\}$ with $|\tau| \geqslant \frac{n}{2}$ such that all $t > t_0$,

$$\mathrm{P}\left(\left\| \left(\widehat{Z} - \bar{Z} \right)_{\tau \times \tau} \right\|_1 > nt \right) \leqslant 2 \exp\left(- \left(\frac{t - t_0}{c} \right)^2 n \right), \quad c = \frac{16\sqrt{2} K_G \ell \sigma}{p - q}. \tag{11}$$

Theorem 1 has a simple consequence in terms of estimation error rate. After computing $\widehat{Z}$, it is natural to estimate the cluster graph $\bar{Z}$ by a random graph obtained by putting an edge between vertices i and j if $\widehat{Z}_{i,j} > 1/2$ and no edge otherwise. Then the proportion π_n of errors in the prediction of the $n(n-1)/2$ edges is given by

$$\begin{aligned}\pi_n &:= \frac{2}{n(n-1)} \sum_{1 \leqslant i < j \leqslant n} |1_{\{\widehat{Z}_{ij} > 1/2\}} - \bar{Z}_{ij}| \\ &\leqslant \frac{2}{n(n-1)} \sum_{1 \leqslant i < j \leqslant n} 2\, |\widehat{Z}_{ij} - \bar{Z}_{ij}| = \frac{2}{n(n-1)} \left\| \widehat{Z} - \bar{Z} \right\|_1 .\end{aligned}$$

The following corollary provides a simple bound for the asymptotic error.

Corollary 1 *We have almost surely*

$$\limsup_{n \to \infty} n^{-2} \left\| \widehat{Z} - \bar{Z} \right\|_1 \leqslant t_0 = \frac{8\sqrt{2 \log 2} K_G \sigma \ell}{p - q}.$$

In the case when the cluster means are pairwise different and fixed while the cluster variances converge to 0, i.e. $\sigma \to 0$, it is easily seen that the right hand side of the above inequality behaves as $O(\sigma)$ so that the error rate converges to 0. This reflects the fact that when all clusters concentrates around their means, clustering becomes trivial.

While our proof of Theorem 1 follows the ideas from Vershynin and Guédon [10], we need to introduce new tools to justify the approximation $A \approx \bar{A}$ in ℓ^∞/ℓ^1-norm. Indeed, unlike in the stochastic block model, the entries of the affinity matrix (3) are not independent. We use Gaussian concentration measure arguments to obtain the following concentration inequality. The ℓ^∞/ℓ^1 norm of a matrix $M \in \mathbb{R}^{n \times n}$ is defined by

$$\|M\|_{\infty \to 1} = \sup_{\|u\|_\infty \leqslant 1} \|Au\|_1 = \max_{u,\ v \in \{-1,1\}^n} \sum_{i,j=1}^{n} u_i v_j M_{i,j}. \tag{12}$$

Proposition 1 *Consider the Gaussian mixture model* (1) *and assume the affinity function f is ℓ-Lipschitz. Then, for any $t > 2\ \sqrt{2 \log 2}\ \ell\ \sigma$,*

$$\mathrm{P}\Big(\left\| A - \bar{A} \right\|_{\infty \to 1} > t\ n^2 \Big) \leqslant 2 \exp\left(- \frac{\big(t - 2\sqrt{2 \log 2} \ell \sigma\big)^2}{32 \ell^2 \sigma^2}\ n \right). \tag{13}$$

Theorem 1 assumes that λ_0 is known. It is worth noting that λ_0 corresponds to the number of edges in the cluster graph and that we can derive from the proof of Theorem 1 how the algorithm behaves when the cluster sizes are unknown, i.e. when the unknown parameter λ_0 is replaced by a different value λ. The intuition is given in [10, Remark 1.6]: if $\lambda < \lambda_0$, the solution $\widehat{Z}$ will estimate a certain subgraph of the cluster graph with at most $\lambda_0 - \lambda$ missing edges; if $\lambda > \lambda_0$, the solution $\widehat{Z}$ will estimate a certain supergraph of the cluster graph with at most $\lambda - \lambda_0$ extra-edges.

2.5 Explicit formula for $\bar{A}$

In order to check condition (8), explicit formulas for the mean affinity matrix are useful. The next proposition solves the case of the Gaussian affinity function.

Proposition 2 *Assume that A is build using the Gaussian affinity function* (4).

- *Let i and j be in the same cluster $\mathcal{C}_k$. Then,*

$$\bar{A}_{i,j} = \prod_{l=1}^{d} \left(1 + 4(\sigma_{k,l}/h_0)^2\right)^{-1/2}$$

with $(\sigma_{k,l}^2)_{1 \leqslant l \leqslant d}$ the eigenvalues of Σ_k.

– *Let i and j be in different clusters $\mathcal{C}_k$ and $\mathcal{C}_{k'}$. Then,*

$$\bar{A}_{i,j} = \prod_{l=1}^{d} \exp\left(-\frac{\langle \mu_k - \mu_{k'}, v_{k,k',l}\rangle^2}{h_0^2 + 2\sigma_{k,k',l}^2}\right)\left(1 + 2(\sigma_{k,k',l}/h_0)^2\right)^{-1/2}$$

with $(\sigma_{k,k',l}^2)_{1\leqslant l\leqslant d}$ and $(v_{k,k',l})_{1\leqslant l\leqslant d}$ respectively the eigenvalues and eigenvectors of $\Sigma_k + \Sigma_{k'}$.

As an interesting consequence of Proposition 2, when the variance matrices from the Gaussian Cluster Model (1) are all equal and isotropic, that is $\Sigma_k = \sigma^2 \mathrm{Id}$ for all $k = 1, \ldots, K$ with $\sigma^2 > 0$, then the constants p and q from Equation (8) are given by

$$p = \left(1 + 4\sigma^2/h_0^2\right)^{-d/2} \quad \text{and} \quad q = \left(1 + 4\sigma^2/h_0^2\right)^{-d/2} \min_{1\leqslant k\neq k'\leqslant K} \exp\left\{-\|\mu_{k'} - \mu_k\|^2/(h_0^2 + 4\sigma^2)\right\}.$$

Condition (8) is therefore satisfied (whatever the choice of $h_0 > 0$) as soon as the cluster means $(\mu_k)_{1\leqslant k\leqslant K}$ are pairwise distinct which is a minimal identifiability condition. But of course the difference $p - q$ is an increasing function of the noise σ^2 and the bounds in Theorem 1 become looser for larger noise.

Proof The proof of the proposition relies on the fact that $X_i - X_j$ is a Gaussian random vector with mean $\mu_{k_i} - \mu_{k_j}$ and variance $\Sigma_{k_i} + \Sigma_{k_i}$ so that the distribution of $\|X_i - X_j\|_2^2$ is related to the noncentral χ^2 distribution with p degrees of freedom. The quantity $\mathbb{E}[\exp(-\|X_i - X_j\|_2^2/h_0)]$ corresponds to the Laplace transform of the noncentral χ^2 distribution explicited in the next lemma. □

Lemma 2 *Let $X \sim \mathcal{N}(\mu, \Sigma)$. If $\mu = 0$, we have*

$$\mathrm{E}\left[e^{t\|X\|^2}\right] = \prod_{d=1}^{p} \left(1 - 2t\sigma_d^2\right)^{-1/2}, \quad t \leqslant 0,$$

with $\sigma_1^2, \ldots, \sigma_p^2$ the eigenvalues of Σ. More generally, for $\mu \neq 0$,

$$\mathrm{E}\left[e^{t\|X\|^2}\right] = \prod_{d=1}^{p} \exp\left(\frac{\langle \mu, v_d\rangle^2 t}{1 - 2t\sigma_d^2}\right)\left(1 - 2t\sigma_d^2\right)^{-1/2}$$

with $\sigma_1^2, \ldots, \sigma_p^2$ the eigenvalues of Σ and $v_1, \ldots, v_p$ the associated eigenvectors.

3 Proofs

3.1 Proof of Proposition 1

Proof The concentration of the affinity matrix A around its mean $\bar{A}$ follows from concentration inequalities for Lipschitz function of independent standard Gaussian variables, see Appendix A. From definition (12)

$$\|A - \bar{A}\|_{\mathscr{L}\to 1} = \max_{u,v\in\{-1,1\}^n} F_{uv} \quad \text{with} \quad F_{uv} = \sum_{i,j=1}^{n} u_i v_j (A_{i,j} - \bar{A}_{i,j}). \tag{14}$$

We introduce the standardized observations: if x_i is in cluster $\mathcal{C}_{k_i}$, i.e. $x_i \sim \mathcal{N}(\mu_{k_i}, \Sigma_{k_i})$, then $y_i = \Sigma_{k_i}^{-1/2}(x_i - \mu_{k_i})$, $1 \leqslant i \leqslant n$ are independent identically distributed random variables with standard Gaussian distribution. In view of definition (14), the random variables F_{uv} can be expressed in terms of the standardized observations

$$F_{uv}(y_1, \ldots, y_n) = 2 \sum_{1\leqslant i<j\leqslant n} u_i v_j \left[f\left(\left\|\Sigma_{k_j}^{1/2} y_j - \Sigma_{k_i}^{1/2} y_i + \mu_{k_j} - \mu_{k_i}\right\|_2\right) - \bar{A}_{i,j})\right].$$

We prove next that the function $F_{uv} : \mathbb{R}^{p\times n} \to \mathbb{R}$ is L-Lipschitz with $L = 2\ell\sigma n^{3/2}$. Indeed, for $(y_1,\dots,y_n),(y_1',\dots,y_n') \in \mathbb{R}^{p\times n}$, we have

$$\begin{aligned}
\left|F_{uv}(y_1,\dots,y_n) - F_{uv}(y_1',\dots,y_n')\right| &\leqslant \ell \sum_{1\leqslant i\neq j\leqslant n} \|x_i - x_i'\|_2 + \|x_j - x_j'\|_2 \\
&= 2(n-1)\ell \sum_{i=1}^n \|\Sigma_{k_i}^{1/2}(y_i - y_i')\|_2 \\
&\leqslant 2n\ell \sum_{i=1}^n \rho(\Sigma_{k_i})^{1/2}\|y_i - y_i'\|_2 \\
&\leqslant 2\ell\sigma n^{3/2}\|(y_1,\dots,y_n) - (y_1',\dots,y_n')\|_2.
\end{aligned}$$

In the first inequality, we use the fact that f is ℓ-Lipschitz. The second inequality relies on the fact that all the eigenvalues of $\Sigma_{k_i}^{1/2}$ are smaller that $\rho(\Sigma_{k_i})$. The last inequality relies on Cauchy-Schwartz inequality and on the definition $\sigma^2 = \frac{1}{n}\sum_{i=1}^n \max_{1\leqslant k\leqslant K}\rho(\Sigma_{k_i})$.

Thanks to this Lipschitz property, the Tsirelson-Ibragimov-Sudakov inequality (Theorem 2 in the Appendix) implies

$$\mathrm{E}\left[\exp(\theta F_{uv})\right] \leqslant \exp\left(L^2\theta^2/2\right) \quad \text{for all} \quad \theta \in \mathbb{R}$$

and we deduce from Theorem 3 that

$$\begin{aligned}
\mathrm{E}\left[\|A - \bar{A}\|_{\infty\to 1}\right] &= \mathrm{E}\left[\max_{u,v\in\{-1,1\}^n} F_{uv}\right] \\
&\leqslant \sqrt{2L^2\log 2^n} = 2\sqrt{2\log 2}\ell\sigma n^2.
\end{aligned}$$

On the other hand, the function $\max_{u,v\in\{-1,1\}^n} F_{uv}$ is also L-Lipschitz and Theorem 2 implies

$$\begin{aligned}
\mathrm{P}\left(|\|A - \bar{A}\|_{\infty\to 1} - \mathrm{E}\|A - \bar{A}\|_{\infty\to 1}| > t\right) &= \mathrm{P}\left(|\max_{u,v\in\{-1,1\}^n} F_{uv} - \mathrm{E}\max_{u,v\in\{-1,1\}^n} F_{uv}| > t\right) \\
&\leqslant 2\exp\left(-\frac{t^2}{8L^2}\right).
\end{aligned}$$

Combining these different estimates, we obtain for $t > 2\sqrt{2\log 2}\ell\sigma$,

$$\begin{aligned}
&\mathrm{P}(\|A - \bar{A}\|_{\infty\to 1} > tn^2) \\
&\leqslant \mathrm{P}\left(|\|A - \bar{A}\|_{\infty\to 1} - \mathrm{E}\|A - \bar{A}\|_{\infty\to 1}| > (t - 2\sqrt{2\log 2}\ell\sigma)n^2)\right) \\
&\leqslant 2\exp\left(-\frac{\left(t - 2\sqrt{2\log 2}\ell\sigma\right)^2}{32\ell^2\sigma^2}n\right).
\end{aligned}$$

□

3.2 Proof of Theorem 1

The proof follows the same lines as in Guédon and Vershynin [10] and we provide the main ideas for the sake of completeness. The proof is divided into 4 steps.

Step 1: we prove that

$$\langle \bar{A}, \bar{Z}\rangle - 2K_G\|A - \bar{A}\|_{\infty\to 1} \leqslant \langle \bar{A}, \hat{Z}\rangle \leqslant \langle \bar{A}, \bar{Z}\rangle \tag{15}$$

with K_G denoting Grothendieck's constant.
The upper bound follows directly from Lemma 1. For the lower bound, we use the definition of $\hat{Z}$ as a maximizer and write

$$\begin{aligned}
\langle \bar{A}, \hat{Z}\rangle &= \langle A, \hat{Z}\rangle + \langle \bar{A} - A, \hat{Z}\rangle \\
&\geqslant \langle A, \bar{Z}\rangle - \langle A - \bar{A}, \hat{Z}\rangle \\
&= \langle \bar{A}, \bar{Z}\rangle + \langle A - \bar{A}, \bar{Z}\rangle - \langle A - \bar{A}, \hat{Z}\rangle.
\end{aligned}$$

Grothendieck's inequality implies that for every $Z \in \mathcal{M}_{opt}$,

$$\left|\langle A - \bar{A}, \hat{Z}\rangle\right| \leqslant K_G \|A - \bar{A}\|_{\infty\to 1}.$$

See Theorem 4 and Lemma 3 in the Appendix. Using this, we get

$$2K_G \|A - \bar{A}\|_{\infty\to 1} \geqslant \langle \bar{A}, \hat{Z} - \bar{Z}\rangle. \tag{16}$$

Step 2: we show that for every $Z \in \mathcal{M}_{opt}$,

$$\langle \bar{A}, \bar{Z} - Z\rangle \geqslant \frac{p-q}{2} \left\|\bar{Z} - Z\right\|_1. \tag{17}$$

This corresponds to Lemma 7.2 in [10] and shows that the expected objective function distinguishes points. Introducing the set

$$\mathrm{In} = \cup_{k=1}^{K} \mathcal{C}_k \times \mathcal{C}_k \tag{18}$$

of edges within clusters and the set

$$\mathrm{Out} = \{1, \ldots, n\}^2 \backslash \mathrm{In} \tag{19}$$

of edges across clusters, we decompose the scalar product

$$\langle \bar{A}, \bar{Z} - Z\rangle = \sum_{(i,j)\in \mathrm{In}} \bar{A}_{ij}(\bar{Z}_{ij} - Z_{ij}) - \sum_{(i,j)\in \mathrm{Out}} \bar{A}_{ij}(Z_{ij} - \bar{Z}_{ij}).$$

Note that the definition of the cluster matrix (2) implies that $\bar{Z}_{ij} - Z_{ij} \geqslant 0$ if $(i,j) \in \mathrm{In}$ and $\bar{Z}_{ij} - Z_{ij} \leqslant 0$ if $(i,j) \in \mathrm{Out}$. This together with condition (8) implies

$$\langle \bar{A}, \bar{Z} - Z\rangle \geqslant p \sum_{(i,j)\in \mathrm{In}} (\bar{Z} - Z)_{ij} - q \sum_{(i,j)\in \mathrm{Out}} (Z - \bar{Z})_{ij}.$$

Introduce $S_{\mathrm{In}} = \sum_{(i,j)\in \mathrm{In}} (\bar{Z} - Z)_{ij}$ and $S_{\mathrm{Out}} = \sum_{(i,j)\in \mathrm{Out}} (\bar{Z} - Z)_{ij}$. Since $\langle \bar{Z}, 1_{n\times n}\rangle = \langle Z, 1_{n\times n}\rangle = \lambda_0$, we have $S_{\mathrm{In}} - S_{\mathrm{Out}} = 0$. On the other hand $S_{\mathrm{In}} + S_{\mathrm{Out}} = \left\|\bar{Z} - Z\right\|_1$. We deduce exact expressions for S_{In} and S_{Out} and the lower bound

$$\langle \bar{A}, \bar{Z} - Z\rangle \geqslant \frac{p-q}{2} \left\|\bar{Z} - Z\right\|_1. \tag{20}$$

Combining (16), (20) and (24), we obtain

$$\left\|\bar{Z} - \hat{Z}\right\|_1 \leqslant \frac{1}{n} \frac{4K_G}{p-q} \|A - \bar{A}\|_{\infty\to 1}. \tag{21}$$

Step 3: we prove (10).
From (21) we get

$$\mathrm{P}\left(\left\|Z - \bar{Z}\right\|_1 > t\, n^2\right) \leqslant \mathrm{P}\left(\left\|A - \bar{A}\right\|_{\infty\to 1} > t\, \frac{p-q}{4K_G}\, n^2\right).$$

and (10) follows then directly from (13).

Step 4: we finally prove (11).
For every matrix $H \in \mathbb{R}^{n\times n}$, we have

$$\|G\|_1 \geqslant \|G\|_{\infty\to 1}. \tag{22}$$

From Proposition 5.2 in [24], there exists a subset $\tau \in \{1, \ldots, n\}$ such that $|\tau| \geqslant \frac{n}{2}$ and

$$\|H_{\tau\times\tau}\|_1 \leqslant \frac{2K_G}{n} \|H\|_{\infty\to 1}.$$

Therefore, taking $H = \bar{Z} - \hat{Z}$, we get

$$\left\| \left(\bar{Z} - \hat{Z}\right)_{\tau\times\tau} \right\|_1 \leqslant \frac{2K_G}{n} \left\| \left(\bar{Z} - \hat{Z}\right)_{\tau\times\tau} \right\|_{\infty\to 1} \tag{23}$$

and Equation (22) entails

$$\left\| \left(\bar{Z} - \hat{Z}\right)_{\tau\times\tau} \right\|_1 \leqslant \frac{2K_G}{n} \left\| \left(\bar{Z} - \hat{Z}\right)_{\tau\times\tau} \right\|_1. \tag{24}$$

Combining this last equation with (21), we obtain

$$\left\| \left(\bar{Z} - \hat{Z}\right)_{\tau\times\tau} \right\|_1 \leqslant \frac{1}{n}\frac{8K_G^2}{p-q} \left\| A - \bar{A} \right\|_{\infty\to 1}.$$

We thus may deduce that

$$\mathrm{P}\left(\left\| \left(\bar{Z} - \hat{Z}\right)_{\tau\times\tau} \right\|_1 > t\, n^2 \right) \leqslant \mathrm{P}\left(\left\| A - \bar{A} \right\|_{\infty\to 1} > t\, \frac{p-q}{4K_G}\, n^2 \right).$$

and (11) follows then directly from (13).

4 Solving the SDP

4.1 Helmberg and Rendl's spectral formulation

We first rewrite the equality constraint from (6). The constraints that every diagonal element of Z should be equal to 1 can be written as

$$\mathrm{Tr}\left(C_i Z\right) = 1, \quad C_i = e_i e_i^T, \quad i = 1, \ldots n. \tag{25}$$

The constraint $\sum_{i,j=1}^{n} Z_{i,j} = \lambda$ can be expressed as the rank-1 constraint

$$\mathrm{Tr}\left(DZ\right) = \lambda, \tag{26}$$

where D is the all-ones matrix of size $n \times n$.

One of the nice features of the Semi-Definite Program (5)-(6) is that it can be rewritten as an eigenvalue optimization problem. Let us adopt a Lagrangian duality approach to this problem. As in Helmberg and Rendl [12], we will impose the reduntant constraint that the trace is constant. The Lagrange function is given by

$$\begin{aligned} L(Z, z) &= \langle A, Z\rangle + \sum_{i=1}^{n} z_i \ \left(\langle C_i, Z\rangle - 1\right) + z_{n+1}\left(\langle D, Z\rangle - \lambda\right) \\ &= \left\langle A + \sum_{i=1}^{n} z_i C_i + z_{n+1} D, Z \right\rangle - \sum_{i=1}^{n} z_i - \lambda z_{n+1}. \end{aligned}$$

Now the dual function is given by

$$\theta(z) = \max_{\substack{Z\succeq 0 \\ \mathrm{trace}(Z)=n}} \quad L(Z, z). \tag{27}$$

Therefore, we easily get that

$$\theta(z) = n\ \lambda_{\max}\left(A + \sum_{i=1}^{n} z_i C_i + z_{n+1} D \right) - \sum_{i=1}^{n} z_i - \lambda z_{n+1}$$

where $\lambda_{\max}$ is the maximum eigenvalue function. Therefore, the solution to this problem is amenable to eigenvalue optimization. An important remark is that a maximizer Z^* in (27) associated to a dual minimizer z^* will be a solution of the original problem. It can be written as

$$Z^* = V^* V^{*^t}$$

where V^* is a matrix whose columns form a basis of the eigenspace associated with $\lambda_{\max}\left(A + \sum_{i=1}^{n} z_i C_i + z_{n+1} D\right)$.

4.2 Using the HANSO algorithm

We now introduce the HANSO algorithm [20]. Note that the maximum eigenvalue is a convex function and denote by $\mathcal{A}$ the affine operator

$$\mathcal{A}(z) = A + \sum_{i=1}^{n} z_i C_i + z_{n+1} D.$$

The subdifferential of $\lambda_{\max}(\mathcal{A}(z))$ is given by

$$\partial\lambda_{\max}(\mathcal{A}(z)) = \mathcal{A}^* \left(V \mathcal{Z} V^t\right)$$

where V is a matrix whose columns form a basis for the eigenspace associated to the maximum eigenvalue of $\mathcal{A}(z)$ and

$$\mathcal{Z} = \left\{ Z \in \mathbb{R}^{r_{\max} \times r_{\max}} \mid Z \succeq 0 \text{ and } \operatorname{trace}(Z) = 1 \right\}$$

with $r_{\max}$ the multiplicity of this maximum eigenvalue.

Using this, it is easy to minimize the dual function θ. Indeed, the sub-differential of θ at z is simply given by

$$\partial\theta(z) = \mathcal{A}^* \left(V \mathcal{Z} V^t\right) - \left\{ \begin{bmatrix} e \\ \lambda \end{bmatrix} \right\}.$$

The algorithm HANSO [20] can then be used to perform the actual minimization of the dual function θ. A primal solution can then be recovered as a maximizer in the definition (27) of the dual function.

4.3 Computing the actual clustering

As advised in [26], the actual clustering can be computed using a minimum spanning tree method and removing the largest edges.

4.4 Choosing λ

Once the clusters have been identified, it is quite easy to identify the underlying Gaussian Mixture Distribution. The choice of λ can then be performed using standard model selection criteria such as AIC [2], BIC [22] or ICL [5].

Another approach is to simply choose the value of λ than minimises the distance to the observed affinity matrix. This is the choice we have made in the subsequent experiments.

5 Simulation results

In all the experiments, the parameter h_0 in (4) was chosen as

$$h_0 = .5 * \max(\operatorname{diag}(X^t * X))^{1/2}.$$

The hyper-parameter λ was chosen so as to minimise the mean squared error between the estimated cluster matrix and the empirical affinity matrix.

5.1 Some interesting examples

In this section, we start with three examples where the separation properties of the Guedon-Vershynin embedding are nicely illustrated. In all three instances, the data were generated using two 2- dimensional Gaussian samples with equal size (100 samples by cluster). These examples are shown in Figure 2, Figure 3 and Figure 4 below. All these example seem to be very difficult to address for methods with guaranteed polynomial time convergence. In each case, one observes that the clusters are well separated after the embedding and that they do not look like Gaussian samples anymore. The fact of not being Gaussian does not impair the success of methods such as minimum spanning trees although such methods might work better with the example in Figure 4 than in the example of Figure 2 and Figure 3.

5.2 Comparison with standard embeddings on a 3D cluster example

Simulations have been conducted to assess the quality of the proposed embedding. In this subsection, we used the Matlab package drtoolbox https://lvdmaaten.github.io/drtoolbox/ proposed by Laurens Van Maatten on a sample drawn from a 10 dimensional Gaussian Mixture Model with 4 components and equal proportions. In Figure 5, we show the original affinity matrix together with the estimated cluster matrix. In Figure 6, we compare the affinity matrix of data with the affinity matrix of the mapped data using various embeddings proposed in the drtoolbox package. This toy experiment shows that the embedding described in this paper can cluster as the same time as is embeds into a small dimensional subspace. This is not very surprising since our embedding is tailored for the joint clustering-dimensionality reduction purpose whereas most of the known existing embedding methods aren't. Given the fact that clustered data are ubiquitous in real world data analysis due to the omnipresence of stratified populations, taking the clustering purpose into account might be a considerable advantage.

5.3 Monte Carlo Experiments

In this section, we present some simulation experiments assessing the performance of the Guedon-Vershynin embedding for Gaussian Mixture Models.

Our experiments were performed on problems of successive sample size 100, 200, ..., 1000 and number of clusters equal to 2, 5 and 10. The dimension of the Gaussian Mixture Model was set to 100. For each experiment, we performed 100 Monte Carlo repeats. All the results in this section show the average over the Monte Carlo experiments. Our Gaussian Mixture Model was built as follows: for a model with K clusters, we set the k^{th} component of each center to 10/3, 20/3, ..., 50/3 for each cluster $k = 1, \ldots, K$. Then, the data are obtained by adding a unit variance i.i.d. Gaussian vector to the center of the cluster it belongs to. All clusters were taken to have equal size.

5.3.1 Error as a function of the inter-cluster distance and sample size

Figures 7, 8 and 9 show the estimation error $\|\bar{Z} - \hat{Z}\|_1$ between the true and the estimated cluster matrix. Theses results illustrate Theorem 1 as it show that the error grows as a function of sample size. Moreover, the grows is quadratic as predicted by the theory. The error increases as a function of level which is the value of the nonzero component of the cluster's centers.

5.3.2 Rank recovery

As remarked in Section the rank of the true cluster matrix $\bar{Z}$ should be equal to the number of clusters. The rank of the estimated cluster matrix $\hat{Z}$ should therefore be close to the number of clusters if the method is working properly.

The spectrum of the estimated cluster matrix $\hat{Z}$ for one instance of the Monte-Carlo experiment with 10 clusters is displayed in Figure 10 below. We see form this picture that the spectrum is almost sparse and that there is a clear jump afer rank 10. The jump between rank 2 and rank 10 shows that exact recovery of the number of clusters K when K is large may be difficult.

Estimation of the rank was done as follows: the estimated rank was taken as the value on the abscissa for which the second finite successive differences in the spectrum was the largest. Figure 11 below shows the histogram for component level 20/3 and Figure 12 shows the histogram for level 40/3

These histograms show that the rank is well recovered for small number of clusters but rank recovery start degrading as the number of clusters is too large.

6 Conclusions

The goal of the present paper was to propose an analysis of Guedon and Vershynin's Semi-Definite Programming approach to the estimation of the cluster matrix and show how this matrix can be used to produce an embedding for preconditioning standard clustering procedures. The procedure is suitable for very high dimensional data because it is based on pairwise distances only. Moreover, increasing the dimension will improve the robustness of the procedure when the Law of Large Numbers will apply along dimensions, hence forcing the affinity matrix to converge to a deterministic limit and thus making the estimator less sensitive to its low dimensional fluctuations.

Another feature of the method is that it may apply to a large number of mixtures type, even when the component's densities are not log-concave, as do a lot of embeddings as applied to data concentrated on complicated manifolds. Further studies will be performed in this exciting direction.

Future work is also needed for proving that the proposed embedding is provably efficient when combined with various clustering techniques. One of the main reason why this should be a difficult problem is that the approximation bound proved in the present paper is not so easy to leverage for controlling the perturbation of the eigenspaces of Z. More precise use of the inherent randomness of the perturbation, in the spirit of [26], might be necessary in order to go a little further in this direction.

A Concentration inequalities

The following inequality is a particular case of the Log-Sobolev concentration inequality, see Theorems 5.5 and 5.6. in [7].

Theorem 2 (Gaussian concentration inequality) *Let $Y_1, \dots, Y_n$ be independent Gaussian random vectors on $\mathbb{R}^p$ with mean 0 and variance I_p. Assume that $F : \mathbb{R}^{n\times p} \to \mathbb{R}$ is Lipschitz with constant L, i.e.*

$$|F(y') - F(y)| \leqslant L\|y' - y\|_2 \quad \text{for all } y, y' \in \mathbb{R}^{n\times p}.$$

Then the random variable $F = F(Y_1, \dots, Y_n)$ satisfies

$$\mathrm{E}[\exp(\theta\,(F - \mathrm{E}F))] \leqslant \exp(L^2\theta^2/2) \quad \text{for all } \theta \in \mathbb{R}$$

and also

$$\mathrm{P}(|F - \mathrm{E}F| > t) \leqslant 2\exp\left(-t^2/(8L^2)\right) \quad \text{for all } t > 0.$$

The next theorem provides result for the expected maxima of (non necessarily independent) subgaussian random variables.

Theorem 3 *Let $Z_1, \cdots, Z_N$ be real valued sub-Gaussian random variables with variance factor ν, i.e. satisfying*

$$\mathrm{E}[\exp(\theta Z_i)] \leqslant \exp(\nu\theta^2/2) \quad \text{for all } \theta \in \mathbb{R}.$$

Then

$$\mathrm{E}\left[\max_{i=1,\cdots,N} Z_i\right] \leqslant \sqrt{2\nu \log N}.$$

B The Grothendieck inequality

In this paper, we use the following matrix version of Grothendieck inequality. We denote by $\mathcal{M}_G$ the set of matrices $Z = XY^T$ with $X, Y \in \mathbb{R}^{n\times n}$ having all raws in the unit Euclidean ball, i.e.

$$\forall i \in \{1, \dots, n\}, \quad \sum_{j=1}^{n} X_{ij}^2 \leqslant 1 \quad \text{and} \quad \sum_{j=1}^{n} Y_{ij}^2 \leqslant 1$$

Theorem 4 (Grothendieck inequality) *There exists an universal constant K_G such that every matrix $B \in \mathbb{R}^{n\times n}$ satisfies*

$$\max_{Z\in\mathcal{M}_G} |\langle B, Z\rangle| \leqslant K_G \|B\|_{\infty\to 1}$$

where the $\ell^\infty \to \ell^1$ norm of B is defined by (12).

It is also useful to note the following properties of $\mathcal{M}_G$, see Lemma 3.3 in [10].

Lemma 3 *Every matrix $Z \in \mathbb{R}^{n\times n}$ such that $Z \succeq 0$ and $\mathrm{diag}(Z) \leqslant 1_n$ satisfies $Z \in \mathcal{M}_G$.*

References

1. Emmanuel Abbe, Afonso S Bandeira, and Georgina Hall, *Exact recovery in the stochastic block model*, arXiv preprint arXiv:1405.3267 (2014).
2. Hirotugu Akaike, *A new look at the statistical model identification*, Automatic Control, IEEE Transactions on **19** (1974), no. 6, 716–723.
3. Afonso S Bandeira, *Ten lectures and forty-two open problems in the mathematics of data science*, (2015).
4. Mikhail Belkin and Partha Niyogi, *Laplacian eigenmaps and spectral techniques for embedding and clustering.*, NIPS, vol. 14, 2001, pp. 585–591.
5. Christophe Biernacki, Gilles Celeux, and Gérard Govaert, *Assessing a mixture model for clustering with the integrated completed likelihood*, Pattern Analysis and Machine Intelligence, IEEE Transactions on **22** (2000), no. 7, 719–725.
6. Christophe Biernacki and Stéphane Chrétien, *Degeneracy in the maximum likelihood estimation of univariate gaussian mixtures with em*, Statistics & probability letters **61** (2003), no. 4, 373–382.
7. Stéphane Boucheron, Gábor Lugosi, and Pascal Massart, *Concentration inequalities*, Oxford University Press, Oxford, 2013, A nonasymptotic theory of independence, With a foreword by Michel Ledoux. MR 3185193
8. Emmanuel J Candès, Xiaodong Li, Yi Ma, and John Wright, *Robust principal component analysis?*, Journal of the ACM (JACM) **58** (2011), no. 3, 11.
9. A. P. Dempster, N. M. Laird, and D. B. Rubin, *Maximum likelihood from incomplete data via the EM algorithm*, J. Roy. Statist. Soc. Ser. B **39** (1977), no. 1, 1–38, With discussion. MR 0501537 (58 #18858)
10. Olivier Guédon and Roman Vershynin, *Community detection in sparse networks via grothendieck's inequality*, Probability Theory and Related Fields (2015), 1–25.
11. Simon Heimlicher, Marc Lelarge, and Laurent Massoulié, *Community detection in the labelled stochastic block model*, arXiv preprint arXiv:1209.2910 (2012).
12. Christoph Helmberg and Franz Rendl, *A spectral bundle method for semidefinite programming*, SIAM Journal on Optimization **10** (2000), no. 3, 673–696.
13. Toby Dylan Hocking, Armand Joulin, Francis Bach, and Jean-Philippe Vert, *Clusterpath an algorithm for clustering using convex fusion penalties*, 28th international conference on machine learning, 2011, p. 1.
14. Anil K Jain, *Data clustering: 50 years beyond k-means*, Pattern recognition letters **31** (2010), no. 8, 651–666.
15. William B Johnson and Joram Lindenstrauss, *Extensions of lipschitz mappings into a hilbert space*, Contemporary mathematics **26** (1984), no. 189-206, 1.
16. Ian Jolliffe, *Principal component analysis*, Wiley Online Library, 2002.
17. Nathan Linial, Eran London, and Yuri Rabinovich, *The geometry of graphs and some of its algorithmic applications*, Combinatorica **15** (1995), no. 2, 215–245.
18. Geoffrey McLachlan and David Peel, *Finite mixture models*, John Wiley & Sons, 2004.
19. Elchanan Mossel, Joe Neeman, and Allan Sly, *Stochastic block models and reconstruction*, arXiv preprint arXiv:1202.1499 (2012).
20. M Overton, *Hanso: a hybrid algorithm for nonsmooth optimization*, Available from cs. nyu. edu/overton/software/hanso (2009).
21. Peter Radchenko and Gourab Mukherjee, *Consistent clustering using an ℓ_1 fusion penalty*, arXiv preprint arXiv:1412.0753 (2014).
22. Gideon Schwarz et al., *Estimating the dimension of a model*, The annals of statistics **6** (1978), no. 2, 461–464.
23. Kean Ming Tan, Daniela Witten, et al., *Statistical properties of convex clustering*, Electronic Journal of Statistics **9** (2015), no. 2, 2324–2347.
24. Joel A Tropp, *Column subset selection, matrix factorization, and eigenvalue optimization*, Proceedings of the Twentieth Annual ACM-SIAM Symposium on Discrete Algorithms, Society for Industrial and Applied Mathematics, 2009, pp. 978–986.
25. Ulrike Von Luxburg, *A tutorial on spectral clustering*, Statistics and computing **17** (2007), no. 4, 395–416.
26. Van Vu, *Singular vectors under random perturbation*, Random Structures & Algorithms **39** (2011), no. 4, 526–538.
27. Binhuan Wang, Yilong Zhang, Wei Sun, and Yixin Fang, *Sparse convex clustering*, arXiv preprint arXiv:1601.04586 (2016).
28. Kilian Q Weinberger and Lawrence K Saul, *Unsupervised learning of image manifolds by semidefinite programming*, International Journal of Computer Vision **70** (2006), no. 1, 77–90.

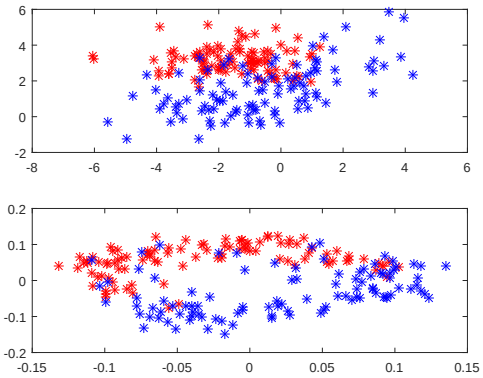


Fig. 2: Example 1

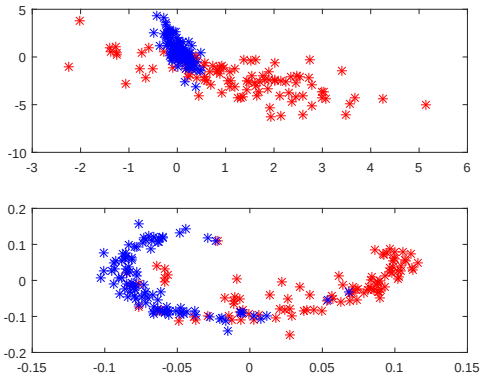


Fig. 3: Example 2

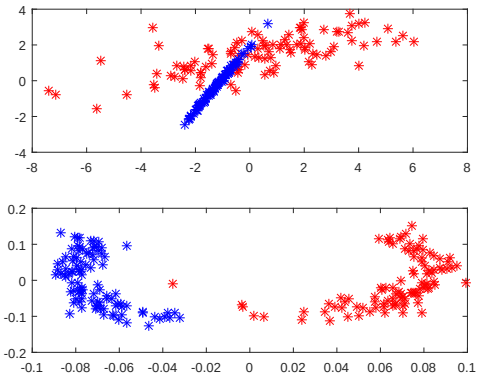


Fig. 4: Example 3

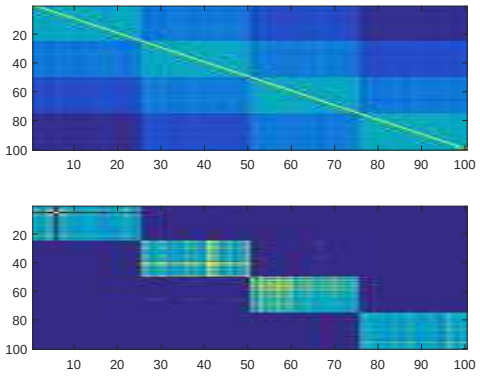
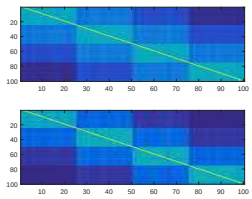
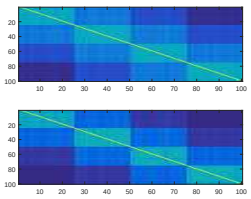


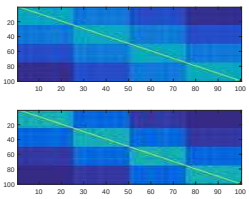
Fig. 5: Original affinity matrix vs. Guedon Vershynin Cluster matrix



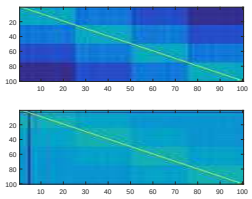
(a) Original affinity matrix vs. affinity matrix after PCA embedding



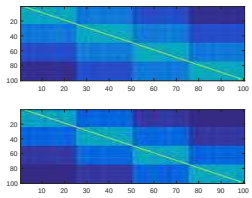
(b) Original affinity matrix vs. affinity matrix after MDS embedding



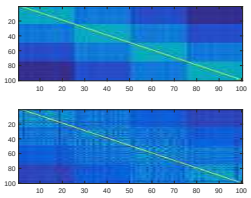
(c) Original affinity matrix vs. affinity matrix after Factor Analysis embedding



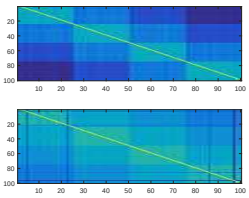
(d) Original affinity matrix vs. affinity matrix after t-SNE embedding



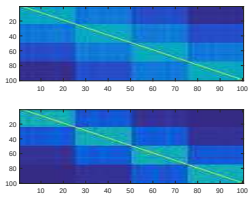
(e) Original affinity matrix vs. affinity matrix after Sammon embedding



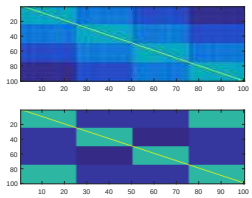
(f) Original affinity matrix vs. affinity matrix after LLE embedding



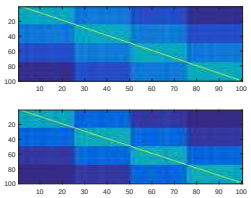
(g) Original affinity matrix vs. affinity matrix after Laplacian embedding



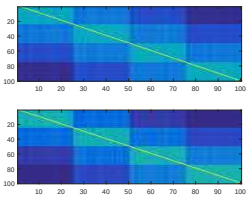
(h) Original affinity matrix vs. affinity matrix after Kernel-PCA embedding



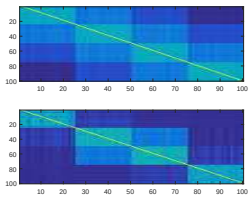
(i) Original affinity matrix vs. affinity matrix after LTSA embedding



(j) Original affinity matrix vs. affinity matrix after MVU embedding



(k) Original affinity matrix vs. affinity matrix after Auto Encoder embedding



(l) Original affinity matrix vs. affinity matrix after DiffusionMap embedding

Fig. 6: The affinity matrix obtained after embedding using different methods from the Matlab package drtoolbox https://lvdmaaten.github.io/drtoolbox/

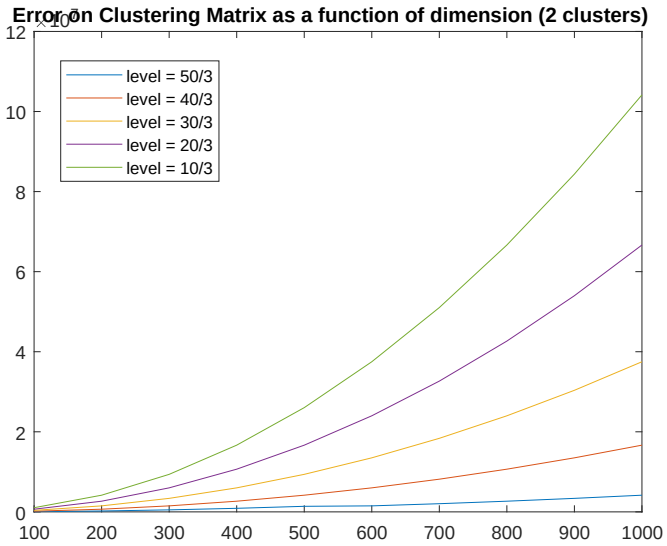


Fig. 7: Estimation error $\|\bar{Z} - \hat{Z}\|_1$

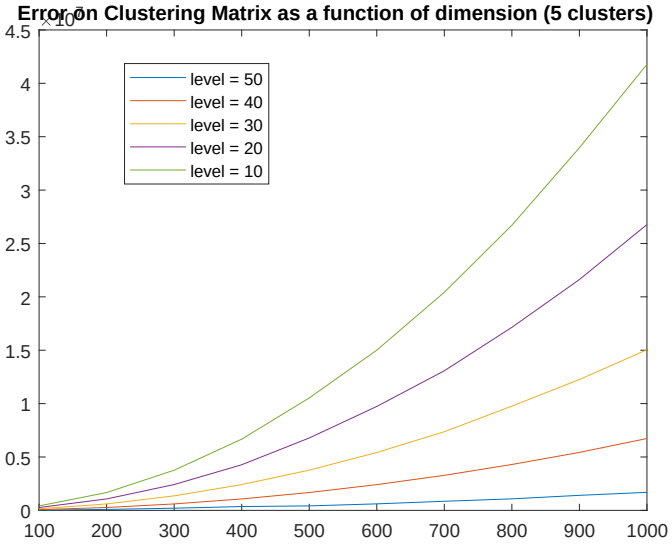


Fig. 8: Estimation error $\|\bar{Z} - \hat{Z}\|_1$

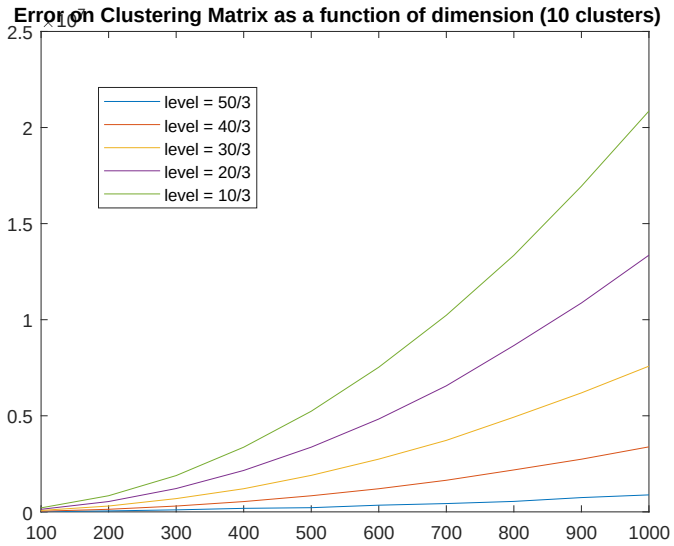


Fig. 9: Estimation error $\|\bar{Z} - \hat{Z}\|_1$

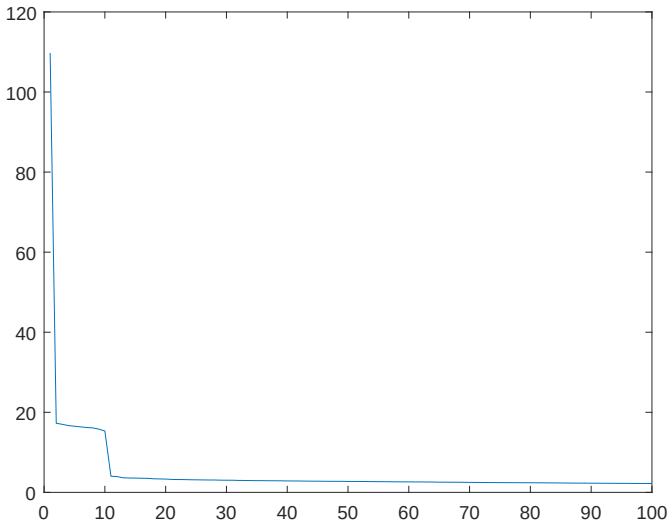


Fig. 10: Example of spectrum for $\hat{Z}$ in the case of 10 clusters

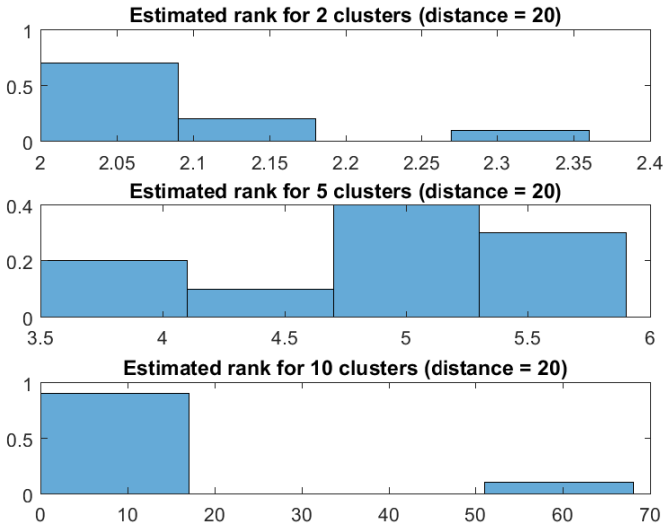


Fig. 11: Histogram of the estimated rank for 2, 5 and 10 clusters and level 20/3

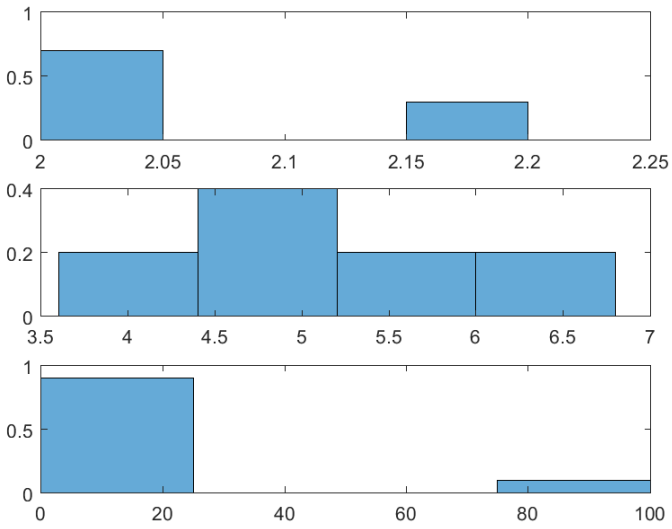


Fig. 12: Histogram of the estimated rank for 2, 5 and 10 clusters and level 40/3

4

Factorisation en matrices positives

On dresse ici un portrait des différentes techniques utilisées en pratique pour attaquer le problème de factorisation en matrices non-négatives (NMF).

4.1 Approches classiques

Nous exposons dans ce chapitre quelques méthodes classiques de calcul de NMF. A l'objectif NMF défini au premier chapitre on préférera substituer

$$\min_{W\in\mathbb{R}_+^{m\times k},H\in\mathbb{R}_+^{k\times n}} \frac{1}{2}\left\|M-WH\right\|_F^2, \tag{4.1}$$

dont le gradient s'écrit mieux.

4.1.1 Conditions de stationnarité, gradient et dualité

On commence par étudier les conditions de stationnarité d'un algorithme calculant une NMF. Les conditions de Karush-Kuhn-Tucker permettent de caractériser les points stationnaires d'un problème d'optimisation sous contrainte. Plutôt que de réintroduire ces conditions, on préfère ici les dériver en remplacant l'objectif (4.1), par

$$\min_{X\in\mathbb{R}^{m\times k},Y\in\mathbb{R}^{k\times n}} \frac{1}{2}\left\|M-(X\circ X)(Y\circ Y)\right\|_F^2,$$

où $\cdot\circ\cdot$ désigne le produit de Hadamard. Le problème perd ainsi ses contraintes, et il suffit d'annuler le gradient pour retrouver les conditions de stationnarité. En effet,

$$\nabla_X\left[\frac{1}{2}\left\|M-(X\circ X)(Y\circ Y)\right\|_F^2\right]=\left[\left((X\circ X)(Y\circ Y)-M\right)(Y\circ Y)^T\right]\circ X,$$
$$\nabla_Y\left[\frac{1}{2}\left\|M-(X\circ X)(Y\circ Y)\right\|_F^2\right]=\left[(X\circ X)^T\left((X\circ X)(Y\circ Y)-M\right)\right]\circ Y.$$

En remplaçant $X\circ X$ par W, et $Y\circ Y$ par H, on obtient donc les conditions de stationnarité

$$\begin{cases}\left[\left(WH-M\right)H^T\right]\circ W=0,\\ \left[W^T\left(WH-M\right)\right]\circ H=0,\end{cases}$$

identiques à celles de Karush-Kuhn-Tucker pour la NMF.

4.1.2 Les descentes de gradients

La plupart des algorithmes classiques sont basés sur des descentes de gradient. La descente de gradient classique consiste en

$$\begin{cases} W = W + \eta(XH^T - WHH^t), \\ H = H + \eta(W^TX - W^TWH), \end{cases}$$

où l'on fait attention à choisir η suffisamment petit pour garder W et H positifs.

Multiplicative Updates

C'est l'une des premières méthodes de calcul de la NMF, développée par Lee et Seung (2001). La méthode est une descente de gradient alternée, où le pas est déterminé par

$$\begin{cases} \eta_{ij}^W = \frac{W_{ij}}{(WHH^T)_{ij}}, \\ \eta_{ij}^H = \frac{H_{ij}}{(W^TWH)_{ij}}. \end{cases}$$

Avec ces règles, Lee et Seung on montré que la méthode convergeait vers un point stationnaire, et que W et H restaient positifs.

Gradient projeté

Une autre possibilité est d'opérer une descente de gradient en projetant la solution obtenue à chaque itération sur le quadrant positif.

$$\begin{cases} W_{i+1} = \Pi_+\left(W_i - \eta\left(W_iH_i - M\right)H_i^T\right), \\ H_{i+1} = \Pi_+\left(W_i - \eta W_i^T\left(W_iH_i - M\right)\right). \end{cases}$$

Cet algorithme converge lui aussi nécessairement vers un point stationnaire.

4.1.3 Moindres carrés alternés

L'un des problèmes de la NMF, est que l'objectif n'est pas convexe en W et H conjointement, comme le montre l'exemple suivant.

Exemple 4.1.1. *On pose* $f_X(W,H) = \|X - WH\|_F^2$, *pour* $X = \begin{pmatrix} 101 & 109 \\ 84 & 87 \end{pmatrix}$, *et*

$$W_1 = \begin{pmatrix} 93 & 106 \\ 90 & 93 \end{pmatrix}, W_2 = \begin{pmatrix} 102 & 101 \\ 113 & 116 \end{pmatrix},$$

$$H_1 = \begin{pmatrix} 144 & 122 \\ 117 & 109 \end{pmatrix}, W_2 = \begin{pmatrix} 55 & 113 \\ 113 & 116 \end{pmatrix}.$$

On a, pour $t = 0.2$,

$$\begin{aligned} &f_X(tW_1 + (1-t)W_2, tH_1 + (1-t)H_2) \\ &\quad > tf_X(W_1,H_1) + (1-t)f_X(W_2,H_2). \end{aligned}$$

L'objectif est par contre convexe en W seul, ou en H seul. Minimiser (4.1) par rapport à un seul des deux facteurs est un problème assez classique appelé moindres carrés non négatifs, ou NNLS. Le problème est dans P, et il existe des algorithmes efficaces permettant de le résoudre. On peut penser aux méthode d'*active set* exposées dans Kim *et al.* (2007) par exemple. Ces méthodes sont généralement basées sur des descentes de gradient, à des degrés de sophistication potentiellement différents.

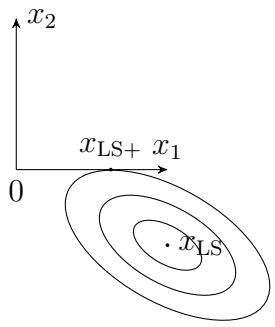
On peut remarquer au passage que la solution peut être très différente de la solution au problème des moindres carrés ordinaire projetée sur le quadrant positif. On peut réécrire les moindres carrés comme

$$\|Ax - b\|_2^2 = x^T A^T Ax - 2b^T Ax + b^t b,$$

et donc interpréter le problème comme faisant intervenir des ellipses d'équation

$$x^T A^T Ax - 2b^T Ax + b^t b = c.$$

En faisant augmenter c jusqu'à ce que l'ellipse intersecte le quadrant positif, comme illustré dans le schéma suivant en dimension 2, on résout le problème des moindres carrés positifs. On peut remarquer que $\Pi_+(x_{LS})$, la solution des moindres carrés ordinaire projetée sur le cadrant positif ne correspond pas à x_{LS+}, la solution des moindres carrés positifs.



L'algorithme des moindre carrés alternés consiste simplement à optimiser tour à tour par rapport à W, puis à H, jusqu'à convergence.

$$\begin{cases} W_{i+1} = \arg\min_{W \in \mathbb{R}_+^{m \times k}} \|X - WH_i\|_F^2, \\ H_{i+1} = \arg\min_{H \in \mathbb{R}_+^{k \times n}} \|X - W_{i+1}H\|_F^2. \end{cases}$$

Avec ces règles, W et H restent positives à chaque itération, et on est sûr de converger vers un point stationnaire.

4.1.4 Algorithme exploitant le rang de la matrice étudiée

Les algorithmes présentés dans les paragraphes précédents ne prennent pas en compte le rang de la matrice considérée. Nous présentons dans ce paragraphe une méthode différente des précédentes, exploitant le rang de la matrice à factoriser. Le rang que nous évoquons ici est le rang classique d'une matrice, autrement dit le plus petit entier k permettant d'écrire une matrice M comme $M = AB$, $A \in \mathbb{R}^{m \times k}, B \in \mathbb{R}^{k \times n}$.

Les facteurs A et B ne sont pas forcément constitués de coefficients positifs. Il arrive en effet fréquemment qu'une matrice positive de rang k n'admette aucune décomposition de ce type avec des facteurs constitués exclusivement d'éléments positifs. Une telle décomposition est cependant possible pour une matrice positive, mais elle demande potentiellement un entier k_+ supérieur à k. Cet entier est appelé rang positif de la matrice, et est noté $\operatorname{rank}_+ M$.

Quand $\operatorname{rank} M = \operatorname{rank}_+ M$, il existe des matrices $A \in \mathbb{R}^{m\times k}$ et $B \in \mathbb{R}^{k\times n}$ quelconques, $W \in \mathbb{R}_+^{m\times k}$ et $H \in \mathbb{R}_+^{k\times n}$ positives, et $X \in \mathbb{R}^{k\times k}$ inversible, telles que

$$\begin{cases} M = AB = WH, \\ W = AX, \\ H = X^{-1}B. \end{cases}$$

Pour une matrice AB de rang k, on peut donc imaginer un algorithme de NMF cherchant X inversible pour lequel $AX \geq 0$, et $X^{-1}B \geq 0$. Pour simplifier la chose, on se contente de X orthogonale. Le problème ressemble alors à un problème de Procuste, problème d'optimisation demandant d'ajuster au mieux deux formes géométriques définies par deux nuages de points, déjà étudié dans le cadre de la NMF symétrique dans Huang *et al.* (2014).

On peut tenter de le résoudre en résolvant

$$\min_{X\in\mathbb{R}^{k\times k}, X^TX=I_k} \|AX - \Pi_+(AX)\|_F^2 + \left\|B^TX - \Pi_+\left(B^TX\right)\right\|_F^2.$$

Pour simplifier encore le problème, on définit itérativement les matrices $W = \Pi_+(AX)$, et $H = \Pi_+\left(X^TB\right)$. De cette façon, la solution de l'équation

$$\min_{X\in\mathbb{R}^{k\times k}, X^TX=I_k} \|AX - W\|_F^2 + \left\|B^TX - H^T\right\|_F^2 \tag{4.2}$$

est simple à calculer. La solution est en effet donnée par une simple SVD. Si

$$A^TW + BH^T = USV^T,$$

alors $X = UV^T$ est solution de (4.2).

L'algorithme résultant est relativement simple. On itère

$$\begin{cases} USV^T = A^TW + BH^T, \\ X = UV^T, \\ W = \Pi_+(AX), \\ H = \Pi_+\left(X^TB\right). \end{cases}$$

On ne propose pas d'analyse théorique de la convergence, ou même de l'exactitude de cet algorithme. Son but est essentiellement de prendre part au comparatif entre les différentes techniques évoquées dans cette section, afin de montrer qu'il est assez facile d'imaginer des algorithmes bien plus rapides que les classiques sur certains types d'exemples bien précis.

4.1.5 Initialisation des algorithmes NMF

Les descentes de gradient souffrent de plusieurs inconvénients. Les points stationnaires vers lesquels elles convergent ne sont pas forcément des minimums globaux. Le temps nécessaire à atteindre l'un des ces points peut varier de façon importante selon les exemples considérés. De plus, les résultats des algorithmes dépendent généralement fortement des points de départ W_0 et H_0 choisis.

Une façon courante de les initialiser est de le faire aléatoirement. C'est rapide, mais les résultats obtenus vont varier d'une exécution à l'autre. Pire encore, la qualité des résultats, mesurée comme la norme euclidienne de la différence entre la matrice à factoriser et le produit des deux facteurs positifs, peut elle aussi osciller.

Pour parer à ce genre d'inconvénients, plusieurs autres méthodes d'initialisation ont été imaginées. L'une d'entre elles se base sur la SVD de la matrice à factoriser. L'argument soutenu par les partisans de ce genre de méthode repose sur le théorème de Perron-Frobenius.

Théorème 4.1.1. *Soit une matrice M à coefficients positifs. Sa meilleure approximation de rang* 1 *au sens de la norme de Frobenius est elle aussi positive. Autrement dit si*

$$M = \sum_{i=1}^{k} u_i s_i v_i^T$$

est la SVD de M, alors $\operatorname{sign} u_1 = \operatorname{sign} v_1$.

On peut donc initialiser la première colonne de W par u_1, et la première ligne de H par v_1^T. A ce stade, W et H sont des points critiques, et semblent donc être des mauvais candidats pour initialiser une descente de gradient (elle s'arrêterait aussitôt). Il faut donc continuer, soit en remplissant les entrées restantes de W et H par des valeurs positives aléatoires, soit en adoptant l'idée développée par Boutsidis et Gallopoulos (2008). L'idée est simple, pour tout $2 \leq i \leq k$ on compare la norme de $\Pi_+(u_i)\Pi_+(v_i)^T$ à celle de $\Pi_-(u_i)\Pi_-(v_i)^T$, et on initialise w_i et h_i par la meilleure paire.

Le temps de calcul dépend lui aussi de l'initialisation. Une bonne initialisation a en effet une influence sur le nombre d'itérations nécessaires à la convergence vers un point stationnaire. Il faut cependant faire attention à ne pas choisir des points de départ trop proches de points stationnaires triviaux, comme la meilleure approximation de rang 1, ou encore plus simplement 0.

4.1.6 Comparatif

Nous comparons dans ce paragraphe plusieurs des méthodes évoquées dans les paragraphes précédents. Nous nous intéressons au comportement de ces algorithmes sur plusieurs exemples générés aléatoirement. Comme le temps d'exécution peut varier fortement d'un problème a l'autre, nous allons générer plusieurs centaines d'exemples pour chaque algorithme, pour essayer de capturer le comportement standard de ces derniers.

Nous comparons 4 algorithmes :

	Erreur quadratique moyenne	Variance de l'erreur	Temps moyen de calcul	Variance du temps de calcul
ALS+RND2	1.0421689	0.4756757	0.0937356	0.0049742
MU+RND2	0.0000616	0.0000025	0.8191967	1.5389947
PRO+RND2	7.359D-09	3.141D-16	0.0012556	0.0000170
ADMM+RND2	0.0001200	0.0000093	5.00969,	8.1145694
ALS+PFK	0.0029530	0.0002601	0.6005667	0.6208594
MU+PFK	1.2916178	0.2018308	0.6673333	4.682622
PRO+PFK	2.323D-08	4.253D-16	0.0158667	0.0000606
ADMM+PFK	0.0000091	4.907D-13	0.1053667	0.1054541

TABLE 4.1 – Comportement de différents algorithmes de NMF sur des exemples simples.

- celui des moindres carrés alternés, décrit paragraphe 4.1.3, noté dans ce paragraphe ALS,
- celui des mises à jours multiplicatives, décrit paragraphe 4.1.2, noté MU,
- celui exploitant le rang de la matrice, décrit paragraphe 4.1.4, noté PRO,
- un algorithme inspiré des méthodes d'optimisation composite, décrit ultérieurement dans ce chapitre (paragraphe 4.4.4), noté ADMM.

Pour pouvoir comparer ces algorithmes, nous allons générer aléatoirement des matrices à factoriser. Dans un premier temps, ces matrices sont générées comme $M = WH$, où les coefficients de $W \in \mathbb{R}_+^{m\times k}$ et $H \in \mathbb{R}_+^{k\times n}$ sont générés aléatoirement selon une loi uniforme sur $[0, 2]$.

Les résultats présentés dans le tableau 4.1 montrent que tout se passe relativement bien avec ces matrices. La majorité du temps, les divers algorithmes retrouvent les facteurs de la NMF permettant une erreur quasi-nulle, en un temps raisonnable. On peut remarquer que certaines combinaisons algorithme/méthode d'initialisation fonctionnent mieux que d'autres. Une méthode semble nettement plus rapide que les autres : c'est celle exploitant le rang de la matrice. Le cadre de cet exemple est en effet idéal pour cette dernière. On a bien affaire à une matrice positive de rang k de rang positif k. Ce n'est généralement pas le cas en pratique.

On s'intéresse ensuite à un exemple plus compliqué, où plusieurs des algorithmes décrits au début de ce chapitre vont être mis en défaut. On s'intéresse cette fois à la NMF d'une matrice positive M définie comme le produit de Hadamard $M = N \circ N$, où chaque coefficient de la matrice $N \in \mathbb{R}^{m\times n}$ est défini comme

$$N_{i,j} \sim \mathcal{N}(0, 1).$$

Les algorithmes testés précédemment sont globalement plus lents sur ces exemples difficiles. Nous restreindrons donc le nombre d'exemples générés. Les résultats sont compilés dans le tableau 4.2. Il est frappant de constater que les algorithmes initialisés aléatoirement ont des temps de calcul pouvant varier de façon très importante d'un exemple à l'autre, en offrant des résultats dont la qualité est elle aussi variable.

	Erreur quadratique moyenne	Variance de l'erreur	Temps moyen de calcul	Variance du temps de calcul
ALS+RND2	1.1665124	0.3908195	18.76973	2530.5039
MU+RND2	0.4949121	0.0109956	3.66901	11.199879
PRO+RND2	0.9444607	0.0010544	0.09468	0.0014729
ADMM+RND2	0.7497382	0.0882963	24.23747	79.253587
ALS+PFK	0.4266032	0.0484506	35.8844	3050.7401
MU+PFK	1.1350556	0.1840098	0.132	0.077264
PRO+PFK	0.9388640	0.1194926	0.1009	0.0261028
ADMM+PFK	0.5041039	0.0653024	17.0653	256.46975

TABLE 4.2 – Comportement de différents algorithmes de NMF sur des exemples plus difficiles.

	Erreur quadratique moyenne	Variance de l'erreur	Temps moyen de calcul	Variance du temps de calcul
ALS+RND2	0.7753212	0.9537547	12.88998	5146.7192
MU+RND2	0.0560397	0.0057403	3.57723	11.511677
PRO+RND2	0.858098	0.0125322	0.16909	0.0406507
ADMM+RND2	0.1688123	0.0729671	21.56972	109.51574
ALS+PFK	0.0345853	0.0049363	7.5522	152.62167
MU+PFK	2.5485356	0.6971424	0.0998	0.0119280
PRO+PFK	0.8740864	0.1452720	0.5679	1.8769272
ADMM+PFK	0.0991793	0.0436871	7.0591	167.95081

TABLE 4.3 – Comportement de différents algorithmes de NMF sur des exemples plus diffiles encore.

Nous développons une dernière classe de problème aléatoire, où une matrice $M \in \mathbb{R}^{m \times n}$ est générée comme la projection d'une matrice $U \sim \mathcal{U}(0, 5)$ sur l'ensemble des matrices positives de rang k. Il s'agit à nouveau d'un problème difficile, mettant à nouveau à rude épreuve les méthodes de calcul de la NMF évoquées jusqu'ici.

Là encore, les algorithmes peuvent renvoyer des résultats très différents en fonction de leur initialisation, comme on peut le voir dans le tableau 4.3 compilant les résultats concernant cet exemple.

4.2 Limites théoriques

Nous présentons ici des éléments plus théoriques liés au calcul de la NMF. Ceux-ci ne nous ont pas permis de proposer des schémas de relaxations efficaces, comme celui proposé dans Chrétien *et al.* (2016). Toutefois, certaines des propriétés affichées par la factorisation des matrices positives semblaient mériter cette section.

Nous avons pu voir dans la section précédente que la résolution des problèmes de NMF pouvait être ardue. Les solutions trouvées par les différents algorithmes dépendent souvent fortement de leur initialisation, et les temps de calcul sont très variables. Il semble cependant qu'ils augmentent rapidement en fonction de la taille

du problème, et du rang demandé. Un des autres soucis tient à l'unicité des solutions, qui semble être l'exception plus que la règle. Dans cette section plus théorique, nous exposons par l'exemple la preuve de Vavasis concernant la NP-complétude du problème. Avant de tenter par deux reprises une relaxation SDP du problème - stratégie fructueuse pour le *clustering* - nous donnons quelques propriétés liant rang positif et complexité de certains programmes linéaires, afin d'exhiber l'un des rôles inattendus endossé par la NMF.

4.2.1 Complexité : réduction au simplexe en sandwich

Décider si le rang positif est égal au rang d'une matrice positive est NP-complet, comme démontré dans Vavasis (2009). Nous donnons ici la réduction du problème vers le problème de l'existence d'un simplexe compris entre deux polytopes, première partie de la preuve, car elle apporte un point de vue intéressant sur la NMF. Elle consiste d'abord à transformer la matrice M à factoriser pour l'amener vers une forme plus simple à exploiter, et ce sans modifier son rang ni son rang positif.

On s'intéresse à M de rang k. Il existe donc $A \in \mathbb{R}^{m\times k}$ et $B \in \mathbb{R}^{k\times n}$, avec $M = AB$, et $a_k \geq 0$, par Perron-Frobenius. Comme le rang et le rang positif restent inchangés quand on remplace M par $\mathrm{diag}(d)M$ pour tout vecteur $d > 0$, on supposera que $a_k = 1_k$. Si le rang de M est égal à son rang positif, il existe $X \in R^{k\times k}$ inversible avec $AX^{-1} \geq 0$, et $XB \geq 0$. Comme on peut remplacer X par $\mathrm{diag}(d)X$ pour tout vecteur $d > 0$ tout en respectant ces deux conditions, on peut se restreindre à $X_k = 1_k$. On pose alors $W = AX^{-1} \geq 0$. Ceci implique que $Wx_k = W1_k = a_k = 1_k$, autrement dit que $w^i \in \Delta_k$ pour tout i. Comme $(a_1, \cdots, a_{k-1}) = W(x_1, \cdots, x_{k-1})$, on en déduit que chaque ligne de $(a_1, \cdots, a_{k-1})$ est dans l'enveloppe convexe des lignes de $x_1, \cdots, x_{k-1}$. La seconde condition $XB \geq 0$ revient à dire que les lignes de $(x_1, \cdots, x_{k-1})$ sont dans un polytope défini par B. Dans le détail, cela équivaut en effet à $(x_1, \cdots, x_{k-1})(b^1, \cdots, b^{k-1})^T \geq -1_k(b^k)^T$.

Chercher une factorisation positive de rang k à la matrice de rang k revient donc à trouver k vecteurs dans $\mathbb{R}^{k-1}$ inclus dans un polytope défini par B, et dont l'enveloppe convexe contient les lignes de A.

Exemple 4.2.1. *On illustre l'interprétation géométrique par l'exemple suivant.*

$$W = \begin{pmatrix} 6 & 4 & 5 \\ 10 & 5 & 12 \\ 19 & 13 & 3 \\ 6 & 10 & 9 \end{pmatrix}, H = \begin{pmatrix} 8 & 14 & 12 & 6 & 9 \\ 12 & 10 & 7 & 7 & 11 \\ 9 & 7 & 8 & 13 & 9 \end{pmatrix}, M = WH.$$

On commence par écrire la décomposition en valeurs singulières de $M = USV^T$, avec $U \in \mathbb{R}^{m\times k}$, $S \in \mathbb{R}^{k\times k}$, et $V \in \mathbb{R}^{n\times k}$. La première colonne de U, qui correspond à la plus grande valeur singulière, est positive strictement, et pose donc $W = \mathrm{diag}(u_1)^{-1}US^{\frac{1}{2}}$, et $H = S^{\frac{1}{2}}V^T$. En inversant l'ordre des colonnes de W, et des lignes de H, on obtient

$$\mathrm{diag}(u_1)^{-1}M = AB$$

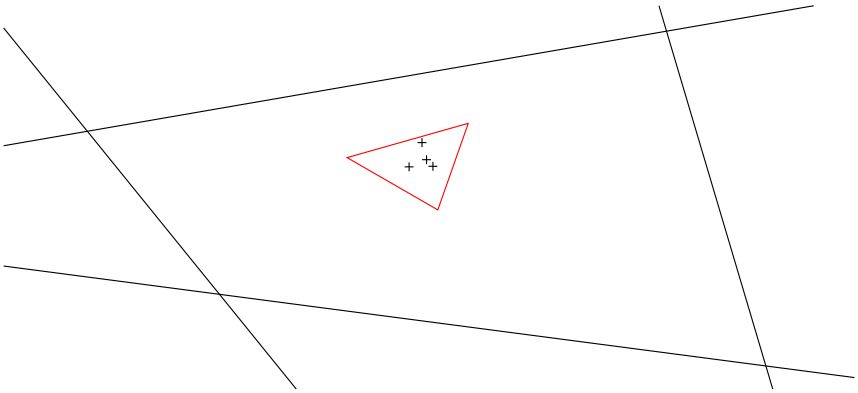


FIGURE 4.1 – Il s'agit de trouver un triangle contenant les points, et inclus dans le polytope défini par les lignes noires.

avec

$$A = \begin{pmatrix} 0.0729196 & -0.0990798 & 1 \\ 0.1832020 & -0.2461903 & 1 \\ -0.0061463 & 0.2804461 & 1 \\ -0.2289556 & -0.2584016 & 1 \end{pmatrix},$$

et

$$B = \begin{pmatrix} -3.0880228 & 0.7846842 & 2.6969892 & 1.5602773 & -1.7644808 \\ -0.7141037 & 4.6633042 & 1.7027187 & -7.0338704 & -0.2936758 \\ 15.02203 & 17.170419 & 14.734204 & 12.791588 & 15.178062 \end{pmatrix}$$

On dessine alors l'ensemble E des points de coordonnées $(a_{i,1}, a_{i,2})$ pour $1 \leq i \leq m$, et le polytope

$$Q = \{(x_1, x_2) \in \mathbb{R}^2, b_{1,i}x_1 + b_{2,i}x_2 + b_{3,i} \geq 0, 1 \leq i \leq n\}.$$

Pour trouver une solution NMF de rang 3, il s'agit alors de trouver un triangle Δ tel que $P = \text{conv}(E) \subset \Delta \subset Q$. La solution correspondant à la solution (W, H) initiale est dessinée en rouge sur la figure 4.1. Il apparaît sur la figure qu'il n'y a pas unicité de la solution, il est en effet possible d'imaginer une large collection de triangles solutions.

Il semble difficile de résoudre directement le problème de cette façon.

4.2.2 NMF comme système de polynômes

On peut voir la NMF comme un problème de polynôme. Chercher $W \geq 0$, $H \geq 0$ tels que $M = WH$ pour M une matrice positive donnée revient à chercher une solution à

$$\sum_{\substack{1 \leq i \leq m \\ 1 \leq j \leq n}} \left(m_{i,j} - \sum_{1 \leq l \leq k} w_{i,l} h_{l,k} \right)^2 = 0, (w_{i,l})_{\substack{1 \leq i \leq m \\ 1 \leq l \leq k}} \in \mathbb{R}_+^{mk}, (h_{l,j})_{\substack{1 \leq l \leq k \\ 1 \leq j \leq n}} \in \mathbb{R}_+^{nk}$$

avec $(w_{i,l})_{\substack{1\leq i\leq m\\1\leq l\leq k}} \in \mathbb{R}_+^{mk}$, et $(h_{l,j})_{\substack{1\leq l\leq k\\1\leq j\leq n}} \in \mathbb{R}_+^{nk}$.

Cela équivaut à déterminer si il existe un point sur la variété algébrique réelle définie comme

$$\left\{ m_{i,j} - \sum_{1\leq l\leq k} w_{i,l}^2 h_{l,k}^2 = 0, 1 \leq i \leq m, 1 \leq j \leq n \right\}.$$

Cette variété est définie par mn polynômes de degré 4 en $mk+kn$ variables. Depuis Tarski-Seidenberg, on sait que ce genre de problème est décidable. Les algorithmes de géométrie algébrique réelle permettant de décrire cette variété sont par contre en temps exponentiel. Par exemple l'algorithme de Renegar calculerait une solution en $(4mn)^{O(mk+kn)}$.

En réduisant le nombre de variables utilisées, les auteurs de Arora *et al.* (2012) montrent qu'il existe un algorithme en $O\left((nm)^{r^2 2^r}\right)$, et que, sous réserve d'une certaine conjecture sur la complexité de SAT, il ne peut exister d'algorithme en $(nm)^{o(r)}$. L'article présente aussi un algorithme polynomial pour une certaine classe de matrices, les matrices séparables, dont nous reparlons en détail section 4.3.1. Moitra montre dans Moitra (2016) qu'on peut réduire encore le nombre de variables utilisées, et propose un algorithme en $(nm)^{O(r^2)}$.

Plutôt que de chercher à résoudre le système de polynôme, on pourrait aussi chercher à minimiser sa norme. Il existe pour cela un outil qui écrit un polynôme comme la somme de carrés de polynômes. On note l'ensemble des polynômes pouvant s'écrire comme somme de carrés SOS. Ainsi, pour minimiser un polynôme P, on cherche à exprimer $P-\alpha$ comme une somme de carrés, pour α le plus grand possible. En montrant qu'il existe $\alpha > 0$ tel que

$$\|M - WH\|_F^2 - \alpha \in \text{SOS},$$

on prouve que le rang positif de M est supérieur à k. Les sommes de carrés sont liées aux programmes SDP, nous en reparlons brièvement paragraphe 4.2.4.

4.2.3 Formulations étendues

Le rang positif a une interprétation intéressante en combinatoire, due à Yannakakis (1988). Son théorème le lie à la complexité polyédrale. On peut définir la taille d'un polyèdre comme le nombre de demi-espaces minimum nécessaire à sa formulation. Autrement dit, si ce dernier est en dimension n, c'est son nombre de faces de dimension $n-1$. Une formulation étendue d'un polyèdre P est la donnée d'un autre polyèdre Q de dimension supérieure, et d'une projection π, vérifiant $P = \pi Q$. La complexité extensionnelle de P est le nombre de faces minimal pour une formulation étendue de P. Cette mesure de la complexité d'un polyèdre est égal au rang positif d'une matrice associée à P, d'après un théorème de Yannakakis.

Un polyèdre peut être défini comme l'enveloppe convexe d'une série de points, ou comme l'intersection d'une série de demi-espaces. On parle respectivement de V-description et de H-description. En mathématique,

$$P = \text{conv}(x_1, x_2, \cdots, x_n),$$

ou

$$P = \left\{ x \in \mathbb{R}^d, Ax \leq b \right\}.$$

La matrice de *slack* de P notée S est définie par

$$s_{i,j} = b_i - A_i x_j.$$

Exemple 4.2.2. *On factorise la matrice de slack S de l'heptagone. Un polygone régulier peut être défini comme l'enveloppe convexe des points $P_k, k \in \mathbb{N}$ de*

$$P_k = \left(\cos \frac{2k\pi}{n}, \sin \frac{2k\pi}{n} \right).$$

Chaque face a pour équation

$$\cos\left((2k+1)\frac{\pi}{n}\right) x + \cos\left((2k+1)\frac{\pi}{n}\right) y = \cos\frac{\pi}{n}.$$

On en déduit que

$$s_{i,j} = \cos\left(\frac{\pi}{n}\right) - \cos\left(\frac{\pi}{n}(2(i-j)+1)\right).$$

On constate au passage que S est circulante. En notant $\alpha = \sin\frac{\pi}{14}$, on obtient

$$\begin{aligned} s_{2,1} &= (\alpha+1)(1-2\alpha), \\ s_{3,1} &= (\alpha+1)(1+2\alpha-4\alpha^2), \\ s_{4,1} &= (\alpha+1)(2-2\alpha). \end{aligned}$$

On pose $T[X] = 1-2X$, $U[X] = 1+2X-4X^2$, $V[X] = 1-4X-4X^2+8X^3$ et

$$M[X] = \begin{pmatrix} 0 & 0 & T[X] & U[X] & U[X]+1 & U[X] & T[X] \\ T[X] & 0 & 0 & T[X] & U[X] & U[X]+1 & U[X] \\ U[X] & T[X] & 0 & 0 & T[X] & U[X] & U[X]+1 \\ U[X]+1 & U[X] & T[X] & 0 & 0 & T[X] & U[X] \\ U[X] & U[X]+1 & U[X] & T[X] & 0 & 0 & T[X] \\ T[X] & U[X] & U[X]+1 & U[X] & T[X] & 0 & 0 \\ 0 & T[X] & U[X] & U[X]+1 & U[X] & T[X] & 0 \end{pmatrix}$$

$$W[X] = \begin{pmatrix} T[X] & T[X]\,U[X] & 0 & 0 & U[X] & 0 \\ 0 & U[X] & 0 & 0 & 1 & T[X] \\ 0 & U[X] & U[X] & 0 & 0 & U[X] \\ 0 & U[X] & 0 & 1 & 0 & T[X] \\ T[X] & T[X]\,U[X] & 0 & U[X] & 0 & 0 \\ U[X] & 0 & U[X] & T[X] & 0 & 0 \\ U[X] & 0 & U[X] & 0 & T[X] & 0 \end{pmatrix}$$

$$H[X] = \begin{pmatrix} 0 & 0 & U[X] & U[X] & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & T[X] & 0 & 0 & T[X] & 0 & 0 \\ U[X] & U[X]+1 & 2X(U[X]+1) & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 2X(U[X]+1) & U[X]+1 & U[X] & 0 \\ U[X] & 0 & 0 & 0 & 0 & U[X] & T[X] \end{pmatrix}$$

Il faut alors vérifier que $T[X]M[X] - W[X]H[X]$ *vaut*

$$\begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & V[X] & -T[X]\,V[X] & 0 & -T[X]\,V[X] \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -T[X]\,V[X] & V[X] & 0 & 0 & 0 & -T[X]\,V[X] \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -V[X]\,(T[X]+1) & T[X]\,V[X] & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & T[X]\,V[X] & -V[X]\,(T[X]+1) & 0 & 0 \end{pmatrix}.$$

Comme $V[X]$ *est le polynôme minimal de* α*, et que* $S = \frac{M(\alpha)}{\alpha+1}$*, il existe un hexaèdre dont la projection est l'heptagone régulier.*

Il manque quelque chose dans l'exemple 4.2.2. Il y est en fait démontré que $\mathrm{rank}_+(S) \leq 6$. Pour connaître le rang positif de S, il faut encore une borne inférieure. Une borne relativement simple à introduire est celle donnée par les couvertures par rectangles. Si $m_{i,j} = 0$, alors $\sum_k w_{i,k}h_{k,j} = 0$, et comme $W \geq 0$ et $H \geq 0$, alors supp $w_i \cap$ supp $h^j = \emptyset$. Le nombre minimum de rectangles nécessaires pour recouvrir les entrées non-nulles d'une matrice est donc une borne inférieure sur son rang positif. Le problème principal de ce genre de borne est la difficulté de son calcul.

D'autres techniques permettent de borner la complexité extensionnelle de certains cônes, dont certains utilisés en optimisation combinatoire. Cela permet d'aborder ceux-ci comme des programmes linéaires, et de garantir que ces formulations sont le plus efficaces possible.

Exemple 4.2.3. *Pour un graphe pondéré complet* (V_n, E_n) *à* n *sommets, trouver un arbre couvrant de poids minimal est un problème classique en optimisation combinatoire. L'algorithme classiquement employé pour résoudre ce problème est l'algorithme de Kruskal, qui résout le problème en complexité* $n \log n$*. Une formulation possible du problème comme programme linéaire est la suivante. Le poids d'une arête* $e \in E$ *est dénoté* c_e*, et, pour un sous ensemble* $W \subset V_n$*, l'ensemble des arêtes dont les deux extrémités sont dans* W *est noté* $E_n(W)$*. On doit alors résoudre*

$$\min \sum_{e \in E_n} c_e x_e \text{ tel que } \begin{cases} \sum_{e \in E} x_e = n-1, \\ \sum_{e \subset E_n(S)} x_e \leq |S| - 1, \forall S \subset V, \\ x_e \geq 0, \forall e \in E. \end{cases}$$

Dans cette formulation, la description du polytope de faisabilité nécessite un nombre exponentiel (il existe 2^n *sous-ensembles de* V*) d'inégalités non redondantes. Résoudre ce programme nécessiterait par conséquent un temps exponentiel. Comme le problème est dans* P*, on pourrait s'attendre à l'existence d'une meilleure description. C'est le cas. On peut en effet décrire ce polytope comme la projection d'un polytope défini par* $O(n^3)$ *inégalités. La description de ce polytope est donnée dans Kaibel (2011). La complexité extensionnelle du polytope des arbres couvrants est donc bornée supérieurement par* $O(n^3)$*.*

D'autres problèmes, NP-durs cette fois, peuvent eux aussi s'écrire sous forme de programme linéaire. C'est le cas du problème du voyageur de commerce, pour

lequel on s'attend donc à ce qu'il n'existe pas de formulation polynomiale. Cela impliquerait $P = NP$. La borne exponentielle a été récemment établie par Fiorini *et al.* (2015).

4.2.4 Deux tentatives de relaxations

On peut imaginer différentes relaxations pour la NMF. Malheureusement, aucune des options envisagées durant cette thèse ne semble produire d'algorithme offrant un temps de calcul raisonnable.

Par programmation co-positive

Une matrice complètement positive est une matrice $M \in \mathbb{R}^{n\times n}$ symétrique se factorisant comme

$$M = AA^T,$$

où $A \in \mathbb{R}_+^{n\times m}$ est une matrice positive. Les matrices complètement positives forment un cône, noté CP_n.

On peut reformuler la NMF comme un problème d'appartenance à ce cône. En effet, si $M = WH$, alors

$$\begin{pmatrix} W \\ H^T \end{pmatrix} \begin{pmatrix} W^T & H \end{pmatrix} = \begin{pmatrix} WW^T & M \\ M^T & H^TH \end{pmatrix} \in \mathrm{CP}_n.$$

On peut donc imaginer une NMF relaxée formulée comme

$$\min_{X\in\mathbb{R}^{m\times m}} \operatorname{tr} X + \operatorname{tr} Y \text{ tel que } \begin{pmatrix} X & M \\ M^T & Y \end{pmatrix} \in \mathrm{CP}_n.$$

Cette formulation est un exemple de programmation complètement positive, un type de programme linéaire sur un cône convexe. Malgré cette écriture convexe, la résolution de ces problèmes d'optimisation est généralement NP-dure. On peut en effet relativement facilement lier ces problèmes à n'importe quel problème de programmation quadratique, comme par exemple dans Burer (2009). Il existe bien des problèmes à la fois NP-durs et convexes. La difficulté tient ici au fait que tester si une matrice est complètement positive est difficile. Les stratégies classiques d'optimisation contrainte, généralement basées sur les projections, ou sur une fonction barrière permettant d'élaborer un algorithme de point intérieur échouent donc dans ce cas.

Il existe cependant des stratégies d'approximation de ce cône. Parrilo (2000) propose par exemple une hiérarchie de cônes imbriqués permettant d'approximer le cône des matrices co-positives, le cône dual de CP. Une matrice $A \in \mathbb{R}^{n\times n}$ est co-positive si,

$$\forall x \in \mathbb{R}_+^n, x^TAx \geq 0.$$

On peut donc tester la co-positivité de A en étudiant la positivité du polynôme

$$P_A(x_1, \cdots, x_n) = \begin{pmatrix} x_1^2 \\ \vdots \\ x_n^2 \end{pmatrix}^T \cdot A \cdot \begin{pmatrix} x_1^2 \\ \vdots \\ x_n^2 \end{pmatrix}.$$

Établir cette positivité est à nouveau un problème complexe. Il existe cependant des relaxations, dont une utilise les sommes de carrés de polynômes, déjà évoquées au paragraphe 4.2.2. Pour tester si un polynôme P de degré $2d$ est une somme de carrés, on le réécrit comme une forme quadratique en z, vecteur regroupant tous les monômes en $x_1, \cdots, x_n$ de degré inférieur ou égal à d. Tester l'appartenance de P au cône SOS des polynômes décomposables en sommes de carrés revient alors à chercher une matrice B semi-définie positive telle que

$$P(x) = z^T Q z.$$

Dans notre cas, il faut donc trouver une matrice B semi-définie positive telle que

$$P_A(x) = \begin{pmatrix} x_1^2 \\ \vdots \\ x_i x_j \\ \vdots \\ x_n^2 \end{pmatrix}^T \cdot B \cdot \begin{pmatrix} x_1^2 \\ \vdots \\ x_i x_j \\ \vdots \\ x_n^2 \end{pmatrix}.$$

Il s'agit déjà d'un gros problème SDP de taille $n(n+1)/2$. De plus, tous les polynômes positifs ne sont pas des sommes de carrés. Un contre exemple est le polynôme de Motzkin $x^4y^2 + x^2y^4 - 3x^2y^2 + 1$, positif sur $\mathbb{R}^2$, mais qui ne peut s'exprimer comme somme de carrés de polynômes en x et y. Pour parer à ce genre de problème, Parrilo propose donc une succession de cônes imbriqués approximant le cône des matrices co-positives comme

$$K_r = \left\{ A \in S_n \text{ tel que } P_A(x) \left(\sum_{i=1}^{n} x_i^2 \right)^r \in \text{SOS} \right\}.$$

Cette décomposition semble donc beaucoup trop lourde pour être mise en œuvre en pratique, même pour des matrices de taille raisonnable.

Une autre façon d'approximer la NMF par programmation complètement positive est celle introduite par Krishnamurthy et d'Aspremont (2012) basée sur une propriété des matrices semi-définies positives : en calculant l'exponentielle coefficient par coefficient de ces matrices, on obtient une matrice complètement positive. Là encore, en pratique, les choses ne semblent pas si aisées.

Par programmation semi-définie positive

On peut aussi imaginer une relaxation plus directe, amenant immédiatement à la résolution d'un SDP. Si $M = WH$, alors

$$Y = \begin{pmatrix} \text{vec}(W) \\ \text{vec}(H) \end{pmatrix} \begin{pmatrix} \text{vec}(W) \\ \text{vec}(H) \end{pmatrix}^T$$

est solution de

$$\min_{X \succeq 0, X \geq 0} \text{rank}\, X \text{ tel que } A^{i,j} \cdot X = m_{i,j},$$

où $A^{i,j} \in \mathbb{R}^{mk \times kn}$ est une matrice définie de façon à ce que

$$A^{i,j} \cdot Y = \sum_{l=1}^{k} w_{i,l} h_{l,j} = m_{i,j}.$$

Plus précisément, elle associe à la variable $X \in S_{mk+kn}$ la somme

$$A^{i,j} \cdot X = \sum_{l=1}^{k} \frac{x_{i+m(l-1),mk+l+k(j-1)} + x_{mk+l+k(j-1),i+m(l-1)}}{2}.$$

En remplaçant le rang par sa relaxation convexe, on obtient donc le SDP suivant

$$\min_{X \succeq 0, X \geq 0} \operatorname{tr} X \text{ tel que } A^{i,j} \cdot X = m_{i,j}.$$

Ce problème est de taille $mk + kn$, et sa résolution semble donc à nouveau hors de portée pour des matrices de taille raisonnable.

4.3 Factorisation des hypercubes

Nous avons pu voir dans ce chapitre que la décomposition en facteurs positifs était difficile. Les relaxations étudiées sont trop lentes pour être appliquées à des matrices de petite taille, et les algorithmes classiques semblent incapable de gérer des matrices de taille raisonnable. Pour appliquer la NMF aux hypercubes, il faut donc adopter une approche quelque peu différente.

4.3.1 Notion de séparabilité

Si les travaux de Arora *et al.* (2012) sont très critiques concernant l'existence d'un algorithme résolvant les problèmes génériques de NMF efficacement, ils détaillent une classe de problèmes pour lesquels il existe un algorithme efficace : les problèmes séparables. Cette condition de séparabilité décrit une classe de matrices dans laquelle on sait résoudre le problème NMF en temps polynomial. Elle équivaut à dire que la matrice étudiée possède une factorisation où l'un des facteurs admet une matrice diagonale comme sous-matrice. Plus précisément, une matrice positive de rang positif k est dite séparable si il existe deux matrices W et H telles que $M = WH$, et que W contient k lignes où seulement une valeur est non-nulle, jamais dans la même colonne. Autrement dit il existe un k-uplet $(i_1, \cdots, i_k)$ tel que

$$\begin{pmatrix} w^{i_1} & \cdots & w^{i_k} \end{pmatrix} = \operatorname{diag}(d_1, \cdots, d_k) \geq 0.$$

En transposant la définition à H on peut alors écrire la matrice M à permutation des colonnes près comme

$$M = W \begin{pmatrix} D & h_{k+1} & \cdots & h_n \end{pmatrix} = \tilde{W} \begin{pmatrix} I_k & \tilde{H} \end{pmatrix} = \begin{pmatrix} \tilde{W} & \tilde{W}\tilde{H} \end{pmatrix}.$$

Il existe donc une factorisation positive de rang k dans laquelle les colonnes de W sont en fait des colonnes de M. Cette propriété des problèmes séparables est

d'ailleurs souvent interprétée en analyse hyper-spectrale par la présence de spectres 'purs' dans l'hyper-cube, c'est à dire qu'il doit exister au moins k pixels dans l'hypercube constitués à 100% d'un des k spectres de base présent dans l'image. Si dans un premier temps cela peut sembler être un progrès considérable, du point de vue de Vavasis cela revient à dire que l'un des polytopes P ou Q décrits section 4.2.1 est déjà un simplexe. Trouver un simplexe entre les deux polytopes$P$ et Q devient donc une tâche quasiment triviale. Il suffit de calculer, en fonction du facteur contenant la matrice diagonale $k \times k$, soit les sommets du polytope P soit ceux du polytope Q.

Les calculs d'enveloppe convexe sont relativement lourds : l'enveloppe convexe de n points en dimension d se calcule en $O(n^{\lfloor d/2 \rfloor})$ en général. Mais comme dans le cas séparable l'enveloppe convexe est un simplexe, il existe des algorithmes plus rapides.

4.3.2 Séparabilité en pratique

La séparabilité est une hypothèse sous-jacente à certains algorithmes s'intéressant à l'extraction des spectres de base mentionnés section 1.4.2. Sans en faire mention explicitement, ces algorithmes tentent en quelque sorte de calculer l'enveloppe convexe des spectres, en s'attendant à ce que cette enveloppe ait le bon nombre de sommets.

Quand l'hypothèse de séparabilité n'est pas parfaitement vérifiée, l'algorithme SPA présente des garanties de robustesse, décrites dans Gillis et Vavasis (2014).

Plus grand simplexe inscrit

Un des premiers algorithmes d'extraction de spectres de base développé est NFINDR, introduit par Winter (1999). On démarre avec k spectres choisis aléatoirement parmi les n spectres disponibles, puis on essaye itérativement d'augmenter le volume du simplexe courant en échangeant un de ces spectres avec un autre spectre du cube. Pour calculer le volume du simplexe engendré, l'algorithme utilise la formule suivante. Elle concerne un $k-1$-simplexe en dimension $k-1$.

$$\operatorname{vol} \Delta\left(u_1, \cdots, u_k\right) = \det \begin{pmatrix} 1 & 1 & \cdots & 1 \\ u_1 & u_2 & \cdots & u_k \end{pmatrix} \tag{4.3}$$

Comme les spectres du cube ne sont généralement pas en dimension $k-1$, mais en dimension m, l'algorithme commence par projeter ces derniers dans un espace de dimension $k-1$ par PCA. On résume tout cela dans l'algorithme 4.

Il n'est pas nécessaire de recalculer le déterminant à chaque itération, il suffit de maintenir k blocs B_i de la matrice décrite équation (4.3). Par exemple, on peut calculer le volume obtenu en remplaçant w_1 par un vecteur $v \in \mathbb{R}^{k-1}$, comme

$$\operatorname{vol} \Delta(v, w_2, \cdots, w_k) = \left(1 - 1_{k-1}^T B_1^{-1} v\right) \det(B_1),$$

où $B_1 = (w_2, \cdots, w_k)$. Si il est intéressant de remplacer w_1 par v, on met alors à jour $B_2, \cdots, B_k$, ainsi que leurs déterminants et inverses. Cela permet d'accélérer fortement l'algorithme.

Algorithme 4 N-FINDR

Entrée : n spectres $m_1, \cdots, m_n \in \mathbb{R}^m$

Sortie : k spectres constituant potentiellement des bons endmembers $w_1, \cdots, w_k \in \mathbb{R}^m$

Calculer la projection PCA des spectres $u_1, \cdots, u_n \in \mathbb{R}^{k-1}$.
Sélectionner aléatoirement k indices $\iota_1, \cdots, \iota_k \in \{1, \cdots, n\}$.
for $i \in \{1, \cdots, n\}$ **do**
  for $j \in \{1, \cdots, k\}$ **do**
    On pose $\kappa = (\iota_1, \cdots, \iota_{j-1}, i, \iota_{j+1}, \cdots, \iota_k)$
    if $\operatorname{vol} \Delta_\kappa > \operatorname{vol} \Delta_\iota$ **then**
      $\iota = \kappa$.
    end if
  end for
end for
On pose $w_1 = m_{\iota_1}, \cdots, w_k = m_{\iota_1}$.

Une idée très similaire développée dans Bauckhage (2014); Thurau *et al.* (2012) propose de remplacer la formule utilisée pour le calcul du volume du simplexe. Les deux articles utilisent la formule de Cayley-Menger. L'algorithme, nommé SiVM, calcule les volumes par

$$\operatorname{vol}^2 \Delta(u_1, \cdots u_k) = \frac{(-1)^k}{2^{k-1}\left((k-1)!\right)^2} \det A,$$

où $A \in \mathbb{R}^{k+1 \times k+1}$ est la matrice définie comme

$$A = \begin{pmatrix} 0 & 1 & 1 & 1 & \cdots & 1 \\ 1 & 0 & d_{1,2}^2 & d_{1,3}^2 & \cdots & d_{1,k}^2 \\ 1 & d_{2,1}^2 & 0 & d_{2,3}^2 & \cdots & d_{2,k}^2 \\ 1 & d_{3,1}^2 & d_{3,2}^2 & 0 & \cdots & d_{3,k}^2 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & d_{k,1}^2 & d_{k,2}^2 & d_{k,3}^2 & \cdots & 0 \end{pmatrix},$$

où $d_{i,j} = \|u_i - u_j\|_2^2$ désigne la distance euclidienne entre le i-ème et le j-ème endmember. Le principal intérêt d'utiliser ce genre de formule est qu'elle est basée uniquement sur la distance entre les spectres, et que l'on peut donc utiliser d'autres distances. Section 6.2.3, on utilisera des géodésiques à la place de la distance euclidienne, pour essayer de calculer une NMF non-linéaire.

Ici encore, plutôt que de recalculer à chaque itération le déterminant, on peut se contenter de beaucoup plus léger. En définissant A_k comme la matrice A amputée de sa dernière ligne et de sa dernière colonne, et $v_k = \left(1, d_{k,1}^2, d_{k,2}^2, \cdots, d_{k,k-1}^2\right)$, on a

$$\det A = (-1)^k \left(v_k^T A_k^{-1} v_k\right) \det A_k.$$

On peut ainsi grandement accélérer le calcul de SiVM. Cette façon de faire n'est pas celle adoptée dans Thurau *et al.* (2012). Les auteurs préfèrent approximer les

distances entre chaque spectre de base par une même constante. Le gain de temps semble assez marginal.

Ce problème de plus volumineux simplexe inscrit est équivalent au problème de sélection de colonnes décrit dans Çivril et Magdon-Ismail (2009). L'article montre que le problème est NP-dur, et même qu'il est NP-dur à approximer au delà de $2\sqrt{2}/3$.

Plus petit simplexe englobant

C'est le point de vue complémentaire à celui développé dans le paragraphe précédent. On cherche cette fois à minimiser le volume d'un simplexe continant l'ensemble des spectres du cube. C'est l'approche adoptée par Bioucas-Dias (2009) ou par Li *et al.* (2015) par exemple. Le problème est généralement posé comme

$$\min \operatorname{vol} \Delta \text{ tel que } x_1, \cdots, x_n \in \Delta.$$

Chacun des algorithmes évoqués ci-dessus fait des hypothèses simplificatrices différentes afin de proposer une solution approximée à ce problème difficile.

Par projections

Il existe un autre type d'algorithme s'essayant au calcul d'endmembers, fonctionnant sur un principe différent de ceux évoqués dans les deux paragraphes précédents. Plutôt que de faire grandir un simplexe contenu dans l'enveloppe des spectres, le simplexe est construit sommet par sommet, en cherchant à chaque étape un point extrême, retenu comme nouveau sommet. C'est ce critère d'extrêmalité qui différencie les 3 algorithmes que nous évoquons maintenant.

- VCA ou vertex component analysis (Nascimento et Dias, 2005a) : une fois les spectres du cube projetés sur les spectres de base déjà déterminés, le nouveau spectre de base est choisi comme le point le plus éloigné d'un hyperplan choisi aléatoirement. Autrement dit, on choisi un vecteur c aléatoire, et on retient le spectre x maximisant $\left|c^T x\right|$.

- SPA ou algorithme de projection successive (Araújo *et al.*, 2001) : une fois les spectres du cube projetés sur les spectres de base déjà calculés, le nouveau spectre de base est choisi comme celui dont la norme euclidienne est la plus importante.

Le fonctionnement de VCA est proche d'un autre algorithme, PPI ou pixel purity index, la différence étant que PPI ne projette pas les spectres à chaque itération sur les endmembers déjà obtenus.

L'algorithme 5 décrit SPA en détail. Il suffirait de changer la quatrième ligne pour implémenter VCA.

4.3.3 Comparatif

Les différents algorithmes ont des comportements relativement similaires lorsque le problème de NMF étudié est séparable. Lorsque ce n'est plus tout à fait le cas, les

Algorithme 5 L'algorithme de projections successives SPA

Entrée : $M \in \mathbb{R}^{m \times n}$, $r \in \mathbb{N}, r < n$ {La matrice de laquelle on veut trouver une base W bonne candidate à la factorisation NMF.}

Sortie : Un ensemble d'indices $\mathcal{K}$ tel que $M(\mathcal{K}, :) \approx W$ {L'algorithme ne renvoie pas une matrice, mais un ensemble d'indices correspondant aux lignes de M à garder pour avoir W.}

1: $R = M$
2: $\mathcal{K} = \{\}$
3: **for** $i = \{1, \cdots, r\}$ **do**
4: $\quad p = \arg\max_{j \in \{1, \cdots n\}} \|r_j\|_2$
5: $\quad R = \left(I - \frac{r_p r_p^T}{\|r_p\|_2^2}\right) R$
6: $\quad \mathcal{K} = \mathcal{K} \cup \{p\}$
7: **end for**

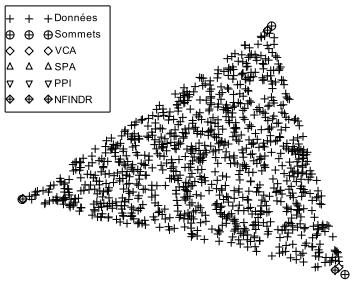
	$\|M - WH\|_F$	temps de calcul de W et H
PPI	95204.37	44.658
NFINDR	54756.918	60.93
VCA	87109.263	39.526
SPA	87643.455	41.195
MU	31945.176	222.682
PRO	32371.047	15.133

TABLE 4.4 – Calcul d'une NMF avec différents algorithmes présentés dans ce chapitre

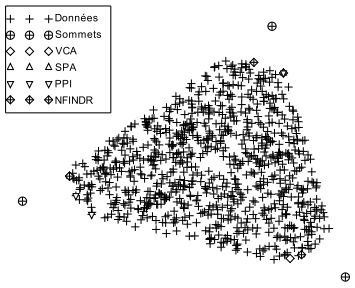
résultats proposés peuvent être légèrement différents. Ces différences sont montrées sur plusieurs exemples correspondant aux figures 4.2a, 4.2b et 4.2c. Ces exemples montrent que les algorithmes décrits dans cette section fonctionnent bien dans les cas où la séparabilité est presque vérifiée. Quand ce n'est pas le cas, certains d'entre eux, comme PPI sur la figure 4.2b, ne parviennent pas à trouver de solution satisfaisante. D'autres semblent capables de faire mieux. En plus de présenter des garanties théoriques de robustesse (Gillis et Vavasis, 2014), SPA semble par exemple donner de meilleurs résultats que ses concurrents.

Nous donnons aussi dans le tableau 4.4 différents résultats obtenus par ces algorithmes sur l'hyper-cube *Urban*[1], un cube de taille $307 \times 307 \times 162$ souvent utilisé dans la littérature hyper-spectrale. Nous comparons les temps de calcul et les erreurs commises par les différents algorithmes évoqués dans ce paragraphe, et ceux obtenus par deux algorithmes introduits au début de ce chapitre : l'algorithme de Lee et Seung, décrit paragraphe 4.1.2, et celui exploitant le rang des données décrit paragraphe 4.1.4. Comme nous l'avons déjà évoqué, l'utilisation de ces algorithmes de détermination des spectres de base est souvent perçue comme l'une des seules façons de calculer la NMF d'un cube en un temps raisonnable, car ces derniers permettent de découpler le calcul, en ne s'intéressant qu'au facteur W, le calcul de H se faisant à posteriori par moindres carrés positifs. Sur cet exemple, l'algorithme de

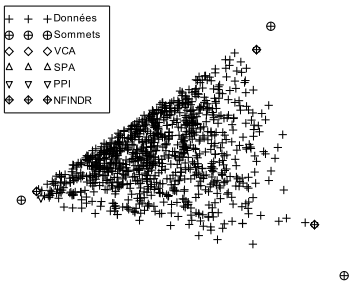
1. Disponible en ligne à http://www.agc.army.mil/



(a) Points répartis uniformément sur le simplexe.



(b) Points répartis uniformément sur l'intersection d'un simplexe et d'une boule.



(c) Points répartis selon la loi de Dirichlet $\mathcal{D}(1, 2, 3)$.

FIGURE 4.2 – Trois exemples de densité sur des 2-simplexes, et les résultats proposés par les algorithmes décrits paragraphe 4.3.2

Lee et Seung est en effet beaucoup plus lent que les autres. L'algorithme exploitant le rang semble par contre être un sérieux concurrent aux méthodes commençant par la détermination des spectres de base.

4.4 Régularisation

Si on ne peut pas trouver de solution exacte et globale au problème NMF, peut-être peut-on obtenir de bons minima locaux en modifiant l'objectif de NMF, en précisant les qualités attendues d'une solution. On peut souhaiter par exemple qu'une solution présente un certain degré de sparsité. En imagerie hyper-spectrale, on s'attendra plutôt à ce que les abondances présentent une certaine cohérence spatiale.

4.4.1 Ajout d'un objectif de sparsité

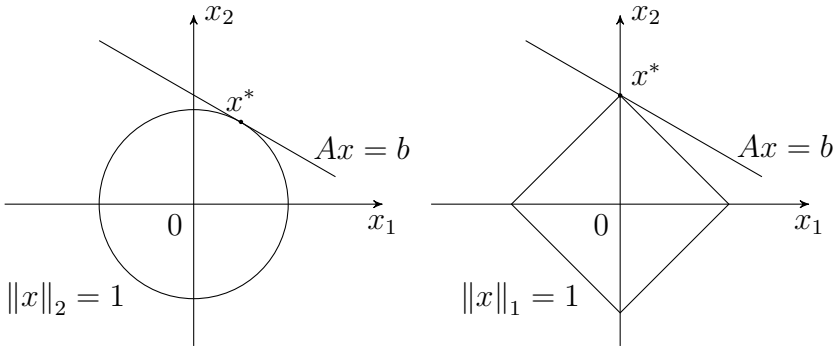
Les solutions à la NMF étant contraintes au quadrant positif, elles présentent souvent naturellement une certaine sparsité. On peut accentuer encore le trait en pénalisant la norme ℓ_1 des endmembers ou des abondances, comme suggéré par Hoyer (2004). La NMF avec sparsité pénalisée s'écrit

$$\min_{W\in\mathbb{R}_+^{m\times k}, H\in\Delta_k^n} \frac{1}{2}\|M - WH\|_2^2 + \lambda \|H\|_1 + \mu \|W\|_1 .$$

On va essayer de donner très sommairement la raison de l'utilisation de la norme ℓ_1. On ne minimise pas avec la norme ℓ_0, qui compte le nombre d'entrées différentes de 0 dans un vecteur car le problème

$$\min \|x\|_0 \text{ tel que } Ax = b$$

est NP-dur. En revanche le même problème avec la norme ℓ_1 est un simple problème de programmation linéaire. Le schéma suivant donne une intuition de pourquoi remplacer la norme ℓ_0 par ℓ_1 fonctionne bien sous certaines conditions.



Une utilisation plus réaliste de ce genre d'outils est le dé-mélange à partir d'une bibliothèque. Plutôt que d'encourager à la fois la sparsité W et H, on pénalise H, en espérant ainsi ne pas utiliser tous les spectres de la bibliothèque pour démêler le cube. Mais la pénalisation se fait coefficient par coefficient, et on ne peut donc pas être sûr qu'un nombre conséquent de lignes de H soit des lignes de 0. C'est pourtant

nécessaire. Ne pas utiliser le k-ème spectre de la librairie revient en effet à ce que la k-ème ligne des abondances ne contiennent que des 0. Pour ce, on remplace la norme ℓ_1 de la matrice H coefficients par coefficients par la somme des normes ℓ_2 de chaque ligne.

4.4.2 Sparsité de la variation totale

La régularisation d'image par variation totale (TV) est une technique couramment utilisée en traitement de l'image, introduite par Rudin *et al.* (1992). L'idée est d'exploiter la cohésion spatiale des images naturelles. Deux pixels voisins ont généralement de bonnes chances d'être relativement semblables. Limiter la variation possible sur une image, tout en essayant de la garder la plus fidèle possible à l'originale, permet de supprimer certains détails en préservant les traits principaux.

Ce type de méthode cherche donc généralement un équilibre entre un terme de fidélité et un terme limitant la variation totale. Une formulation générique pour un filtre dé-bruiteur sur une image $X \in \mathbb{R}^{m\times n}$ pourrait se formuler par exemple

$$\min_y \|Y - X\| + \lambda TV(Y),$$

où TV est le terme pénalisant la variation totale, que l'on définira comme

$$TV(Y) = \sum_{i,j} \left[|y_{i+1,j} - y_{i,j}| + |y_{i,j+1} - y_{i,j}|\right].$$

Pour faciliter la notation, on préferera généralement une écriture matricielle. Pour ce, on pose

$$D_n = \begin{pmatrix} 1 & -1 & & \\ & 1 & -1 & \\ & & \ddots & \ddots \\ & & & 1 & -1 \end{pmatrix}. \tag{4.4}$$

On a ainsi,

$$TV(Y) = \|D_m X\|_1 + \|X D_n\|_1 .$$

Pour factoriser un cube par NMF, on commence généralement par le déplier, pour pouvoir travailler avec une matrice dont chaque colonne correspond à un spectre. Sur cette matrice, étudier la variation totale ligne par ligne permet de mesurer la cohésion spatiale dans une direction. Colonne par colonne on mesure la la variation spectrale. Une fois cette matrice factorisée, la variation totale comme définie précédemment perd son sens. En effet, dans la matrice W représentant les endmembers, il n'y a aucune raison de s'intéresser à la variation ligne par ligne, celle-ci pouvant changer en modifiant l'ordre de concaténation de chaque spectre de base dans W. Seule la variation colonne par colonne a encore un intérêt, elle traduit la régularité spectrale des spectres sélectionnés. La même remarque vaut pour la matrice des abondances H. Ligne par ligne, on mesure la cohésion spatiale. Colonne par colonne, la variation totale est dépendante de l'ordre d'empilement choisi pour les spectres de base. Pour formuler une NMF régularisée par la variation totale, on ne pénalisera donc que la variation de la matrice W colonne par colonne, et celle de H ligne par ligne.

$$\min_{W\in\mathbb{R}_+^{m\times k}, H\in\Delta_k^n} \frac{1}{2}\|M - WH\|_2^2 + \lambda \|D_m W\|_1 + \mu \|H D_n\|_1 . \tag{4.5}$$

L'approche est motivée et détaillée au chapitre 5.

4.4.3 Outils algorithmiques pour la NMF régularisée

L'objectif décrit dans l'équation (4.5) est difficile à optimiser avec un algorithme type descente de gradient. La fonction n'est en effet différentiable ni en W, ni en H. Des méthodes de sous-gradient pourraient aider, mais on préfère ici exposer une autre méthode nous permettant à la fois de résoudre le problème (4.5) ainsi que d'autres problèmes que l'on rencontrera par la suite. Il s'agit des méthodes d'optimisation proximale, comme *forward-backward* ou ADMM par exemple. Ces techniques permettent d'optimiser un objectif composite, avec des garanties de convergence. Nous exposons dans les prochains paragraphes quelques éléments d'analyse convexe nécessaires à la compréhension de ces techniques.

Dualité de Fenchel-Moreau-Legendre

Pour f une fonction de $\mathbb{R}^n$ dans $\mathbb{R}$, on définit le dual de f, noté $f^* : \mathbb{R}^n \to \mathbb{R}$, par

$$f^*(y) = \sup_{x \in \mathbb{R}^n} y^T x - f(x).$$

On définit aussi l'opérateur proximal de f comme

$$\text{prox}_f(y) = \arg\min_x f(x) + \frac{1}{2} \|x - y\|_2^2 .$$

Avant de pouvoir énoncer certaines propriétés concernant ces deux objets, on rappelle brièvement quelques définitions et propriétés concernant le sous-différentiel. Le sous-différentiel d'une fonction f en x est défini comme l'ensemble des sous-gradients, autrement dit

$$\partial f(x) = \{g, g^T(y - x) \leq f(y) - f(x), \forall y \in \text{dom}\, f\}.$$

Plusieurs des propriétés classiques de la dérivation restent vraies, en particulier si $f(x) = g(Ax + b)$ alors $\partial f(x) = A^T \partial g(Ax + b)$. L'intérêt des sous-différentiels est d'exprimer facilement les conditions d'optimalité locale.

$$u = \arg\min_x f(x) \Leftrightarrow 0 \in \partial f(u).$$

De plus, si f est fermée et convexe, la notion est compatible avec la conjugaison

$$y \in \partial f(x) \Leftrightarrow x \in \partial f^*(y) \Leftrightarrow x^T y = f(x) + f^*(y).$$

On a donc

$$\begin{aligned} u = \text{prox}_f\, x &\Leftrightarrow u = \arg\min_y f(y) + \frac{1}{2} \|y - x\|_2^2 \\ &\Leftrightarrow 0 \in \partial f(u) + \{u - x\} \\ &\Leftrightarrow x - u \in \partial f(u) \\ &\Leftrightarrow u \in \partial f^*(x - u) \\ &\Leftrightarrow x - u = \text{prox}_f^*(x) \end{aligned}$$

On vient de démontrer la décomposition de Moreau, qui énonce que

$$y = \text{prox}_f(y) + \text{prox}_{f^*}(y).$$

Ces différents éléments permettent d'écrire différents algorithmes d'optimisation proximale, adaptés aux objectifs composites, comme

$$\min_x f(x) + g(x). \tag{4.6}$$

Forward-backward

Un algorithme possible pour résoudre l'équation (4.6) est la descente de gradient proximale, appelée parfois algorithme forward-backward (Combettes et Pesquet, 2011), où l'on itère

$$x_{k+1} = \text{prox}_{\lambda_k g}\left(x_k - \lambda_k \nabla f(x_k)\right),$$

où λ_k désigne le pas utilisé à la k-ème itération. Une autre possibilité est l'algorithme Douglas-Rachford qui se propose d'itérer

$$\begin{cases} x_{k+1} = \text{prox}_{\lambda f}(y_k) \\ y_{k+1} = y_k + \text{prox}_{\lambda g}\left(2x_{k+1} - y_k\right) - x_k. \end{cases} \tag{4.7}$$

ADMM

ADMM, ou alternating direction method of multipliers (Boyd *et al.*, 2011), permet de résoudre

$$\min_{x,y} f(x) + g(y) \text{ tel que } Ax + By = c. \tag{4.8}$$

Le lagrangien du problème est donné par

$$L(x, y, \lambda) = f(x) + g(y) + \lambda^T \left(Ax + By - c\right). \tag{4.9}$$

Le problème dual (Lagrange) est donc donné par

$$\max_\lambda \min_{x,y} L(x, y, \lambda) = \max_\lambda \left[-c^T\lambda - f^*(-A^T\lambda) - g^*(-B^T\lambda)\right].$$

On applique ensuite Douglas-Rachford à ce problème. Pour ce, on a besoin des opérateurs proximaux de

$$\tilde{g}(\lambda) = c^T\lambda + f^*(-A^T\lambda) \text{ et } \tilde{f}(\lambda) = g^*(-B^T\lambda).$$

On arrive ainsi aux équations décrivant une itération d'ADMM.

$$\begin{cases} x_{k+1} = \arg\min_x L_\rho(x, y_k, \lambda_k), \\ y_{k+1} = \arg\min_y L_\rho(x_{k+1}, y, \lambda_k), \\ \lambda_{k+1} = \lambda_k + \rho(Ax_{k+1} + By_{k+1} - c), \end{cases} \tag{4.10}$$

où $L_\rho(x, y, \lambda)$ est le lagrangien augmenté correspondant à (4.9).

4.4.4 Calcul d'une NMF régularisée

Les différents outils développés au paragraphe précédent permettent la résolution de divers problèmes d'intérêt.

Exemple : régularisation TV

La régularisation TV se prête bien à l'optimisation via ADMM. On rappelle l'objectif type d'un filtre TV en une dimension :

$$\min_x \frac{1}{2} \|x - a\|_F^2 + \mu TV(x),$$

où $a \in \mathbb{R}^{m \times n}$ est le signal à dé-bruiter. En cassant l'objectif en deux, on obtient

$$\min_x \frac{1}{2} \|x - a\|_F^2 + \mu \|y\|_1 \text{ tel que } D_m x = y,$$

où D_m est défini comme dans l'équation (4.4), ce qu'on calcule grâce à ADMM en résolvant alternativement

$$\begin{cases} x_{k+1} = \arg\min_x \frac{1}{2} \|x - a\|_F^2 + \lambda \cdot (D_m x - y_k) + \frac{\rho}{2} \|D_m x - y_k\|_F^2, \\ y_{k+1} = \arg\min_y \mu \|y\|_1 + \lambda \cdot (x_{k+1} - y) + \frac{\rho}{2} \|x_{k+1} - y\|_F^2, \\ \lambda_{k+1} = \lambda_k + \rho(x_{k+1} - y_{k+1}). \end{cases}$$

Ces deux problèmes se résolvent aisément :

$$\begin{cases} x_{k+1} = \frac{1}{1+\rho} (a - \lambda + \rho y_k) \\ y_{k+1} = \text{soft} (x - y + \lambda/\rho, \mu/\rho), \end{cases}$$

où soft désigne la fonction de seuillage doux définie comme l'opérateur proximal de la fonction $x \to \lambda \|x\|_1$ pour $\lambda \in \mathbb{R}^+$. Cet opérateur se calcule aisément coordonnée par coordonnée comme

$$\text{soft}(x, \lambda) = \text{sign}(x - \lambda) \max(|x| - \lambda, 0).$$

Exemple : NMF

La NMF se prête assez mal à l'optimisation via ADMM. Il faut tricher un peu. On l'écrit comme

$$\min \frac{1}{2} \|M - X_1 Y_2\|_F^2 + \chi_+(X_2) + \chi_+(Y_1) \text{ tel que } X_1 = Y_1, X_2 = Y_2.$$

On ne peut pas appliquer ADMM strictement sur cette décomposition, le premier terme dépendant à la fois de X et Y. Toutefois, Zhang a montré dans Zhang (2010) que cette façon de faire convergeait vers un point critique.

Boyd propose dans Boyd *et al.* (2011) de décomposer la NMF comme

$$\min \frac{1}{2} \|M - X_1\|_F^2 + \chi_+(X_2) + \chi_+(Y) \text{ tel que } X_1 = X_2 Y.$$

On sort à nouveau du cadre du strict ADMM, la contrainte s'exprimant non-linéairement en X, Y.

Exemple : NMF-TV

L'un des intérêts de décomposer la NMF pour ADMM est de faciliter l'ajout de nouvelles contraintes, de nouveaux termes de régularisation par exemple. Dans ce paragraphe, on expose très sommairement le problème NMF régularisée par TV. Les détails sont donnés dans le chapitre suivant, qui correspond à un article soumis pour publication rédigé durant cette thèse portant sur le sujet. Contrairement au problème décrit par l'équation (4.5), on s'efforce de ne pas déplier le cube.

$$\min_{\substack{\mathcal{H}\in\Delta_k^{m\times n}\\ W\in\mathbb{R}_+^{o\times k}}} \frac{1}{2}\left\|\mathcal{M}-\mathcal{H}\times_3 W\right\|_F^2+\lambda_s\left\|\mathcal{H}\times_1 D_m\right\|_1+\lambda_s\left\|\mathcal{H}\times_2 D_n\right\|_1+\lambda_t\left\|D_oW\right\|_1,$$

Cette équation est exposée avec force détails au chapitre suivant. On découpe l'équation précédente en

$$\min \frac{1}{2}\left\|\mathcal{M}-\mathcal{Z}_1\times_3 X_1\right\|_F^2+\lambda_s\left\|\mathcal{Z}_2\right\|_1+\lambda_s\left\|\mathcal{Z}_3\right\|_1+\chi_+\left(\mathcal{Z}_4\right)+\lambda_t\left\|X_2\right\|_1+\chi_+\left(X_3\right),$$

satisfaisant les contraintes

$$\begin{cases} X_1=Z_0,\\ X_2=D_oZ_0,\\ X_3=Z_0, \end{cases} \qquad \text{et} \qquad \begin{cases} \mathcal{Z}_1=\mathcal{X}_0\\ \mathcal{Z}_2=\mathcal{X}_0\times_1 D_m,\\ \mathcal{Z}_3=\mathcal{X}_0\times_2 D_n,\\ \mathcal{Z}_4=\mathcal{X}_0. \end{cases}$$

Exemple : NMF-TV, pour abondances seules

Quand on dispose d'une bibliothèque de spectres, on peut décomposer un cube hyper-spectral par rapport aux spectres de cette bibliothèque. Il peut être opportun de demander de cette décomposition qu'elle présente une cohérence spatiale. Contrairement au paragraphe précédent, on peut cette fois décomposer le problème de façon à appliquer strictement ADMM à la résolution du problème. Si $W\in\mathbb{R}^{m\times k}$ est la bibliothèque de k spectres, on cherchera à résoudre

$$\min_{\mathcal{H}\in\mathbb{R}_+^{m\times n\times k}} \frac{1}{2}\left\|\mathcal{M}-\mathcal{H}\times_3 W\right\|_F^2+\lambda_s\left\|\mathcal{H}\times_1 D_m\right\|_1+\lambda_s\left\|\mathcal{H}\times_2 D_n\right\|_1+\sum_k\left\|\mathcal{H}_{\cdot,\cdot,k}\right\|_F^2.$$

Il est possible de décomposer cet objectif de façon similaire au paragraphe précédent.

5

Total variation regularized non-negative matrix factorization for smooth hyperspectral unmixing

Ce chapitre consiste en un article soumis pour publication à *Inverse Problems and Imaging*. Il expose comment intégrer les informations de répartition spatiale des pixels d'un hypercube dans le dé-mélange. Pour ce, nous y exposons comment adjoindre un terme de régularisation par variation totale à l'objectif NMF classique. Les notions de calcul tensoriel esquissées en appendice sont introduites plus en détail section 6.4.

TOTAL VARIATION REGULARIZED NON-NEGATIVE MATRIX FACTORIZATION FOR SMOOTH HYPERSPECTRAL UNMIXING

Adrien Faivre*

Digital Surf, 16 Rue Lavoisier, 25000 Besançon, FRANCE
Email: afaivre@digitalsurf.fr
and
Laboratoire de Mathématiques de Besançon, UMR CNRS 6623,
Université de Bourgogne Franche-Comté,
16 route de Gray, 25030 Besançon cedex, FRANCE

Clément Dombry

Laboratoire de Mathématiques de Besançon, UMR CNRS 6623,
Université de Bourgogne Franche-Comté,
16 route de Gray, 25030 Besançon cedex, FRANCE
Email: clement.dombry@univ-fcomte.fr

Abstract. Hyperspectral analysis has gained popularity over recent years as a way to infer what materials are displayed on a picture whose pixels consist of a mixture of spectral signatures. Computing both signatures and mixture coefficients is known as unsupervised unmixing, a set of techniques usually based on non-negative matrix factorization. Unmixing is a difficult non-convex problem, and algorithms may converge to one out of many local minima, which may be far removed from the true global minimum. Computing this true minimum is NP-hard and seems therefore out of reach. Aiming for interesting local minima, we investigate the addition of total variation regularization terms. Advantages of these regularizers are two-fold. Their computation is typically rather light, and they are deemed to preserve sharp edges in pictures. This paper describes an algorithm for regularized hyperspectral unmixing based on the Alternating Direction Method of Multipliers.

1. Hyperspectral unmixing

Spectral sensors can nowadays acquire electromagnetic intensity with a resolution of thousands of wavebands, spread across an increasing number of pixels. Analyzing hyperspectral data is therefore of growing complexity. This article tackles with hyperspectral unmixing, a matrix factorization technique designed to retrieve spectral signatures of pure materials and corresponding proportions from an hyperspectral image, while enhancing smoothness of both spectrums and mixing proportions. Pure spectral signatures and their mixing proportions are usually respectively referred to as endmembers and abundances throughout the hyperspectral analysis literature.

1991 *Mathematics Subject Classification.* Primary: 65F22, 15A23; Secondary: 65K10.

Key words and phrases. Non-negative matrix factorization, Total variation regularization, Hyperspectral imaging, Constrained optimization, Hyperspectral unmixing.

* Corresponding author: Adrien Faivre.

Spectral signatures are by nature positive, and abundances are by definition positive proportions. These facts lead to a formulation of unmixing as the following constrained optimization problem:

$$\min_{W\in\mathbb{R}_+^{m\times k},H\in\mathbb{R}_+^{k\times n}} \frac{1}{2}\|M-WH\|_F^2, \tag{1}$$

designed to recover k endmembers $W_1,\cdots,W_k$ and their corresponding abundances $H_1,\cdots,H_n$ from n observed spectrums $M_1,\cdots,M_n$. Notations W_i,H_i and M_i respectively stand for the i-th column of matrices W,H and M. The non-negative reals are denoted $\mathbb{R}_+$, and $\mathbb{R}_+^{m\times n}$ stands for the set of $m\times n$ non-negative matrices. The norm $\|\cdot\|_F$ is the standard Frobenius matrix norm.

Problem (1) is known as Non-negative Matrix Factorization (NMF), and is deemed quite difficult. While optimizing with respect to W or H alone is a simple non-negative least squares problem that can be solved in polynomial time, optimizing with both W and H simultaneously turns out to be very hard. Vavasis [14] proved that deciding whether the non-negative rank of a matrix is the same as its rank is NP-complete. Arora et al. [2] showed how under the Exponential Time Hypothesis, there can be no exact algorithm for NMF running in time $(mn)^{o(r)}$. There already exists many algorithms producing approximate solutions to NMF [5]. They range from simple alternating algorithms [10], to more theoretically involved convexifications [9]. In hyperspectral literature, one recurring trend is to try to recover first the endmembers, and only then compute abundances. This is the path followed by VCA [11], SPA [5], SIVM [3], and a few others. Another approach is to add constraints to the NMF problem, aiming for more interesting local minima. The most obvious one we can add is for columns of H to sum to 1. Abundances are indeed supposed to be proportions. Every column of the abundances matrix must therefore belong to the k-simplex. We note:

$$H=\begin{pmatrix}H_1 & H_2 & \cdots & H_n\end{pmatrix}\in\Delta_k^n.$$

Hoyer [7] remarked that the matrices resulting from NMF are usually sparse, and emphasized this by adding sparseness constraints on W or H. More recently, it was considered natural for abundances to display some sort of spatial regularity across the image. Inspired by Total Variation (TV) regularization techniques, Iordache et al. [8] penalized the l_1 norm of the abundances gradient. A related idea from Warren and Osher [15] is to consider that endmembers should also display limited total variation. The main topic of the present paper is to combine both spectral and spatial TV regularization to produce a smoothed NMF.

Adding all these constraints and regularizers together yields a goal functional a little more complicated than the one described in Equation (1). The TV regularized NMF problem reads

$$\min_{W\in\mathbb{R}_+^{m\times k},H\in\Delta_k^n} \frac{1}{2}\|M-WH\|_2^2+\mu\mathrm{TV}(H)+\lambda\mathrm{TV}(W). \tag{2}$$

This problem can fortunately be split into easier sub-problems, through techniques such as the Alternating Direction Method of Multipliers (ADMM). See the seminal paper of Boyd et al. [4] for an in-depth treatment of ADMM techniques.

The ADMM algorithm allows for the optimization of composite problems of the form

$$\min f(x)+g(z) \text{ such that } Ax+Bz=c. \tag{3}$$

To achieve this, first form the augmented Lagrangian

$$L_\rho(x,z,y) = f(x) + g(z) + y^T(Ax + Bz - c) + \frac{\rho}{2}\|Ax + Bz - c\|_2^2.$$

Then, using updates

$$\begin{cases} x_{k+1} = \text{argmin}_x L_\rho(x, z_k, y_k), \\ z_{k+1} = \text{argmin}_z L_\rho(x_{k+1}, z, y_k), \\ y_{k+1} = y_k + \rho(Ax_{k+1} + Bz_{k+1} - c), \end{cases} \tag{4}$$

one is guaranteed to get a local optimum for problem (3) under relatively mild assumptions. In this paper we will instead use the equivalent shortened version

$$\begin{cases} x_{k+1} = \text{argmin}_x f(x) + \frac{\rho}{2}\|Ax + Bz_k - c + u_k\|_2^2, \\ z_{k+1} = \text{argmin}_z g(z) + \frac{\rho}{2}\|Ax_{k+1} + Bz - c + u_k\|_2^2, \\ u_{k+1} = u_k + (Ax_{k+1} + Bz_{k+1} - c), \end{cases}$$

that follows from Equation (4) by introducing $u_k = y/\rho$.

NMF hardly fits the ADMM framework, but Zhang [17] studied the convergence of the following splitting scheme

$$\min_{X_1,X_2,Z_1,Z_2} \|M - X_1Z_2\|_F^2 + \chi_+(X_2) + \chi_+(Z_1) \text{ such that } \begin{cases} X_1 = Z_1, \\ X_2 = Z_2. \end{cases} \tag{5}$$

In Equation (5), χ_+ denotes the characteristic function of matrices with positive entries, valued 0 for matrices with no negative entries, and ∞ for any other. The sub-problem in X_1 is a least square problem, and the one in X_2 can be solved using a projection on the positive orthant. Sub-problems in Z_1, Z_2 can be dealt with similarly. These four sub-problems can be solved efficiently. The difference with standard ADMM formulation is that it does not seem possible to properly split Equation (1) as the sum of two functions.

One advantage of the ADMM technique is that adding regularization terms is rather straightforward, enabling us to interleave TV regularization steps between every NMF factor update. ADMM algorithms were already shown to be quite successful for TV denoising problems [13].

In Section 2, we present how classical ADMM algorithms for image TV denoising can be adapted to hyperspectral images, using realistic boundary conditions. In Section 3, we derive an efficient ADMM algorithm for TV regularized NMF.

2. Total Variation Denoising

2.1. **Neuman boundary conditions.** The penalization of total variation is a popular technique used to remove noise, introduced by Rudin et al [13]. The idea is that a pure signal should have limited variation. Typically, the norm of the discrete gradient is penalized, while faithfulness to the original signal is promoted. Corresponding optimization problem for 1-dimensional signal $y \in \mathbb{R}^n$ reads

$$\min_x \frac{1}{2}\|y - x\|_2^2 + \lambda\|D_n x\|_p, \tag{6}$$

where D_n denotes a discrete differentiation operator for signals of length n, and λ is a parameter controlling the strength of the denoising. Using $p = 1$ leads to anisotropic denoising, and $p = 2$ to isotropic denoising. We will focus on the case $p = 1$. Problem (6) can be solved efficiently using ADMM. It can indeed be split as

$$\min_{x,z} \frac{1}{2}\|y-x\|_2^2 + \lambda\|z\|_1 \text{ such that } D_n x = z.$$

The sub-problem for z reads

$$\min_z \lambda\|z\|_1 + \frac{\rho}{2}\|D_n x - z + u\|\,2^2, \tag{7}$$

and can be solved efficiently. Let $\text{soft}(x,\lambda) = \text{sgn}(x)(|x|-\lambda)_+$ denote the soft-thresholding operator for real x. The solution of sub-problem (7) is then given by

$$z_{k+1} = \text{soft}\left(D_n x + u, \frac{\lambda}{\rho}\right),$$

where soft is applied coefficient-wise. The x update computation is harder, as it requires to solve

$$(I_n + \rho D_n^T D_n)x = y + \rho(z-u), \tag{8}$$

which can be quite difficult for large values of n. One common way to handle this is to pretend that the signal has some sort of specific boundaries. Periodic boundaries are for instance often used as they allow to represent operator D_n as a circulant matrix. This enables the use of Fast Fourier Transform (FFT) to solve Equation (8), which is very efficient in practice. However, real images are usually not periodic, and we will therefore focus on slightly more complicated boundary conditions called Neuman boundaries.

Those conditions express that the gradient is zero on one side of the signal. Corresponding gradient operator reads

$$D_n = \begin{pmatrix} 1 & -1 & & & \\ & 1 & -1 & & \\ & & \ddots & \ddots & \\ & & & 1 & -1 \end{pmatrix}.$$

In order to solve Equation (8), we need to invert matrix $L_n = D_n^T D_n$, which is a lengthy computation using standard dense matrix inversion algorithms. But L_n is a tridiagonal matrix with a very specific structure

$$L_n = \begin{pmatrix} 1 & -1 & & & \\ -1 & 2 & -1 & & \\ & \ddots & \ddots & \ddots & \\ & & -1 & 2 & -1 \\ & & & -1 & 1 \end{pmatrix}. \tag{9}$$

One key property enabling us to inverse L_n swiftly is that Toeplitz-plus-Hankel matrices can be diagonalized by Discrete Cosine Transform (DCT) operators, as shown by Ng et al. [12]. There are many flavors of DCT. We will focus on DCT-II, as it is the most common one. The DCT-II matrix is given by

$$(\text{DCT}_n^{\text{II}})_{i,j} = \cos i(j+\frac{1}{2})\frac{\pi}{n},$$

which is not orthogonal. Its inverse is known as the DCT-III matrix. The diagonalization of the squared gradient operator L_n from Equation (9) reads

$$\mathrm{DCT}_n^{\mathrm{II}} L_n \mathrm{DCT}_n^{\mathrm{III}} = \mathrm{diag}(s_1, \cdots, s_n),$$

where the diagonal values $s_1, \cdots, s_n$ can be recovered from

$$\mathrm{DCT}_n^{\mathrm{II}} L_n e_1 = \mathrm{diag}(s_1, \cdots, s_n) \mathrm{DCT}_n^{\mathrm{II}} e_1,$$

with e_1 the first vector of the canonical basis of $\mathbb{R}^n$. We get

$$s_i = \frac{\cos(\frac{\pi i}{2n}) - \cos(\frac{3\pi i}{2n})}{\cos(\frac{\pi i}{2n})}. \tag{10}$$

2.2. **Discrete differentiation in 3-dimensions.** Hypercubes are multi-indices tables, and a few notions of tensor calculus can be useful to derive update rules for TV regularization. See Appendix for a brief overview of those rules. The proposed unmixing algorithm is based on spatial, or even spectral-spatial regularization. For one-dimensional problems we introduced operator D_n in Section 2.

$$D_n x = (x_i - x_{i+1})_{1 \leq i \leq n}.$$

Higher orders come with more intricate formulas, especially when unfolding the cube. A synthetic and efficient way to write down multidimensional discrete differentiation is through tensor contraction. For notational simplicity, we focus on the 3-dimensional case that corresponds to hyperspectral images. Our discussion remains valid for any other dimension. For a cube $\mathcal{Y} \in \mathbb{R}^{m \times n \times o}$, differentiation on the first mode can be written as

$$\mathcal{Y} \times_1 D_m = D_m Y^{(1)},$$

where $Y^{(1)}$ denotes the first unfolding of $\mathcal{Y}$ (see Appendix for more details).

Matricization enables us to simply state 3-dimensional anisotopic TV as

$$\mathrm{TV}(\mathcal{Y}) = \|\mathcal{Y} \times_1 D_m\|_1 + \|\mathcal{H} \times_2 D_n\|_1 + \|\mathcal{H} \times_3 D_o\|_1.$$

Combining matricization with ADMM theory, we then derive a simple spectral-spatial filtering method close in spirit to the one described by Aggarwal [1]. But instead of using a conjugate-gradient type method for solving sparse linear equations every other update, we use two 3-dimensional DCT.

Such a filter could for instance read

$$\min_{\mathcal{X}} \frac{1}{2} \|\mathcal{Y} - \mathcal{X}\|_F^2 + \lambda_s \|\mathcal{X} \times_1 D_m\|_1 + \lambda_s \|\mathcal{X} \times_2 D_n\|_1 + \lambda_t \|\mathcal{X} \times_3 D_o\|_1.$$

We split last problem as

$$\min_{\mathcal{X}, \mathcal{Z}_1, \mathcal{Z}_2, \mathcal{Z}_3} \frac{1}{2} \|\mathcal{Y} - \mathcal{X}\|_F^2 + \lambda_s \|\mathcal{Z}_1\|_1 + \lambda_s \|\mathcal{Z}_2\|_1 + \lambda_t \|\mathcal{Z}_3\|_1$$

subject to constraints

$$\begin{cases} \mathcal{Z}_1 = \mathcal{X} \times_1 D_m, \\ \mathcal{Z}_2 = \mathcal{X} \times_2 D_n, \\ \mathcal{Z}_3 = \mathcal{X} \times_3 D_o. \end{cases}$$

In order to compute the zeros the gradient with respect to $\mathcal{X}$, we need to solve a generalized Sylvester equation

$$\mathcal{X} + \rho\mathcal{X} \times_1 L_m + \rho\mathcal{X} \times_2 L_n + \rho\mathcal{X} \times_3 L_o = \mathcal{B}, \tag{11}$$

where

$$\mathcal{B} = \mathcal{Y} + \rho(\mathcal{Z}_1 - \mathcal{B}_1) \times_1 D_m^T + \rho(\mathcal{Z}_2 - \mathcal{B}_2) \times_1 D_n^T + \rho(\mathcal{Z}_3 - \mathcal{B}_3) \times_1 D_o^T.$$

The easiest way to solve the classical Sylvester matrix equation is to vectorize the unknown and use Kronecker products to produce an equivalent one. Applying this idea to Equation (11) yields

$$A \operatorname{vec}\left(X^{(1)}\right) = \operatorname{vec}\left(B^{(1)}\right), \tag{12}$$

with

$$A = \left[I_{mno} + \rho\left(I_o \otimes I_n \otimes L_m + I_o \otimes L_n \otimes I_m + L_o \otimes I_n \otimes I_m\right)\right].$$

The interested reader can report to the Appendix to see details concerning the previous operation. Matrix A is potentially enormous and solving Equation (12) may seem impossible at first. However, noticing that A can be written as a Kronecker sum $I_{mno} + \rho L_o \oplus L_n \oplus L_m$, one can diagonalize A as

$$A = \mathrm{DCT}^{\mathrm{II}}_{o,n,m}\left[I_{mno} + \rho(S_o \oplus S_n \oplus S_m)\right]\mathrm{DCT}^{\mathrm{III}}_{o,n,m},$$

where $\mathrm{DCT}^{\mathrm{II}}_{o,n,m}$ is the 3-dimensionnal DCT-II defined as $\mathrm{DCT}^{\mathrm{II}}_o \otimes \mathrm{DCT}^{\mathrm{II}}_n \otimes \mathrm{DCT}^{\mathrm{II}}_m$, and $\mathrm{DCT}^{\mathrm{III}}_{o,n,m}$ is defined similarly. Matrices S_o, S_n and S_m are diagonal matrices whose values are given by Equation (10).

Assuming Neumann boundary conditions, we can now solve efficiently Equation (11) using two 3-dimensional DCT only.

$$\mathcal{X} = \mathrm{DCT}^{\mathrm{III}}_{o,n,m}\left(\frac{\mathrm{DCT}^{\mathrm{II}}_{o,n,m}(\mathcal{B})}{\rho\mathcal{S} + 1}\right), \tag{13}$$

where $\mathcal{S}$ is defined as in (10) as

$$s_{i,j,k} = \frac{\cos(\frac{\pi i}{2m}) - \cos(\frac{3\pi i}{2m})}{\cos(\frac{\pi i}{2m})} + \frac{\cos(\frac{\pi j}{2n}) - \cos(\frac{3\pi j}{2n})}{\cos(\frac{\pi j}{2n})} + \frac{\cos(\frac{\pi k}{2o}) - \cos(\frac{3\pi k}{2o})}{\cos(\frac{\pi k}{2o})}. \tag{14}$$

Tensors $\mathcal{Z}_1, \mathcal{Z}_2, \mathcal{Z}_3$ are then readily updated according to

$$\begin{cases} \mathcal{Z}_1 = \operatorname{soft}\left(\mathcal{X} \times_1 D_m + \mathcal{U}_1, \frac{\lambda_s}{\rho}\right), \\ \mathcal{Z}_2 = \operatorname{soft}\left(\mathcal{X} \times_2 D_n + \mathcal{U}_2, \frac{\lambda_s}{\rho}\right), \\ \mathcal{Z}_3 = \operatorname{soft}\left(\mathcal{X} \times_3 D_o + \mathcal{U}_3, \frac{\lambda_t}{\rho}\right), \end{cases}$$

where the soft-thresholding operator is applied coefficient-wise. The $\mathcal{X}$ and $\mathcal{Z}$ updates are relatively simple. The hardest one, the $\mathcal{X}$ update, is computable in $O(mno\log(mno))$. Despite consisting of these simple steps, the algorithm is rather slow for big cubes. We improve on this issue in Section 3 by regularizing a smaller tensor than $\mathcal{X}$ obtained through factorization.

3. Non-negative matrix factorization

TV regularization is able to preserve discontinuities in images and is often used in image treatment applications, for instance when facing debluring problems [16, 12] that would otherwise be numerically unstable. NMF is an ill-posed problem [5] for which solutions are usually not unique. We hope that adding TV regularization to the problem can lead to better solutions. In this section, we show that TV denoising can be integrated to the unmixing phase without slowing down convergence significantly.

We have to modify slightly what was introduced in Section 2 in order to fit the NMF framework. Indeed, the total variation on the third mode does not mean anything any meaning any more. It could increase or decrease simply by changing the order in which we stack the endmembers in matrix W. That is why we will not focus on the spectral variation of abundances, but will rather add a TV-regularization term on each endmember instead. Abundances will be regularized slice by slice with 2-dimensional TV.

More precisely, we are going to devise an algorithm able to solve the following problem:

$$\min_{\substack{\mathcal{H}\in\Delta_k^{m\times n}\\ W\in\mathbb{R}_+^{o\times k}}} \frac{1}{2}\left\|\mathcal{M}-\mathcal{H}\times_3 W\right\|_F^2+\lambda_s\left\|\mathcal{H}\times_1 D_m\right\|_1+\lambda_s\left\|\mathcal{H}\times_2 D_n\right\|_1+\lambda_t\left\|D_oW\right\|_1,$$

which is a 3-dimensional equivalent to problem (2).

As in Section 2, we start by splitting the problem as:

$$\min \frac{1}{2}\left\|\mathcal{M}-\mathcal{Z}_1\times_3 X_1\right\|_F^2+\lambda_s\left\|\mathcal{Z}_2\right\|_1+\lambda_s\left\|\mathcal{Z}_3\right\|_1+\chi_+\left(\mathcal{Z}_4\right)+\lambda_t\left\|X_2\right\|_1+\chi_+\left(X_3\right), \tag{15}$$

with the following constraints:

$$\begin{cases} X_1=Z_0,\\ X_2=D_oZ_0,\\ X_3=Z_0, \end{cases} \qquad \text{and} \qquad \begin{cases} \mathcal{Z}_1=\mathcal{X}_0\\ \mathcal{Z}_2=\mathcal{X}_0\times_1 D_m,\\ \mathcal{Z}_3=\mathcal{X}_0\times_2 D_n,\\ \mathcal{Z}_4=\mathcal{X}_0. \end{cases}$$

We now list every update for solving problem (15).

- The first sub-problem to solve is

$$\min_{\mathcal{X}_0}\begin{pmatrix} \frac{\rho}{2}\left\|\mathcal{Z}_1-\mathcal{X}_0+\mathcal{U}_4\right\|_F^2+\\ \frac{\rho}{2}\left\|\mathcal{Z}_2-\mathcal{X}_0\times_1 D_m+\mathcal{U}_5\right\|_F^2+\\ \frac{\rho}{2}\left\|\mathcal{Z}_3-\mathcal{X}_0\times_2 D_n+\mathcal{U}_6\right\|_F^2+\\ \frac{\rho}{2}\left\|\mathcal{Z}_4-\mathcal{X}_0+\mathcal{U}_7\right\|_F^2 \end{pmatrix}.$$

  Nullity of gradient implies

$$2\mathcal{X}_0+\mathcal{X}_0\times_1 D_mD_m^T+\mathcal{X}_0\times_2 D_nD_n^T=\mathcal{B},$$

  with $\mathcal{B}=\mathcal{Z}_1+\mathcal{U}_4+\mathcal{Z}_2\times_1 D_m^T+\mathcal{U}_5+\mathcal{Z}_3\times_2 D_n^T+\mathcal{U}_6+\mathcal{Z}_4+\mathcal{U}_7$. As in Section 2, we use a discrete cosine transform to solve this Sylvester equation. We

get

$$\mathcal{X}_0 = \mathrm{DCT}^{\mathrm{III}}_{k,n,m}\left(\frac{\mathrm{DCT}^{\mathrm{II}}_{k,n,m}(\mathcal{B})}{\rho\mathcal{S}+1}\right), \tag{16}$$

with $s_{i,j,k} = s_{i,j,0}$ adapted to the 2-dimensional case from Equation (14).

- The second sub-problem is an order-3 least-squares problem, and reads

$$\min_{X_1} \frac{1}{2}\left\|\mathcal{M} - \mathcal{Z}_1 \times_3 X_1\right\|_F^2 + \frac{\rho}{2}\left\|X_1 - Z_0 + U_1\right\|_F^2.$$

  A third mode unfolding of the previous equation gets us a classical least squares problem

$$\min_{X_1} \frac{1}{2}\left\|M^{(3)} - X_1 Z_1^{(3)}\right\|_F^2 + \frac{\rho}{2}\left\|X_1 - Z_0 + U_1\right\|_F^2.$$

  The update is then given by

$$X_1 = \left(M^{(3)} {Z_1^{(3)}}^T + \rho\left(Z_0 - U_1\right)\right)\left(Z_1^{(3)} {Z_1^{(3)}}^T + \rho I_k\right)^{-1}.$$

- The third sub-problem reads:

$$\min_{X_2} \lambda_t \left\|X_2\right\|_1 + \frac{\rho}{2}\left\|X_2 - D_o Z_0 + U_2\right\|_F^2.$$

  It is a simple proximal problem, with a straightforward solution, that reads

$$X_2 = \mathrm{soft}\left(D_o Z_0 - U_2, \frac{\lambda_t}{\rho}\right)$$

- The fourth sub-problem reads

$$\min_{X_3} \chi_+(X_3) + \frac{\rho}{2}\left\|X_3 - Z_0 + U_3\right\|_F^2.$$

  The solution is given by the projection Π_+ on the positive orthant applied coefficient-wise to the difference between Z_0 and U_3:

$$X_3 = \Pi_+\left(Z_0 - U_3\right).$$

- The fifth sub-problem reads

$$\min_{Z_0} \frac{\rho}{2}\left\|X_1 - Z_0 + U_1\right\|_F^2 + \frac{\rho}{2}\left\|X_2 - D_o Z_0 + U_2\right\|_F^2 + \frac{\rho}{2}\left\|X_3 - Z_0 + U_3\right\|_F^2.$$

  It is solved by

$$Z_0 = \left(D_o^T D_o + 2I_k\right)^{-1}\left(X_1 + U_1 + D_o^T(X_2 + U_2) + X_3 + U_3\right).$$

- The sixth sub-problem is again a least squares-problem :

$$\min_{\mathcal{Z}_1} \frac{1}{2}\left\|\mathcal{M} - \mathcal{Z}_1 \times_3 X_1\right\|_F^2 + \frac{\rho}{2}\left\|\mathcal{Z}_1 - \mathcal{X}_0 + \mathcal{U}_4\right\|_F^2.$$

  This yields

$$\mathcal{Z}_1 = \left[\mathcal{M} \times_3 X_1^T + \rho(\mathcal{X}_0 - \mathcal{U}_4)\right] \times_3 \left(X_1^T X_1 + \rho I_k\right)^{-1}. \tag{17}$$

- The seventh and eighth sub-problem read respectively

$$\min_{\mathcal{Z}_2} \lambda_s \|\mathcal{Z}_2\|_1 + \frac{\rho}{2} \|\mathcal{Z}_2 - \mathcal{X}_0 \times_1 D_m + \mathcal{U}_5\|_F^2 ,$$

$$\min_{\mathcal{Z}_3} \lambda_s \|\mathcal{Z}_3\|_1 + \frac{\rho}{2} \|\mathcal{Z}_3 - \mathcal{X}_0 \times_2 D_n + \mathcal{U}_6\|_F^2 .$$

  Solutions for both problems are given by

$$\mathcal{Z}_2 = \text{soft}\left(\mathcal{X}_0 \times_1 D_m - \mathcal{U}_5, \frac{\lambda_s}{\rho}\right),$$

$$\mathcal{Z}_3 = \text{soft}\left(\mathcal{X}_0 \times_2 D_n - \mathcal{U}_6, \frac{\lambda_s}{\rho}\right).$$

- The ninth sub-problem is another projection:

$$\min_{\mathcal{Z}_4} \chi_+(\mathcal{Z}_4) + \frac{\rho}{2} \|\mathcal{Z}_4 - \mathcal{X}_0 + \mathcal{U}_7\|_F^2 .$$

  The update is thus given by

$$\mathcal{Z}_4 = \Pi_+ (\mathcal{X}_0 - \mathcal{U}_7) .$$

The longest update is the one involving the tensor contraction, described in Equation (17). The computation of ${X_1}^T M^{(3)}$ indeed takes $O(kmno)$ steps to compute. The one involving the DCT of the abundance matrix, described by Equation (16), can be computed in $O(mnk \log(mnk))$ and is therefore usually lighter (depending on o). This is also lighter that the DCT computations described in Section 2, where Equation (13) required a $O(mno \log(mno))$ computation.

4. Discussion and final remarks

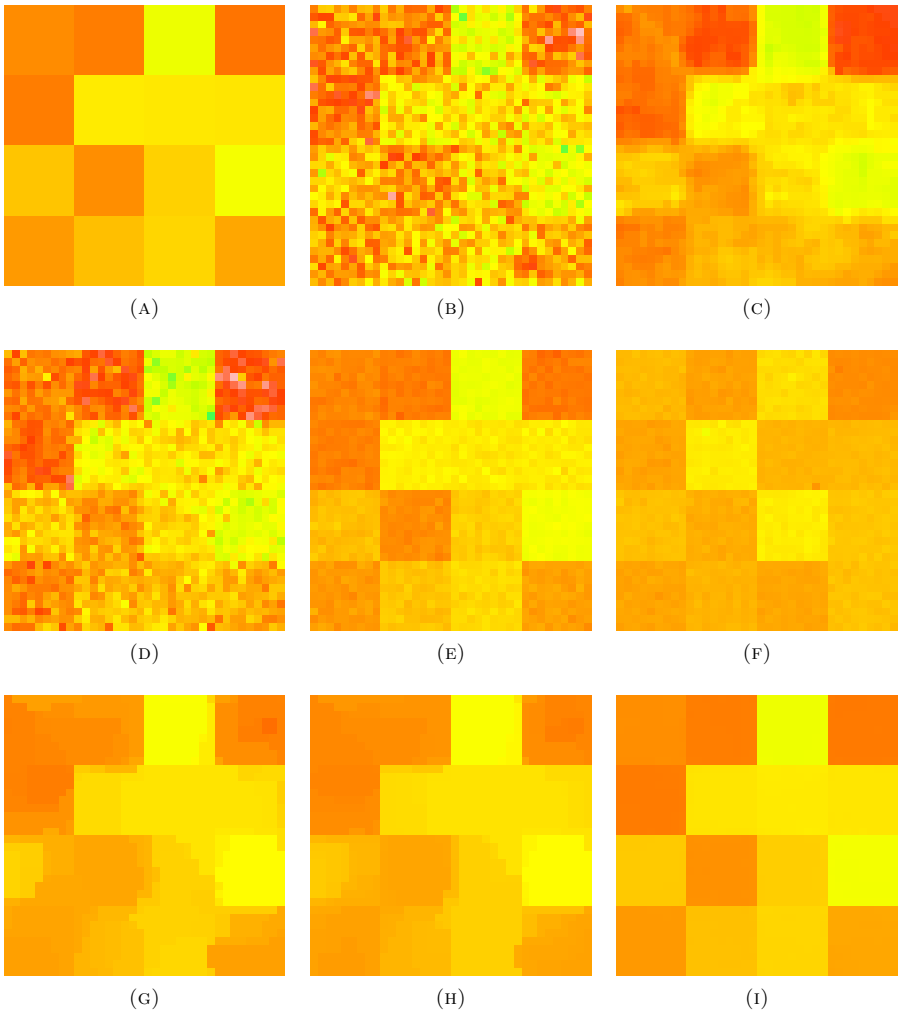
We were able to add TV regularization to the ADMM algorithm for NMF without slowing it down significantly. Resulting unmixings seem to benefit from introduced regularization.

To illustrate the denoising techniques proposed in this article, we generate a synthetic hypercube, and apply the 3-dimensional TV denoising introduced in Section 2, and the NMF-TV algorithm from Section 3. NMF was not designed to remove noise. Its goal is to recover non-negative factors from a matrix. However, these factors are usually not unique, and can be quite different from the ones used to generate an example. Instead of assessing factors, we therefore focus on their product.

We generate a cube consisting of 16 homogenous regions, each consisting of a randomly chosen mixture of 5 randomly chosen spectrums in the USGS 1995 Library[1]. We then add white noise, and try out our techniques.

More precisely, we start by building true endmembers matrix $W_0 \in \mathbb{R}^{224\times 5}$ by stacking up 5 endmembers randomly selected amongst the 498 ones available in the library. We then build a first abundance matrix $H_0 \in \mathbb{R}^{5\times 16}$ where every column is randomly drawn from a Dirichlet distribution. The matrix product $W_0 H_0$ defines a matrix $M_0 \in \mathbb{R}^{224\times 16}$ whom we fold into hypercube $\mathcal{M}_0 \in \mathbb{R}^{4\times 4\times 224}$. To induce spatial regularity, we then replace each pixel x of $\mathcal{M}_0$ by a 9×9 block of copies of x. We get a cube of size $36 \times 36 \times 224$. We finally obtain hypercube $\mathcal{M}$ by adding gaussian noise.

[1]Available online at https://speclab.cr.usgs.gov/spectral.lib06/



(A) (B) (C)

(D) (E) (F)

(G) (H) (I)

FIGURE 1. Simulation results for band 132. (1a) Simulated hypercube before adding noise, (1b) Simulated hypercube after adding noise, (1c) Median denoising, (1d) Wiener denoising, (1e) Lee and Seung's NMF used for denoising, (1f) SPA used for denoising, (1g) TV denoising with $\lambda_s = 0.05$, and $\lambda_t = 0$, (1h) TV denoising with $\lambda_s = 0.05$, and $\lambda_t = 0.01$, (1i) NMF-TV denoising with $\lambda_s = 2.0$, and $\lambda_t = 0.1$.

$$\mathcal{M} = \mathcal{M}_0 + \mathcal{N},$$

where every coefficient of $\mathcal{N} \in \mathbb{R}^{36\times36\times224}$ was randomly chosen according to the centered normal distribution.

We compare proposed techniques with 4 relatively common algorithms, a median filter, a Wiener filter, and two NMF algorithms : the seminal multiplicative update

scheme from Lee and Seung [10], and the Successive Projection Algorithm (SPA) described by Gillis in [5]. The median filter replaces each pixel by the median of its $3 \times 3 \times 3$ neighborhood. The Wiener filter changes every pixel value x according to

$$y = \begin{cases} \frac{\sigma^2}{\sigma_x^2} m_x + \left(1 - \frac{\sigma^2}{\sigma_x^2}\right) x \text{ if } \sigma_x^2 \geq \sigma^2, \\ m_x \text{ if } \sigma_x^2 < \sigma^2, \end{cases} \tag{18}$$

where m_x and σ_x are the local mean and variance of the $3 \times 3 \times 3$ neighborhood of pixel x.

The median filter, Wiener filter and TV filter all give a result that we can directly compare to our synthesized example. The NMF algorithms results consist of two factors W and H. We use their product WH for comparisons.

ADMM parameter ρ can theoretically be set to any value without compromising convergence. In practice, a good choice for ρ is important, as it has an influence on the algorithm's speed. We experimentally determined that setting $\rho = 10$ allows our algorithm to converge reasonably fast. Parameters λ and ρ controlling the desired strength of TV-regularization were also set experimentally.

Results are displayed in Figure 1. Only one slice of the resulting hypercubes is displayed. The median and Wiener filter are not able to recover precisely the boundaries of every region as can be seen in Figure 1c and 1d. The classical NMF algorithms are not designed specifically with denoising in mind. However, their result displayed in Figure 1e and 1f show that they are quite successful at recovering sharp boundaries between every region. They however fail at producing smooth regions. Figure 1g results from a spatial only TV regularization, whereas Figure 1h is regularized by spatial–spectral TV. A careful comparison of both figures reveals that adding a spectral component to the TV denoising algorithm described in Section 2 seems to slightly improve the result. The NMF-TV algorithm recovers almost perfectly the original cube, as can be seen in Figure 1i.

To conclude, we state an interesting direction for future research. The base ingredient we used throughout our paper is a simple ADMM algorithm. We could instead try to use the Nesterov accelerated scheme described by Goldstein et al. in [6]. This would most likely speed up convergence.

References

1. Hemant Kumar Aggarwal and Angshul Majumdar, *Hyperspectral image denoising using spatio-spectral total variation*, IEEE Geoscience and Remote Sensing Letters **13** (2016), no. 3, 442–446.
2. Sanjeev Arora, Rong Ge, Ravindran Kannan, and Ankur Moitra, *Computing a nonnegative matrix factorization–provably*, Proceedings of the forty-fourth annual ACM symposium on Theory of computing, ACM, 2012, pp. 145–162.
3. Christian Bauckhage, *A purely geometric approach to non-negative matrix factorization.*, LWA, 2014, pp. 125–136.
4. Stephen Boyd, Neal Parikh, Eric Chu, Borja Peleato, and Jonathan Eckstein, *Distributed optimization and statistical learning via the alternating direction method of multipliers*, Foundations and Trends in Machine Learning **3** (2011), no. 1, 1–122.
5. Nicolas Gillis, *The why and how of nonnegative matrix factorization*, Regularization, Optimization, Kernels, and Support Vector Machines **12** (2014), 257.
6. Tom Goldstein, Brendan O'Donoghue, Simon Setzer, and Richard Baraniuk, *Fast alternating direction optimization methods*, SIAM Journal on Imaging Sciences **7** (2014), no. 3, 1588–1623.
7. Patrik O Hoyer, *Non-negative matrix factorization with sparseness constraints*, The Journal of Machine Learning Research **5** (2004), 1457–1469.

8. Marian-Daniel Iordache, José M Bioucas-Dias, and Antonio Plaza, *Total variation spatial regularization for sparse hyperspectral unmixing*, Geoscience and Remote Sensing, IEEE Transactions on **50** (2012), no. 11, 4484–4502.
9. Vijay Krishnamurthy and Alexandre d'Aspremont, *Convex algorithms for nonnegative matrix factorization*, arXiv preprint arXiv:1207.0318 (2012).
10. Daniel D Lee and H Sebastian Seung, *Algorithms for non-negative matrix factorization*, Advances in neural information processing systems, 2001, pp. 556–562.
11. José MP Nascimento and José MB Dias, *Vertex component analysis: A fast algorithm to unmix hyperspectral data*, IEEE transactions on Geoscience and Remote Sensing **43** (2005), no. 4, 898–910.
12. Michael K Ng, Raymond H Chan, and Wun-Cheung Tang, *A fast algorithm for deblurring models with neumann boundary conditions*, SIAM Journal on Scientific Computing **21** (1999), no. 3, 851–866.
13. Leonid I Rudin, Stanley Osher, and Emad Fatemi, *Nonlinear total variation based noise removal algorithms*, Physica D: Nonlinear Phenomena **60** (1992), no. 1, 259–268.
14. Stephen A Vavasis, *On the complexity of nonnegative matrix factorization*, SIAM Journal on Optimization **20** (2009), no. 3, 1364–1377.
15. R Warren and S Osher, *Hyperspectral unmixing by the alternating direction method of multipliers*, Inverse Problems and Imaging **14** (2015), no. 3, 00–00.
16. Junfeng Yang, Wotao Yin, Yin Zhang, and Yilun Wang, *A fast algorithm for edge-preserving variational multichannel image restoration*, SIAM Journal on Imaging Sciences **2** (2009), no. 2, 569–592.
17. Yin Zhang, *An alternating direction algorithm for nonnegative matrix factorization*, (2010).

Appendix

For an order 3 tensor $\mathcal{T} \in \mathbb{R}^{m\times n\times o}$ the mode 1-fiber corresponding to indices (j,k) is the vector $(T_{i,j,k})_{1\leq i\leq m}$, the 2-fiber corresponding to (i,k) is defined as $(T_{i,j,k})_{1\leq j\leq n}$ and mode 3-fibers are defined similarly as $(T_{i,j,k})_{1\leq k\leq n}$. Mode n matricization, also known as unfolding, of tensor $\mathcal{T}$ transforms a tensor of any order in a matrix, by stacking its n-fibers in a precise order. The mode k unfolding of tensor $\mathcal{T}$ is denoted $T^{(k)}$. A mode n tensor contraction with a matrix is the generalization of the usual matrix product. The notion is probably better understood through an example. Let $\mathcal{T}$ be an order 3 tensor, and X be a matrix. The mode 2 contraction of $\mathcal{T}$ and X, denoted $\mathcal{T}\times_2 X$ is defined as

$$(\mathcal{T}\times_2 X)_{i,l,k} = \sum_l T_{i,l,k}X_{l,j}.$$

Tensor contractions, unfoldings, and matrix product are linked by the following property :

$$\mathcal{U} = \mathcal{T}\times_k X \Leftrightarrow U^{(k)} = XT^{(k)}.$$

Multiple tensor contractions can be expressed with Kronecker products, thanks to one of its fundamental properties

$$\left(B^T\otimes A\right)\mathrm{vec}(X) = \mathrm{vec}(AXB) = \mathrm{vec}(C).$$

This fact can be used to prove that the following propositions are equivalent :

(i) $\mathcal{Y} = \mathcal{X}\times_1 A^1\times_2 A^2\times_3\cdots\times_n A^n$,
(ii) $Y^{(k)} = A^kX^{(k)}\left(A^n\otimes\cdots\otimes A^{k+1}\otimes A^{k-1}\otimes\cdots\otimes A^1\right)^T$,
(iii) $\mathrm{vec}(Y^{(k)}) = \left(A^n\otimes\cdots\otimes A^{k+1}\otimes A^{k-1}\otimes\cdots\otimes A^1\otimes A^k\right)\mathrm{vec}(X^{(k)})$.

Last property lets us derive successively to the vectorization of Equation (11) in Section 2 :

$$B^{(1)} = (I_{mno} + \rho L_o \otimes I_n \otimes I_m + \rho I_o \otimes L_n \otimes I_m + \rho I_o \otimes I_n \otimes L_m)X^{(1)}.$$

Kronecker sums are usually defined by

$$A \oplus B = A \otimes I_n + I_m \otimes B.$$

This definition can be extended to

$$A \oplus B \oplus C = A \otimes I_n \otimes I_k + I_m \otimes B \otimes I_k + I_m \otimes I_n \otimes C.$$

Note that Kronecker sums are associative but do not commute.

A useful fact about Kronecker sums and products is that they behave well with eigenvalue decompositions. Specifically, if $A = Q_A D_A Q_A^T$ and $B = Q_B D_B Q_B^T$ are the eigenvalue decompositions of matrices A and B, then

$$A \otimes B = (Q_A \otimes Q_B)(D_A \otimes D_B)(Q_A \otimes Q_B)^T,$$
$$A \oplus B = (Q_A \otimes Q_B)(D_A \oplus D_B)(Q_A \otimes Q_B)^T,$$

are the respective eigen-decompositions of $A \otimes B$ and $A \oplus B$.

6

Autres problèmes rencontrés

Plusieurs autres problèmes ont été rencontrés lors de cette thèse. Nous en présentons quelques uns dans ce chapitre, retenus en fonction de leur intérêt mathématique :

- l'analyse en composantes principales robuste,
- la réduction non-linéaire de la dimension,
- la régularisation spatiale du *clustering*,
- la diagonalisation des tenseurs,
- l'analyse en composantes indépendantes.

6.1 PCA robuste

L'analyse en composante principale classique cherche à résoudre

$$\min \|M - L\|_2 \text{ tel que } \operatorname{rank}(L) \leq k.$$

Bien que très employée, sa grande sensibilité aux aberrations souvent présentes dans les données la rend peu fiable. En effet, une seule valeur grossièrement erronée dans M peut rendre l'estimation de L très différente de sa valeur réelle. Dans Candès *et al.* (2011), les auteurs suggèrent de changer la définition de la PCA, et choisit de résoudre un autre problème :

$$\min \|L\|_2 + \lambda \|S\|_1 \text{ tel que } L + S = M.$$

Contrairement à l'hypothèse d'un bruit normal faible N dans la PCA classique, les entrées de S peuvent être aussi grandes que l'on veut, la seule hypothèse que l'on fait sur elles sont que leur support est sparse.

La RPCA peut être calculée par une méthode ADMM, décrite en détail dans Candès *et al.* (2011). Nous avons déjà introduit ce type d'algorithme section 4.4.3. Nous l'avions alors utilisé pour proposer un algorithme de calcul de la NMF, ainsi

que plusieurs méthodes de régularisation par la variation totale. Nous sommes donc désormais en mesure de présenter rapidement cet algorithme.

Le lagrangien augmenté du problème s'écrit comme

$$L_\mu(L, S, \Lambda) = \|L\|_2 + \lambda\|S\|_1 + \langle \Lambda, M - L - S\rangle + \frac{\mu}{2}\|M - L - S\|_F^2.$$

En notant $\mathcal{S}_\tau$ l'opérateur de seuillage (*shrinkage operator*)

$$\text{soft}(x, \tau) = \text{sign}(x)\max(|x| - \tau, 0)\,,$$

et en l'étendant aux matrices en l'appliquant élément par élément, on peut montrer que

$$\arg\min_S L_\mu(L, S, \Lambda) = \text{soft}\left(M - L + \mu^{-1}\Lambda, \frac{\lambda}{\mu}\right).$$

On introduit aussi pour toute matrice X l'opérateur de seuillage des valeurs singulières $\text{soft}_\sigma(X, \tau)$ défini par $\text{soft}_\sigma(X, \tau) = U\,\text{soft}(S, \tau)\,V^T$, où $X = USV^T$ est une SVD. On peut alors montrer que

$$\arg\min_L L_\mu(L, S, \Lambda) = \text{soft}_\sigma\left(M - S + \mu^{-1}\Lambda, \mu^{-1}\right).$$

Si la méthode par ADMM est relativement simple à mettre en œuvre, elle demeure relativement lente, et difficilement applicable à un hypercube de grande taille. Il existe une alternative, plus récente, développée dans Netrapalli *et al.* (2014), basée sur des projections alternées entre l'ensemble des matrices de rang faibles, et celui des matrices sparse.

6.2 Réduction non-linéaire de la dimension

Le dé-mélange hyperspectral fait généralement l'hypothèse d'un mélange linéaire. Toutefois, certains articles expliquent comment des phénomènes peuvent influencer un mélange de spectres de façon non-linéaire. Même en faisant abstraction de ce genre de non-linéarité, il est intéressant de voir si une réduction non-linéaire de la dimension peut produire des projections plus pertinentes que les méthodes classiques, notamment la PCA.

6.2.1 Laplacian Eigenmap

Une des méthodes les plus employées en apprentissage de variété (réduction non-linéaire de la dimension), est la méthode des cartes propres du laplacien. Avant de l'introduire, on discute rapidement de la décomposition en valeurs propres de matrices liées à la matrice d'adjacence d'un graphe.

Théorie spectrale des graphes

On a discuté au chapitre 2 de *clustering*, sans évoquer dans le détail le *clustering* spectral. Ce dernier se propose de classifier les données en ne s'intéressant qu'à la

géométrie locale en chaque point, calculée à partir d'un de ses voisinages. Pour ce, on commence généralement par définir un graphe G spécifiant quels points sont voisins. On calcule ensuite la matrice d'adjacence A de ce graphe, habituellement pondérée par un noyau K_η. En choisissant un noyau gaussien, on obtiendra par exemple, pour deux voisins $x_i \sim x_j$,

$$a_{i,j} = K_\eta(x_i, x_j) = 1_{i\sim j} \exp\left[-\frac{\|x_i - x_j\|_2^2}{2\eta}\right].$$

On forme alors la laplacienne non-symétrique comme $L = D - A$, où

$$d_{i,j} = \begin{cases} \sum_{k=1}^n a_{i,k}, & i = j \\ 0, & i \neq j. \end{cases}$$

On utilise souvent les k plus proches voisins pour définir la relation permettant de construire le graphe. Sommairement, on note $d_{i,j}$ la distance entre le i-ème spectre et le j-ème, et π_i une permutation telle que $\forall j, d_{i,\pi_i(j)} \leq d_{i,\pi_i(j+1)}$. On pose alors $N_k(i)$ la collection des k plus proches voisins de i, autrement dit

$$j \in N_k(i) \Leftrightarrow \pi_i(j) \leq k.$$

Le spectre i est voisin de j, i.e. $i \sim j$, si

$$i \in N_k(j) \text{ ou } j \in N_k(i).$$

Il est important de remarquer que la relation ainsi définie est symétrique. Une fois le graphe construit, on peut étudier sa matrice d'adjacence. Le spectre de A est en effet lié à certaines propriétés du graphe étudié. Par exemple, 0 est toujours valeur propre de L, et sa multiplicité correspond au nombre de composantes connexes de G. Ainsi, si G a deux composantes connexes, l'ordre de 0 comme valeur propre est de deux. Un des deux vecteurs propres associé est 1_n. L'autre est appelé vecteur de Fiedler. Ce vecteur noté v_2 permet de retrouver les deux composantes connexes

$$C_1 = \{i, v_{2,i} \geq 0\} \text{ et } \quad C_2 = \{i, v_{2,i} < 0\}.$$

Calcul du graphe

Avant de pouvoir calculer les plongements, il faut construire le graphe de voisinage. On choisit de le construire comme le graphe des plus proches voisins. Cela permet entre autre de garantir que la matrice d'adjacence correspondante, ainsi que le laplacien sera sparse. En implémentant naïvement cette construction, le calcul de $n(n+1)/2$ distances est nécessaire, distances qu'il faut ensuite trier. Une des astuces classiques en dimension 2 est de partitionner les données en arbre quaternaire – ou en *octree* en dimension 3 – afin de s'éviter beaucoup de calculs inutiles. Cette stratégie est difficile à appliquer à des données hyper-spectrales pour lesquelles la dimension est typiquement très supérieure à 3, car ce genre de stratégie a pour principe de subdiviser l'espace en 2^d sous-régions.

On lui préfère donc la structure de *vantage point tree*, ou arbre VP, qui propose de subdiviser l'espace en choisissant un point de référence définissant deux sous-régions, une regroupant les données dont la distance est inférieure à un certain seuil

et l'autre regroupant l'ensemble des autres points. Avec cette méthode, on arrive à construire le graphe des k plus proches voisins de façon efficace. La méthode est exposée dans Yianilos (1993).

Calcul des cartes propres

La difficulté principale inhérente au *Laplacian eigenmap* est la diagonalisation de la matrice du laplacien. Cette matrice est en effet de taille $n \times n$, ce qui, en fonction de la résolution du cube étudié, peut être très important.

Pour diagonaliser une matrice sparse de grande taille, on peut espérer exploiter le fait que le produit avec un vecteur est relativement rapide à calculer, et utiliser alors l'algorithme de SVD aléatoire décrit au début de ce mémoire. Ce dernier calcule les k plus grandes valeurs propres d'une matrice. Pour trouver les valeurs propres les plus petites, on va donc appliquer la méthode à une matrice modifiée. Le spectre λ d'une matrice définie positive M et le spectre μ de $(M + \alpha I)^{-1}$ pour $\alpha \geq 0$ sont liés par

$$\mu = \frac{1}{\lambda + \alpha},$$

et les deux matrices partagent les mêmes vecteurs propres.

L'étape primordiale de l'algorithme de SVD rapide calcule le produit de la matrice à diagonaliser par une suite de vecteurs aléatoires. Dans le cas exposé dans ce paragraphe, il faut par conséquent calculer le produit de l'inverse d'une matrice et d'un vecteur aléatoire. Pour un vecteur aléatoire noté ω, et en notant L le laplacien du graphe étudié, il faudra donc résoudre

$$Lx = (1 + \alpha)\omega.$$

Pour une matrice sparse, une façon efficace de procéder est d'utiliser une méthode de gradient conjugué, méthode classique décrite par exemple dans Saad (2003). On obtient ainsi une méthode suffisamment rapide pour être appliquée à un hypercube. Les résultats sont présentés figure 6.1. Les valeurs représentées par différents niveaux de gris ne sont pas positives sur chacune de ces 4 projections. Il est par conséquent difficile d'interpréter ces résultats, problème sur lequel nous revenons au paragraphe 6.2.3. Nous présentons donc également une comparaison de la méthode avec celle introduite dans le paragraphe suivant sur un exemple plus visuel, figure 6.2c.

6.2.2 Plongement maximisant la variance

Une autre façon de calculer une réduction non-linéaire de dimension est de maximiser la distance entre chaque point, en essayant de conserver au maximum un graphe de voisinage défini au préalable.

Le SDP correspondant

On cherche plus précisément un plongement $\pi : \mathbb{R}^d \to \mathbb{R}^{d'}$, avec $d > d'$ respectant au mieux

$$\forall\ i \sim j, \quad \mathrm{dist}\left(\pi(x_i), \pi(x_j)\right) = \mathrm{dist}\left(x_i, x_j\right).$$

FIGURE 6.1 – Laplacian Eigenmap appliqué à un hypercube.

Sous ces contraintes, on cherche à maximiser la variance des $\pi(x_i)$ notés y_i par la suite. Ce problème de maximisation de la variance correspond à un programme semi-défini positif. En effet, en posant $z_{i,j} = y_i \cdot y_j$, et $\delta_{i,j} = \operatorname{dist}(x_i, x_j)$, on peut réécrire le problème comme le SDP suivant :

$$\max_{Z \succeq 0} \operatorname{tr}(Z) \text{ tel que } \begin{cases} \forall i \sim j,\ z_{i,i} + 2z_{i,j} + z_{j,j} = \delta_{i,j}, \\ \sum_{1 \leq i \leq j \leq n} z_{i,j} = 0. \end{cases}$$

La seconde contrainte est nécessaire pour que $\operatorname{tr}(Z)$ corresponde à la variance des y_i, car elle permet de centrer ces derniers. Comme

$$\left(\sum_i y_i\right)^2 = \sum_{1 \leq i \leq j \leq n} z_{i,j} = 0,$$

on a bien

$$\operatorname{var}(y) = \sum_i y_i^2 = \operatorname{tr}(Z).$$

On ne fixe pas la dimension de l'espace d'arrivée. Celle-ci correspondra au rang de la matrice Z solution du problème.

L'astuce Burer-Monteiro

Comme précédemment lors du calcul du SDP pour le *clustering* de gaussiennes, il s'agit de calculer un SDP de taille $n \times n$. Bien que ce genre de résolution s'opère en temps polynomial, elle demeure très longue en pratique (de l'ordre de $O(n^3)$). Il est donc intéressant de recourir à nouveau à l'astuce de Burer-Monteiro. On remplace la variable Z dans le SDP original par le produit YY^T, pour $Y \in \mathbb{R}^{n \times k}$.

On doit donc résoudre le problème contraint suivant.

$$\max_Y \operatorname{tr}\left(Y^T Y\right) \text{ tel que } \begin{cases} \operatorname{tr}\left(Y^T 1_n 1_n^T Y\right) = 0 \\ \forall i \sim j,\ \operatorname{tr}\left(Y^T A_{i,j} Y\right) = \delta_{i,j}, \end{cases}$$

où $A_{i,j}$ est la matrice définie comme

$$\begin{pmatrix} a_{i,i} & a_{i,j} \\ a_{j,i} & a_{j,j} \end{pmatrix} = \begin{pmatrix} 1 & -1 \\ -1 & 1 \end{pmatrix}.$$

le reste des coefficients de A étant nul.

On résout ensuite ce problème en utilisant l'algorithme Broyden–Fletcher–Goldfarb–Shanno (BFGS) limité en mémoire sur le lagrangien augmenté du problème, comme suggéré dans le travail de Burer et Monteiro (2003) et celui de Kulis *et al.* (2007).

Points d'ancrage

Quand le programme développé au paragraphe précédent tourne encore trop lentement, on peut essayer de sous-échantillonner les données, en ne calculant que le plongement de certains points d'ancrage, que l'on espère uniformément répartis sur la variété de départ. On calcule alors les coordonnées de chaque point dans l'espace

originel en fonction de ses k plus proches ancres. On calcule ensuite la projection de ces points à partir de la projection des ancres. L'approche est détaillée dans Weinberger *et al.* (2005). Elle permet de calculer une projection non-linéaire pour un grand nombre de points, comme par exemple figure 6.2b où nous calculons la projection de 10000 points par l'intermédiaire de 500 ancres.

6.2.3 Compatibilité avec la NMF

Limitations

Les deux techniques de réduction de la dimension évoquées dans ce chapitre ne sont pas conçues pour préserver la positivité des données. Il semble donc difficile de parler de NMF. Cependant, le point de vue selon lequel les spectres de base sont les sommets du plus grand simplexe inscrit (ou du plus petit simplexe circonscrit) reste valable.

Une fois les spectres de base calculés, on peut déterminer les abondances par moindres carrés contraints au simplexe, ou bien les calculer comme des ratios de volume. Nous ne nous sommes pas attardé sur cette formulation au chapitre 4, on la donne donc ici. Les abondances peuvent être calculées comme

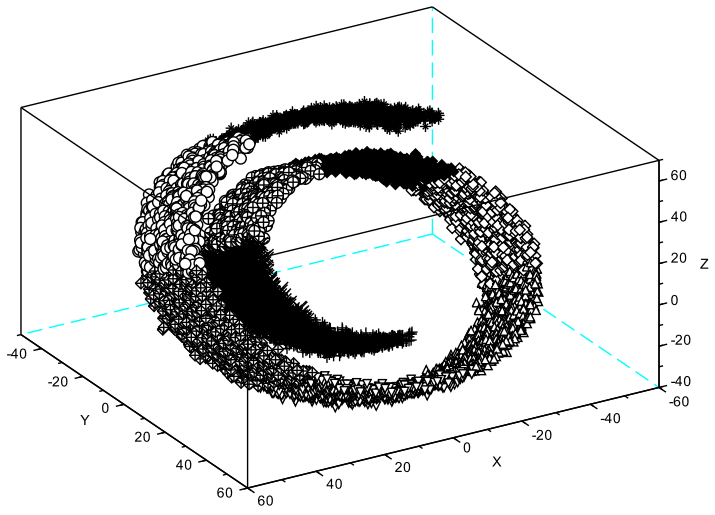
$$h_{i,j} = \frac{\mathrm{vol}(w_1, \cdots, w_{j-1}, m_j, w_{j+1}, \cdots, w_k)}{\mathrm{vol}(w_1, \cdots, w_k)}.$$

Cette formule n'est valable que pour un point m_j à l'intérieur du simplexe défini par W. Le calcul basé sur la formule classique de volume par déterminant (celle basée sur les coordonnées des spectres de base) et le calcul par moindres carrés classiques coïncident. En revanche, en utilisant le déterminant de Cayley-Menger pour calculer le volume, on a alors les abondances en fonction des distances entre les points. En utilisant la distance de la variété calculée précédemment à la place de la distance euclidienne dans l'espace de départ, on calcule ainsi en quelque sorte des abondances non-linéaires. C'est l'idée exploitée par Heylen *et al.* (2011), exposée dans le paragraphe suivant.

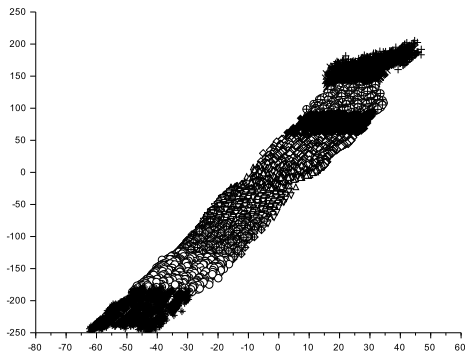
Volume de simplexes à partir de géodésiques

L'idée détaillée dans Heylen *et al.* (2011) est un prolongement naturel de celle de Bauckhage (2014). Dans l'algorithme proposé par Bauckhage, on essaie itérativement de calculer le plus grand simplexe inscrit dans le nuage des spectres étudiés, en exploitant la distance euclidienne entre chaque sommet retenu. Heylen propose simplement d'utiliser une approximation de la distance géodésique (en suivant la variété) entre les sommets.

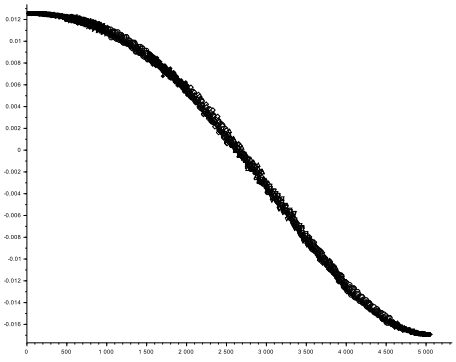
Pour ce, on peut utiliser le graphe des k plus proches voisins. On approxime alors la distance géodésique par la plus courte distance dans le graphe, grâce à un algorithme de parcours de graphe type Djikstra. Cet algorithme permet de calculer la distance d'un sommet du graphe à l'ensemble des autres sommets.



(a) Points répartis uniformément sur une variété.

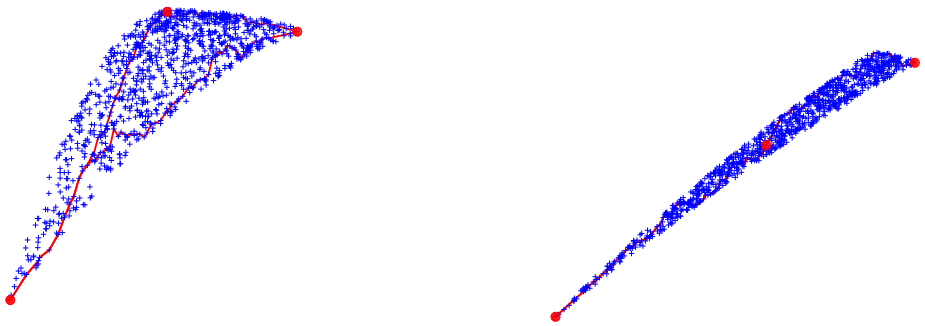


(b) Projection en deux dimensions par MVU.



(c) Projection en deux dimensions par Laplacian Eigenmaps.

FIGURE 6.2 – Comparatif des projections proposées par les Laplacian Eigenmaps et MVU



(a) Un exemple où la technique marche bien. (b) Un exemple où la technique marche mal.

FIGURE 6.3 – Deux exemples de NMF non-linéaires

6.3 Régularisation spatiale

Une fois le cube dé-mélangé, chaque spectre est représenté comme une combinaison convexe de spectres de bases. En voyant ces coefficients comme des probabilités d'appartenance à une classe, on peut produire une partition des spectres. Plutôt que d'assigner chaque spectre à la classe à laquelle il appartient avec la probabilité la plus importante, on peut essayer de faire apparaître une régularité spatiale dans le *clustering*.

6.3.1 Modèle d'Ising-Potts

Ising a développé au début des années 1920 un modèle de mécanique statistique permettant d'étudier les changements de phase dans un réseau cristallin. Développé à l'origine pour le ferro-magnétisme, le modèle consiste en un graphe (souvent un réseau) Λ où chaque sommet possède un moment magnétique $\sigma_i \in \{-1, 1\}$. Chaque sommet interagit avec ses voisins. L'énergie d'une configuration est donnée par

$$H(\sigma) = \sum_i h_i \sigma_i + \sum_{i \sim j} \mu_{i,j} \sigma_i \sigma_j. \tag{6.1}$$

Le modèle de Potts généralise le modèle d'Ising en permettant aux variables σ de prendre k différentes valeurs. L'énergie est alors donnée par

$$H(\sigma) = \sum_i h_i \sigma_i + \sum_{i \sim j} \mu_{i,j} \delta(\sigma_i, \sigma_j), \tag{6.2}$$

où δ est le δ de Kronecker.

On applique ce modèle aux images dé-mélangées $\mathcal{H} \in \Delta_k^{n_1 \times n_2}$, en posant

$$H(\sigma) = \sum_{i \in I} h_{i_1, i_2, \sigma_i} + \mu \sum_{i \sim j} \delta(\sigma_i, \sigma_j),$$

où $I = \{1, \cdots, n_1\} \times \{1, \cdots, n_2\}$ représente les coordonnées de chaque spectre, et où

$$i \sim j \Leftrightarrow |i_1 - j_1| + |i_2 - j_2| \leq 1$$

est la relation définissant les arêtes du graphe de voisinage. Pour régulariser le cube des abondances, il faut alors maximiser la fonction H définie équation 6.2.

L'optimisation des fonctions d'énergie de ces modèles est généralement longue. C'est le cas en utilisant les méthodes type Monte-Carlo par chaîne de Markov. Dans ces méthodes, on procède en effet généralement pixel par pixel. On change la valeur d'un pixel, on observe le changement d'énergie que cela induit, et, en fonction d'un paramètre de température, on accepte ou refuse alors ce changement. Dans ce genre de problématique, il y a souvent beaucoup de minima locaux, et la solution que l'on va produire dépendra fortement de la gestion de la température. Ce type de technique a souvent été utilisé faute de mieux, le problème étant dans certains cas NP-dur, comme expliqué dans Boykov *et al.* (2001). Il existe cependant des techniques d'approximation efficaces.

6.3.2 Optimisation par *min-cut*

Plutôt que de faire un seul changement à chaque itération, Boykov *et al.* (2001) proposent d'en faire plusieurs à la fois, en opérant itérativement à une extension ou à un échange, les deux procédures de base de calcul définies dans l'article. Ces deux ingrédients s'appliquent non à un pixel seul, mais à sous-ensemble de pixels. Ces deux changements peuvent être calculés de façon optimale par *min-cut*, et, en itérant ces changements, les auteurs démontrent que l'on parvient à trouver un minimum proche du minimum global.

L'algorithme décrit dans l'article propose donc une succession de changements élémentaires, basés sur des *min-cut*. L'idée est de ne s'intéresser pour la configuration courante, qu'aux pixels appartenant à deux classes notées α et β. On construit un graphe où chacun de ces pixels est un sommet, et où il existe une arête reliant deux pixels voisins $p \sim q$. A ces sommets et arêtes, on ajoute encore deux sommets, appelés sommets terminaux, notés s_α et t_β. Chaque sommet représentant un pixel p est relié par une arête à s_α et t_β , appelée arête terminale. Ces arêtes sont pondérées de façon à faire correspondre une s_α-t_β coupe minimale à la meilleure configuration possible. En effet, une telle coupe doit inclure exactement une des arêtes terminales. L'arête terminale restante détermine la classe du pixel considéré. Ces considérations sont illustrées figure 6.4.

6.3.3 Autres solutions envisagées

Il existe d'autres façons de régulariser spatialement une partition. On en présente ici deux considérées lors de cette thèse.

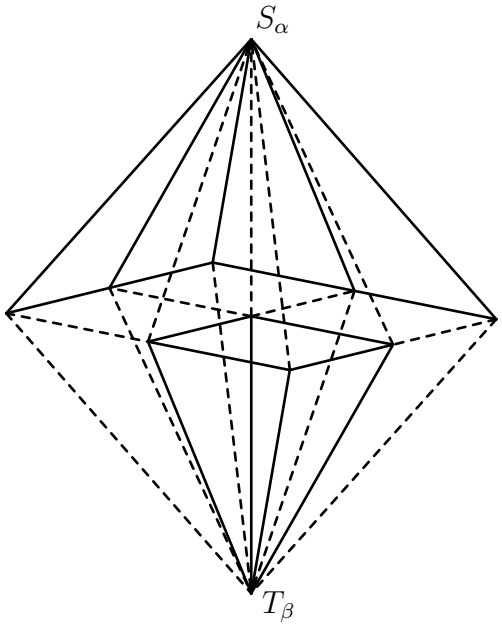


FIGURE 6.4 – Une coupe minimale (en pointillés) définissant une configuration optimale.

Par ligne de partage des eaux

Une des autres solutions envisagées pour la régularisation spatiale des résultats est de commencer par une segmentation du cube. Une fois le cube divisé en régions, on regarde pour chacune d'elles à quelle classe appartient la majorité des pixels. On assigne ensuite l'ensemble des pixels de la région à cette classe. Plus précisément, le i-ème pixel, contenu dans la région R_h, est dans la classe C_k si et seulement si

$$i \in C_k \Leftrightarrow \left[i \in R_h \text{ et } \forall l \sum_{j \in R_h} \delta\left(k, \max m_j\right) \geq \sum_{j \in R_h} \delta\left(l, \max m_j\right) \right].$$

La partie compliquée de cette régularisation spatiale réside donc dans la partie segmentation. Une méthode simple de segmentation est la ligne de partage des eaux. Cette méthode propose de détecter les contours sur une image $\mathbb{R}^m \times \mathbb{R}^n \to \mathbb{R}$ en utilisant une analogie issue de la topographie. On considère la valeur de chaque pixel comme une altitude, et on détecte les minimums locaux d'altitude sur l'image. On place sur chacun de ces derniers une source, qui va petit à petit inonder les pixels qui la bordent, en fonction de leur altitude. Quand deux sources se rejoignent, on a déterminé un pixel appartenant aux contours. Cette idée repose sur le fait que l'on peut comparer les valeurs d'altitude entre deux pixels. Dans le cas des hypercubes, comme un pixel a pour valeur un vecteur, il est difficile de trouver un ordre naturel permettant une telle comparaison. Pour tester rapidement la méthode de régularisation par LPE, nous avons donc travaillé sur la première composante principale du cube. Les résultats ne sont pas convaincants, comme en atteste la figure 6.5. Il est certainement possible d'améliorer quelque peu la méthode, en recourant à des opérations de morphologie mathématique plus sophistiquées, comme par exemple

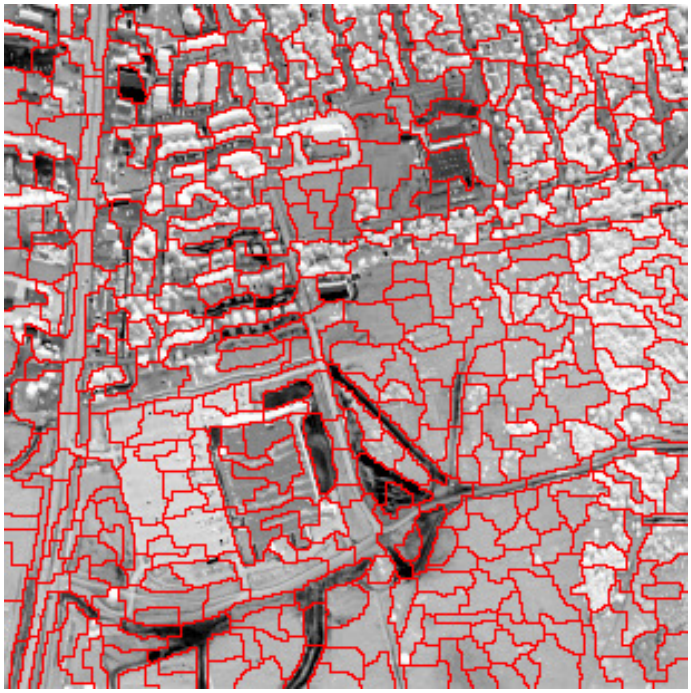


FIGURE 6.5 – Ligne de partage des eaux sur la première composante principale de Urban

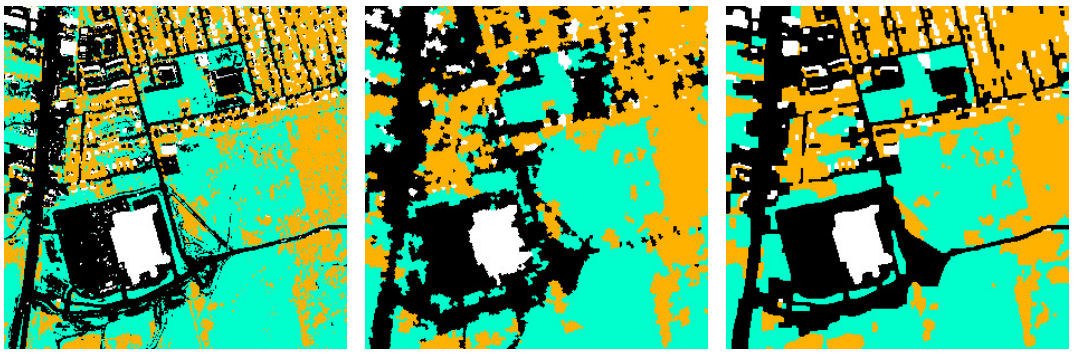


FIGURE 6.6 – Clustering naïf / par LPE / par Potts

au gradient morphologique. C'est le choix fait dans Tarabalka *et al.* (2008). La méthode semble, tout du moins expérimentalement, moins efficace que celle proposée dans le paragraphe précédent, toujours d'après les travaux de Tarabalka *et al.* (2010). On peut comparer les résultats offerts par la méthode décrite dans ce paragraphe à ceux du paragraphe précédent sur la figure 6.6. Le constat est sans appel.

Par opérateurs morphologiques et SVD d'ordre supérieur

D'autres méthodes encore proposent d'intégrer les informations spatiales au *clustering*. L'idée est de sélectionner des *features* qui prendront en compte le côté spatial. En incluant ces *features* spatiaux supplémentaires dans le *clustering*, on espère alors mettre en avant la cohérence spatiale du résultat. On calcule pour cela des profils morphologiques pour chaque tranche du cube. Ces profils sont définis comme

la concaténation des images de chaque tranche obtenue en faisant varier les paramètres d'une certaine opération morphologique (type ouverture ou fermeture). On obtient ainsi un tenseur d'ordre 4. Comme cela représente énormément de données, souvent redondantes, on procède alors ensuite à une réduction de la dimension, par une analyse en composantes indépendantes dans Mura *et al.* (2011), ou par une SVD d'ordre supérieur dans Velasco-Forero et Angulo (2013). Nous présentons ce genre d'outils tensoriels dans la section 6.4. Faute de temps, nous n'avons pas expérimenté ces techniques, mais elles semblent de prime abord très demandeuses en temps de calcul.

6.4 Les tenseurs et leurs valeurs propres

Nous nous sommes déjà rendu compte chapitre 5 qu'il pouvait être intéressant de travailler avec les cubes hyper-spectraux sans les déplier. Cette section présente quelques uns des outils d'algèbre tensorielle utiles pour travailler directement avec des cubes, et définit au passage précisément la notion de dépliage de cube, notion déjà régulièrement utilisée dans les chapitres précédents.

6.4.1 Les tenseurs

Nous considérerons par la suite les tenseurs comme de simples tableaux multi-indices, sur lesquels seront définies plusieurs opérations.

Des tableaux multidimensionnels

Le nombre d'indices nécessaires pour accéder à un élément d'un tenseur est appelé ordre du tenseur. Pour un tenseur $\mathcal{X}$ d'ordre d, le symbole $x_{i_1,\cdots,i_d}$ désigne donc un élément du tenseur. Si cet élément est un réel, et que l'indice i_j varie de 1 à n_j, on notera le tenseur comme

$$\mathcal{X} \in \mathbb{R}^{n_1 \times \cdots \times n_d}.$$

La façon dont on pense à une matrice d'un point de vue informatique, se généralise bien aux tenseurs d'ordre supérieur. On peut voir une matrice de taille $m \times n$ comme un tableau contenant m tableaux de tailles n, ou comme un tableau de taille mn avec une bijection $\{1, \cdots, m\} \times \{1, \cdots, n\} \to \{1, \cdots, mn\}$ associant à deux indices un indice permettant de retrouver l'élément correspondant dans le tableau. Cette deuxième approche pose la question du choix de la bijection parmi les $(mn)!$ possibles. Une solution possible, adoptée dans la bibliothèque de calcul Blas ou dans le quelque peu oublié *slice* du standard C++ par exemple, s'inspire de la numération en base mixte. Par exemple, on peut représenter une matrice A de taille 12×23 comme la donnée d'un tableau t de taille 276 et d'une bijection qui enverrait par exemple $43 = 6_1 3_{12} + 1$ sur $(6+1, 3+1)$. Le 43-ème élément de t correspondrait donc à l'élément $a_{7,4}$. On dira que le couple $(1, 12)$ constitue le *stride* de cette représentation.

Ces considérations se généralisent, et on représente donc un tenseur $\mathcal{T}$ de taille $n_1 \times \cdots \times n_d$ comme la donnée d'un tableau de taille $n_1 \cdots n_d$ et d'une bijection

$$\{1, \cdots, n_1\} \times \cdots \times \{1, \cdots, n_d\} \to \{1, \cdots, n_1 \cdots n_d\}.$$

Cette permutation est caractérisée par un *stride*, permutation d'un certain n-uplet $(1, s_1, \cdots, s_{d-1})$. Pour que la fonction définie par ce n-uplet soit bien une bijection, elle doit vérifier $(n_1, \cdots, n_d) \mapsto n_1 \cdots n_d$. On pourra, comme dans Blas, relâcher cette condition pour pouvoir définir des sous-tenseurs.

Des transpositions matricielles aux transpositions tensorielles

La transposition B de la matrice A est définie comme

$$b_{i_1,i_2} = a_{i_{\sigma(1)},i_{\sigma(2)}} = a_{i_2,i_1},$$

avec $\sigma = (12)$. Si $A \in \mathbb{R}^{m \times n}$ est représenté comme un tableau $c \in \mathbb{R}^{mn}$ avec un *stride* $(1, n)$, on pourra représenter B comme le même tableau c, mais avec un *stride* différent : $(m, 1)$. Si par exemple $c = (0, 1, 2, 3, \cdots, 11)$, les matrices de *stride* $(1, 3)$ et $(4, 1)$ s'écrivent respectivement

$$\begin{pmatrix} 0 & 1 & 2 \\ 3 & 4 & 5 \\ 6 & 7 & 8 \\ 9 & 10 & 11 \end{pmatrix} \text{ et } \begin{pmatrix} 0 & 4 & 8 \\ 1 & 5 & 9 \\ 2 & 6 & 10 \\ 3 & 7 & 11 \end{pmatrix}$$

Dans la nomenclature Blas, quand le 1 du *stride* est en première position, on parle de représentation *row major*, quand il est placé en seconde position, on parle de *column major*. Si les versions c de Blas permettent de travailler avec deux matrices représentées dans des formats différents, ce n'est pas le cas pour les versions Fortran. Cette situation préfigure ce qu'il va se passer avec les tenseurs d'ordre quelconque. On va donc s'attarder un peu sur ce problème dans le paragraphe suivant. Disons pour l'instant seulement que la représentation des matrices que l'on a définie n'est pas unique. Une matrice peut se représenter de deux façons différentes, par exemple

$$\begin{pmatrix} 0 & 1 & 2 \\ 3 & 4 & 5 \\ 6 & 7 & 8 \\ 9 & 10 & 11 \end{pmatrix}$$

peut se représenter comme $(0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11)$ avec *stride* $(3, 1)$, ou comme $(0, 3, 6, 9, 1, 4, 7, 10, 2, 5, 8, 11)$ avec le *stride* $(1, 4)$.

Ce que l'on a défini ci-dessus se généralise aux tenseurs. Si $\mathcal{A} \in \mathbb{R}^{n_1 \times \cdots \times n_d}$ est un tenseur d'ordre d, sa transposition $\mathcal{B}$ par $\sigma \in \mathbb{S}_d$ est définie par

$$\forall (i_1, \cdots, i_d) \in \{1, \cdots, n_1\} \times \cdots \times \{1, \cdots, n_d\}, B_{i_1,\cdots,i_d} = A_{i_{\sigma(1)},\cdots,i_{\sigma(d)}}. \tag{6.3}$$

Comme pour les matrices, un tenseur peut être représenté de plusieurs façons différentes. Par exemple, si $\mathcal{T} \in \mathbb{R}^{2 \times 3 \times 4}$ est défini comme

$$\left(\begin{pmatrix} 1 & 3 & 5 \\ 2 & 4 & 6 \end{pmatrix}, \begin{pmatrix} 7 & 9 & 11 \\ 8 & 10 & 12 \end{pmatrix}, \begin{pmatrix} 13 & 15 & 17 \\ 14 & 16 & 18 \end{pmatrix}, \begin{pmatrix} 19 & 21 & 23 \\ 20 & 22 & 24 \end{pmatrix} \right) \tag{6.4}$$

il existe $6 = 3!$ représentations différentes, compilées dans le tableau 6.1.

stride	tableau unidimensionnel associé
$(1, 2, 6)$	$(1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24)$
$(1, 8, 2)$	$(1, 2, 7, 8, 13, 14, 19, 20, 3, 4, 9, 10, 15, 16, 21, 22, 5, 6, 11, 12, 17, 18, 23, 24)$
$(3, 1, 6)$	$(1, 3, 5, 2, 4, 6, 7, 9, 11, 8, 10, 12, 13, 15, 17, 14, 16, 18, 19, 21, 23, 20, 22, 24)$
$(4, 8, 1)$	$(1, 7, 13, 19, 2, 8, 14, 20, 3, 9, 15, 21, 4, 10, 16, 22, 5, 11, 17, 23, 6, 12, 18, 24)$
$(12, 1, 3)$	$(1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23, 2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24)$
$(12, 4, 1)$	$(1, 7, 13, 19, 3, 9, 15, 21, 5, 11, 17, 23, 2, 8, 14, 20, 4, 10, 16, 22, 6, 12, 18, 24)$

TABLE 6.1 – Les différentes représentations de $\mathcal{A}$ définies par (6.4)

Fibres et tranches de tenseurs

Les fibres sont les analogues d'ordre supérieur des colonnes et des lignes de matrices. Une fibre de mode k est définie en fixant tous les indices d'un tenseur, à l'exception du k-ème. Une ligne de matrice est ainsi une fibre de mode 1, que l'on dénommera aussi 1-fibre, et une colonne une 2-fibre. Une référence très complète introduisant en détail ces notions est Kolda et Bader (2009). Les tranches sont les équivalents des fibres d'ordre 2. Elles sont définies en fixant tout les indices, sauf 2.

Ordre des modes

Il est possible de calculer les *stride*s à partir de la taille d'un tenseur, et de l'ordre dans lequel on veut le stocker en mémoire. Si l'on veut stocker un tenseur de $\mathbb{R}^{n_1 \times n_2, \times n_3}$ en commençant colonne par colonne, face frontale par face frontale, cela va donner un *stride* de $(1, n_1, n_1 n_2)$. Si l'on préfère stocker colonne par colonne, face transversale par face transversale, cela donnera le *stride* $(1, n_1 n_3, n_1)$. On pourrait donc se contenter de spécifier l'ordre des modes pour décrire la représentation d'un tenseur. Si l'on préfère spécifier le *stride*, c'est car cela permet de définir des sous-tenseurs.

Dépliages et contractions

La contraction de deux tenseurs est une opération clé du calcul tensoriel. Cette opération généralise le produit matriciel aux tenseurs d'ordre quelconque. Quand deux tenseurs, pas forcément du même ordre, ont chacun un mode de même taille n, on peut contracter ce mode. La définition générale est relativement pénible à écrire, alors que la notion n'est pas très compliquée. On se contentera donc d'un exemple. Si $\mathcal{A} \in \mathbb{R}^{m \times n \times o}$ et $\mathcal{B} \in \mathbb{R}^{n \times p \times q \times r}$, on peut contracter les deux tenseurs pour obtenir un tenseur $\mathcal{C} \in \mathbb{R}^{m \times o \times p \times q \times r}$, défini comme

$$c_{h,i,j,k,l} = \sum_{g=1}^{n} a_{h,g,i} b_{g,j,k,l}. \tag{6.5}$$

Un cas particulier est la contraction avec une matrice, dont on précise la définition utile pour la suite. Pour un tenseur $\mathcal{A} \in \mathbb{R}^{n_1 \times \cdots \times n_d}$ et une matrice $B \in \mathbb{R}^{n_k \times m}$, on définit la contraction $\mathcal{C} = \mathcal{A} \times_k B \in \mathbb{R}^{n_1 \times n_{k-1} \times m \times n_{k+1} \times \cdots \times n_d}$ comme

$$c_{i_1, \cdots i_n} = \sum_{i=1}^{n_k} a_{i_1, \cdots, i_{k-1}, i, i_{k+1} \cdots i_d} b_{i, i_k}. \tag{6.6}$$

Ces généralisations du produit matriciel peuvent être également introduites via la définition du dépliage d'un tenseur. On verra alors les contractions comme le produit matriciel de deux dépliages. Étant donné un tenseur $\mathcal{A} \in \mathbb{R}^{n_1 \times \cdots \times n_d}$, on définit un dépliage de $\mathcal{A}$ selon le mode k comme une matrice regroupant l'ensemble des k-fibres, dans un certain ordre, parmi les $(d-1)!$ possibles. Grâce à ces dépliages, on peut réécrire la contraction :

$$\mathcal{C} = \mathcal{A} \times_k B \Leftrightarrow C^{(k)} = BA^{(k)}, \tag{6.7}$$

où $C^{(k)}$ est le k-dépliage de $\mathcal{C}$, $A^{(k)}$ est le k-dépliage de $\mathcal{A}$, et où l'on a veillé à déplier les deux tenseurs $\mathcal{C}$ et $\mathcal{A}$ de façon cohérente. Dans le cadre de ce que l'on a choisi pour représenter les tenseurs, on peut préciser les choses. Si un tenseur est représenté par (t, s), avec le *stride* k-major $s = (s_1, \cdots, s_{k-1}, 1, s_{k+1}, \cdots, s_d)$, un de ses k-dépliages admet $((1, \frac{n_1 \cdots n_d}{n_k}), t)$ comme représentation. Par exemple, pour calculer $\mathcal{T} \times_3 U$, avec

$$U = \begin{pmatrix} 25 & 26 & 27 & 28 \\ 29 & 30 & 31 & 32 \end{pmatrix},$$

on commence par trouver une représentation 3-majeur de $\mathcal{T}$ comme par exemple celle de *stride* $(12, 4, 1)$,

$$(1, 7, 13, 19, 3, 9, 15, 21, 5, 11, 17, 23, 2, 8, 14, 20, 4, 10, 16, 22, 6, 12, 18, 24)\,,$$

que l'on choisit de voir comme une matrice de taille 4×6, de *stride* $(1, 4)$

$$\begin{pmatrix} 1 & 3 & 5 & 2 & 4 & 6 \\ 7 & 9 & 11 & 8 & 10 & 12 \\ 13 & 15 & 17 & 14 & 16 & 18 \\ 19 & 21 & 23 & 20 & 22 & 24 \end{pmatrix}.$$

On multiplie alors ce dépliage par U, pour obtenir

$$\begin{pmatrix} 1180 & 1404 & 1628 & 1292 & 1516 & 1740 \\ 1220 & 1452 & 1684 & 1336 & 1568 & 1800 \end{pmatrix}.$$

Reste alors à replier cette matrice en tenseur d'ordre 3. On commence par en construire une représentation de *stride* $(1, 2)$, ce qui donne

$$(1180, 1220, 1404, 1452, 1628, 1687, 1292, 1336, 1516, 1568, 1740, 1800)\,,$$

ce qui, avec le *stride* $(6, 2, 1)$ représente le tenseur

$$\left(\begin{pmatrix} 1180 & 1404 & 1628 \\ 1292 & 1516 & 1740 \end{pmatrix}, \begin{pmatrix} 1220 & 1452 & 1687 \\ 1336 & 1568 & 1800 \end{pmatrix} \right).$$

Il faut remarquer que l'on a fait attention à rester cohérent dans le choix de l'ordre des fibres à chaque dépliage/repliage. Le premier dépliage est caractérisé par le *stride* $(12, 4, 1)$, le dernier repliage est donc caractérisé par le *stride* $(6, 2, 1)$, les deux *stride*s sont dans le même ordre.

6.4.2 Diagonalisation des tenseurs

Les diverses opérations tensorielles définies dans les paragraphes précédents permettent d'envisager différents types de diagonalisation pour les tenseurs. On en présente maintenant deux versions, une relativement simple mais peu utile en pratique, puis une autre, aux applications plus nombreuses, mais nettement plus compliquée.

SVD d'ordre supérieur

La décomposition de Tucker, est une forme de décomposition en valeurs singulières d'ordre supérieur (HOSVD). Elle factorise un tenseur en un noyau (du même ordre) contracté sur chaque mode par une matrice, comme défini dans Kolda et Bader (2009). Pour un tenseur $\mathcal{X} \in \mathbb{R}^{I \times J \times K}$ d'ordre 3, cela donne

$$\mathcal{X} = \sum_{p=1}^{P} \sum_{Q=1}^{Q} \sum_{r=1}^{R} g_{pqr} a_p \otimes b_q \otimes c_r,$$

ce qu'on écrit aussi

$$\mathcal{X} = \mathcal{G} \times_1 A \times_2 B \times_3 C.$$

Dans l'égalité ci-dessus, $A \in \mathbb{R}^{I \times P}, B \in \mathbb{R}^{J \times Q}$, et $C \in \mathbb{R}^{K \times R}$ sont les facteurs matriciels (que l'on considère habituellement comme orthogonaux), et peuvent être vus comme les composantes principales de chaque mode. Le produit dénoté $\otimes$ opéré sur les colonnes des facteurs matriciels est le produit externe (la même notation est utilisée pour le produit de Kronecker, il faudra donc les différencier en fonction du contexte). Le tenseur $\mathcal{G} \in \mathbb{R}^{P \times Q \times R}$ est appelé noyau, et ses coefficients montrent le niveau d'interaction entre chaque composante.

On peut définir une notion de rang compatible avec cette généralisation de la SVD. Si $\mathcal{X}$ est un tenseur d'ordre N, de taille $I_1 \times I_2 \times \cdots \times I_N$, alors son n-rang est défini comme le rang de $X_{(n)}$.

Élément par élément, la décomposition de Tucker (ou HOSVD) s'écrit

$$x_{ijk} = \sum_{p=1}^{P} \sum_{q=1}^{Q} \sum_{r=1}^{R} g_{pqr} a_{ip} b_{jq} c_{kr},$$

pour $i = 1, \cdots I, j = 1, \cdots J, k = 1, \cdots K$. Les entiers P, Q, et R représentent le nombre de composantes des facteurs matriciels A, B, et C respectivement. Quand P, Q, R sont plus petits que I, J, K, on peut voir $\mathcal{G}$ comme une version compressée de $\mathcal{X}$.

Les formes dépliées sont données par

$$\begin{aligned} X_{(1)} &\approx A G_{(1)} (C \otimes B)^T, \\ X_{(2)} &\approx B G_{(2)} (C \otimes A)^T, \\ X_{(3)} &\approx C G_{(3)} (B \otimes A)^T. \end{aligned}$$

Algorithme basique de HOSVD

La décomposition de Tucker se généralise aux tenseurs d'ordre N, comme

$$\mathcal{X} \approx \mathcal{G} \times_1 A^{(1)} \times_2 A^{(2)} \times_3 \cdots \times_N A^{(N)}.$$

On en donne les formes matricées ci-dessous :

$$X_{(n)} = A^{(n)} G_{(n)} \left(A^{(N)} \otimes \cdots \otimes A^{(n+1)} \otimes A^{(n-1)} \otimes \cdots \otimes A^{(1)} \right)^T.$$

On en déduit l'algorithme 6 permettant le calcul de HOSVD.

Algorithme 6 HOSVD permettant de calculer une décomposition de Tucker de rang $(R_1, R_2, \cdots, R_N)$.

Entrée : $\mathcal{X} \in \mathbb{R}^{I_1 \times \cdots \times I_N}$ de rang R
Sortie : $\mathcal{G}, A^{(1)}, \cdots, A^{(N)}$

1: Pour $n = 1, \cdots, N$, on assigne à $A^{(n)}$ les R_n premiers vecteurs singuliers à gauche de $X_{(n)}$.
2: $\mathcal{G} = \mathcal{X} \times_1 {A^{(1)}}^T \times_2 {A^{(2)}}^T \times_3 \cdots {A^{(N)}}^T$

Pour un tenseur $\mathcal{X}$ quelconque, on peut facilement donner une décomposition de Tucker de rang $(R_1, R_2, \cdots, R_N)$, avec $R_n = \text{rank}_n(\mathcal{X})$. Quand par contre, on cherche à calculer une décomposition avec $R_n < \text{rank}_n(\mathcal{X})$ pour un ou plusieurs modes n, les choses deviennent plus délicates. Le théorème d'Eckart-Young, qui garantit que la SVD tronquée au rang k est la meilleure approximation de rang k de la matrice de départ au sens de Frobenius, n'est en effet pas vérifié systématiquement par HOSVD. Pour ce, il faut utiliser une procédure nommée HOOI améliorant itérativement le résultat.

Higher Order Orthogonal Iteration

C'est ce que l'on fait, si l'on veut se rapprocher d'une factorisation respectant une sorte de théorème d'Eckart-Young. Contrairement à la SVD tronquée matricielle, il n'existe pas de théorème d'Eckart–Young garantissant que le produit de cette factorisation soit la meilleure approximation de rang $(R_1, R_2, \cdots, R_N)$. Mais même si cette approximation n'est pas la meilleure possible, elle constitue un bon point de départ pour une procédure de moindres carrés alternés. Si $\mathcal{X}$ est un tenseur de taille $I_1 \times I_2 \times \cdots \times I_N$, le but est de résoudre le problème d'optimisation suivant :

$$\min_{\mathcal{G}, A^{(1)}, \cdots, A^{(N)}} \left\| \mathcal{X} - \mathcal{G} \times_1 {A^{(1)}}^T \times_2 {A^{(2)}}^T \times_3 \cdots {A^{(N)}}^T \right\|,$$

avec

$$\begin{cases} \mathcal{G} \in \mathbb{R}^{R_1 \times \cdots \times R_N}, \\ A^{(n)} \in \mathbb{R}^{I_n \times R_n} \text{ orthogonales, pour tout } n = 1, \cdots, N. \end{cases}$$

Ce problème peut être résolu avec l'algorithme 7.

Algorithme 7 Algorithme HOOI calculant une décomposition de Tucker de rang $(R_1, R_2, \cdots, R_N)$ pour un tenseur $\mathcal{X}$ d'ordre N.

Entrée : $\mathcal{G}, A^{(1)}, \cdots, A^{(N)}$
Sortie : $\lambda, A^{(1)}, \cdots, A^{(N)}$

On initialise $A^{(n)} \in \mathbb{R}^{I_n \times R}$ pour $n = 1, \cdots, N$ avec HOSVD
repeat
  for $n \in \{1, \cdots, N\}$ **do**
    $\mathcal{Y} = \mathcal{X} \times_1 {A^{(1)}}^T \times_2 \cdots \times_{n-1} {A^{(n-1)}}^T \times_{n+1} {A^{(n+1)}}^T \times_{n+2} \cdots \times_N {A^{(N)}}^T$
    $A^{(n)} = R_n$ principaux vecteurs singuliers de $Y_{(n)}$
  end for
until Nombre maximum d'itération atteint, ou convergence.
$\mathcal{G} = \mathcal{X} \times_1 {A^{(1)}}^T \times_2 {A^{(2)}}^T \times_3 \cdots \times_N {A^{(N)}}^T$

Application : filtrage de Wiener pour les hypercubes

On montre dans ce paragraphe l'une des utilités envisageables de la SVD d'ordre supérieur, utilisée comme filtre dé-bruiteur pour les cubes. Pour filtrer un tenseur d'ordre supérieur, on fait l'hypothèse du modèle suivant :

$$\mathcal{R} = \mathcal{X} + \mathcal{N},$$

où $\mathcal{R}$ est le tenseur des observations, $\mathcal{X}$ celui du signal pur sans bruit, et $\mathcal{N}$ celui du bruit. Le filtre de Wiener s'écrit alors

$$\hat{\mathcal{X}} = \mathcal{R} \times_1 H^{(1)} \times_2 H^{(2)} \times_3 \cdots \times_N H^{(N)},$$

où $\hat{\mathcal{X}}$ est l'estimateur de $\mathcal{X}$, et $H^{(n)}$ sont les filtres de Wiener d'ordre n.

Comme pour les filtres de Wiener d'ordre n, le problème principal consiste à estimer le signal pur et le bruit. On expose dans la suite de ce paragraphe la méthode utilisée dans Marot et Bourennane (2008) pour obtenir ces estimations. On utilise ici une PCA dans le but d'estimer les deux sous espaces suivants.

- $E_1^{(n)}$ est le sous-espace vectoriel engendré par les K_n vecteurs propres dominants de $X_{(n)}$, assimilé à l'espace du signal pur,
- $E_2^{(n)}$ est le sous-espace vectoriel engendré par les $I_n - K_n$ vecteurs propres dominés de $X_{(n)}$, assimilé à l'espace du bruit.

De façon similaire à ce que l'on ferait pour un signal d'ordre 1, on définit chaque filtre comme :

$$H^{(n)} = \gamma_{XR}^{(n)} {\Gamma_{RR}^{(n)}}^{-1},$$

avec

$$\begin{cases} \gamma_{XR}^{(n)} = \mathbb{E} X_{(n)} T^{(n)} {R_{(n)}}^T \\ \Gamma_{RR}^{(n)} = \mathbb{E} R_{(n)} Q^{(n)} {R_{(n)}}^T \end{cases}$$

où

$$\begin{cases} T^{(n)} = H^{(1)} \otimes \cdots \otimes H^{(n-1)} \otimes H^{(n+1)} \otimes H^{(N)} \\ Q^{(n)} = {T^{(n)}}^T T^{(n)} \end{cases}$$

Le problème est qu'on ne connaît pas $X_{(n)}$. Pour l'estimer, on utilise $E_1^{(n)}$ et $E_2^{(n)}$ définis plus haut. On écrit

$$X_{(n)} = V_S^{(n)} O^{(n)},$$

où les lignes de $V_S^{(n)}$ sont une base de $E_1^{(n)}$, et les colonnes de $O^{(n)}$ sont ses coordonnées.

Le filtre de Wiener prend alors la forme

$$H^{(n)} = V_S^{(n)} \gamma_{OO}^n {\Lambda_{\Gamma_S}^{(n)}}^{-1},$$

avec

$$\gamma_{OO}^n {\Lambda_{\Gamma_S}^{(n)}}^{-1} = \operatorname{diag}\begin{pmatrix} \frac{\lambda_1^\gamma - \sigma_\Gamma^{(n)}}{\lambda_1^\Gamma} & \cdots & \frac{\lambda_{K_N}^\gamma - \sigma_\Gamma^{(n)}}{\lambda_{K_N}^\Gamma} \end{pmatrix},$$

où

- les réels $\begin{pmatrix} \lambda_1^\gamma & \cdots & \lambda_{K_N}^\gamma \end{pmatrix}$ sont les valeurs propres dominantes de $\gamma_{RR}^{(n)}$,
- les réels $\begin{pmatrix} \lambda_1^\Gamma & \cdots & \lambda_{K_N}^\Gamma \end{pmatrix}$ sont les valeurs propres dominantes de $\Gamma_{RR}^{(n)}$,
- et le réel ${\sigma_\Gamma^{(n)}}^2 = \frac{1}{I_n - K_n} \sum_{k_n = K_{n+1}}^{I_n} \lambda_{k_n}^\gamma$ est une estimation de la variance du bruit.

Afin d'effectuer toutes ces estimations, on fixe tous les $H^{(k)}$ sauf un, on le calcule avec les équations décrites ci-dessus, et on passe au suivant, décrivant ainsi un algorithme de moindres carrés alternés. On notera en plus qu'il faut à chaque itération soit connaître K_n, soit l'estimer. On obtient finalement un filtre lourd en temps de calcul, et difficile à paramétrer.

Un autre type de diagonalisation : la décomposition polyadique canonique

La décomposition canonique polyadique Kolda et Bader (2009) factorise un tenseur comme une somme de tenseurs de rang 1. Par exemple, on factorise un tenseur $\mathcal{X} \in \mathbb{R}^{I \times J \times K}$ d'ordre 3 comme

$$\mathcal{X} = \sum_{r=1}^{R} a_r \otimes b_r \otimes c_r,$$

où R est un entier positif, et $a_r \in \mathbb{R}^I$, $b_r \in \mathbb{R}^J$ et $c_r \in \mathbb{R}^K$ pour $r = 1, \cdots R$.

Les facteurs matriciels de cette décomposition sont obtenus par concaténation des vecteurs formant les composantes de rang 1, i.e. , $A = \begin{pmatrix} a_1 & a_2 & \cdots & a_R \end{pmatrix}$, même chose pour B et C.

On peut alors écrire CP dans une forme matricielle, avec l'une des équations suivantes :

$$\begin{aligned} A_{(1)} &\approx A\,(C \odot B)^T, \\ A_{(2)} &\approx B\,(C \odot A)^T, \\ A_{(3)} &\approx C\,(B \odot A)^T. \end{aligned}$$

Dans les trois équations précédentes, $\cdot \odot \cdot$ désigne le produit de Khatri-Rao, sorte de produit de Kronecker colonne par colonne. Étant données les matrices $A \in \mathbb{R}^{I \times K}$ et $B \in \mathbb{R}^{J \times K}$, leur produit de Khatri-Rao, noté $A \odot B$, est une matrice de taille $(IJ) \times K$ définie par

$$A \odot B = \begin{pmatrix} a_1 \otimes b_1 & a_2 \otimes b_2 & \cdots & a_K \otimes b_K \end{pmatrix}.$$

Cela se généralise aux tenseurs $\mathcal{X} \in \mathbb{R}^{I_1 \times \cdots \times I_N}$ d'ordre d, leur décomposition CP s'écrit

$$\mathcal{X} = \sum_{r=1}^{R} \lambda_r \; a_r^{(1)} \otimes a_r^{(2)} \otimes \cdots \otimes a_r^{(N)}.$$

Le rang d'un tenseur $\mathcal{X}$, noté $\operatorname{rank}(\mathcal{X})$, est défini comme le plus petit nombre de tenseur de rang 1 admettant $\mathcal{X}$ comme somme. Cette définition est analogue à celle du rang matriciel, même si leurs propriétés sont très différentes. Une de ces différences est que le rang d'un tenseur peut être différent sur $\mathbb{R}$ ou sur $\mathbb{C}$. Une autre différence importante est qu'on ne connaît pas d'algorithme permettant de déterminer le rang d'un tenseur. Ce problème est NP-dur. En pratique, on détermine le rang d'un tenseur en calculant plusieurs factorisations CP de rangs différents, puis en choisissant la meilleure.

La décomposition CP commence à trouver de nombreux usages, liés notamment à l'analyse en composantes indépendantes, à la Latent Dirichlet Association, et même aux *Stochastic Bloc Models*, cf. Anandkumar *et al.* (2013, 2014). Une application plus proche du sujet de la thèse est l'analyse de cubes issus de cathodo-luminescence par CP. L'approche est détaillée dans Bro (1997)

Une fois le rang fixé, il existe plusieurs façons de calculer une décomposition CP. Dans la paragraphe suivant, on s'intéresse à l'algorithme le plus classique, un algorithme de moindres carrés alternés.

Algorithme basique pour la décomposition CP

Soit $\mathcal{X} \in \mathbb{R}^{I \times J \times K}$ un tenseur d'ordre 3. Le but de l'algorithme ALS est de trouver une décomposition CP avec R composantes de rang 1 approximant au mieux $\mathcal{X}$, i.e. de résoudre

$$\min_{\hat{\mathcal{X}}} \left\| \mathcal{X} - \hat{\mathcal{X}} \right\|, \quad \text{with} \quad \hat{\mathcal{X}} = \sum_{r=1}^{R} \lambda_r \; a_r \otimes b_r \otimes c_r. \tag{6.8}$$

L'approche par moindres carrés alternés fixe A et B, et résout le problème (6.8) en C. On fixe ensuite B et C pour résoudre en A, et ainsi de suite. Cela revient à résoudre

$$\min_{\hat{A}} \left\| X_{(1)} - \hat{A} \left(C \odot B \right)^T \right\|_F,$$

avec $\hat{A} = A \cdot \operatorname{diag}(\lambda)$. La solution à ce dernier problème est donnée par

$$\hat{A} = X_{(1)} \left[\left(C \odot B \right)^T \right]^{\dagger},$$

ce que l'on peut aussi écrire

$$\hat{A} = X_{(1)} \left(C \odot B \right)^T \left(C^T C \circ B^T B \right)^{\dagger},$$

où $\circ$ désigne le produit de Hadamard, et $\cdot^\dagger$ la pseudo-inverse.

La procédure complète est décrite dans l'algorithme 8. Les facteurs matriciels peuvent être initialisés soit de façon aléatoire, soit avec

$$A^{(n)} = R \text{ premiers vecteurs singuliers à gauche de } X_{(n)} \text{ pour } n = 1, \cdots N.$$

On remarquera que l'algorithme 8 déplie le tenseur sur chacun des modes itérativement.

Algorithme 8 Algorithme ALS calculant une décomposition Canonique Polyadique à R composantes pour un tenseur d'ordre N.

Entrée : $\mathcal{X} \in \mathbb{R}^{I_1 \times \cdots \times I_N}$ de rang R
Sortie : $\lambda, A^{(1)}, \cdots, A^{(N)}$

initialize $A^{(n)} \in \mathbb{R}^{I_n \times R}$ for $n = 1, \cdots, N$
repeat
  for $n \in \{1, \cdots, N\}$ **do**
    $V = {A^{(1)}}^T A^{(1)} \circ \cdots \circ {A^{(n-1)}}^T A^{(n-1)} {A^{(n+1)}}^T A^{(n+1)} \circ \cdots \circ {A^{(N)}}^T A^{(N)}$
    $A^{(n)} = X_{(n)} \left(A^{(N)} \odot \cdots \odot A^{(n+1)} \odot A^{(n-1)} \odot \cdots \odot A^{(1)} \right) V^\dagger$
  end for
until Nombre maximum d'itération atteint, ou convergence.

Diagonalisation alternée des dépliages, intérêt des *shuffles*

Les algorithmes de décomposition 6 et 8 utilisent beaucoup de transpositions tensorielles. Naïvement implémentées, ces dernières nécessitent des copies intermédiaires.On s'intéresse donc dans ce paragraphe aux transpositions tensorielles sans copies intermédiaires, ou *in place* en anglais.

Pour appliquer une transposition σ à un tenseur $\mathcal{A} \in \mathbb{R}^{n_1 \times \cdots \times n_d}$, l'algorithme naïf consiste à créer un tenseur $\mathcal{B} \in \mathbb{R}^{n_{\sigma(1)} \times \cdots \times n_{\sigma(d)}}$, puis en bouclant sur

$$(i_1, \cdots, i_d) \in \{1, \cdots, n_1\} \times \cdots \times \{1, \cdots, n_d\},$$

à assigner à $B_{i_1, \cdots, i_d}$ la valeur de $A_{i_{\sigma(1)}, \cdots, i_{\sigma(d)}}$. Cet algorithme naïf demande l'introduction d'un tenseur temporaire, ce qui pourrait être évité grâce au *perfect shuffle*.

On commence par s'intéresser au problème pour les matrices, on reviendra plus tard aux tenseurs. Pour les matrices, le problème est assimilable à une technique de mélange de cartes, nommée *perfect shuffle*. On coupe un paquet de cartes en deux paquets de même taille, que l'on fusionne en imbriquant parfaitement les cartes d'un paquet dans l'autre. Ce 2-*perfect shuffle* se généralise en *k*-*perfect shuffle*, en commençant par couper le paquet en k paquets de cartes. Coupons par exemple un

paquet de 52 cartes, noté $p = \{1, \cdots, 52\}$, en 4 paquets de 13 cartes.

$$p_1 = \begin{pmatrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \\ 9 \\ 10 \\ 11 \\ 12 \\ 13 \end{pmatrix} \quad p_2 = \begin{pmatrix} 14 \\ 15 \\ 16 \\ 17 \\ 18 \\ 19 \\ 20 \\ 21 \\ 22 \\ 23 \\ 24 \\ 25 \\ 26 \end{pmatrix} \quad p_3 = \begin{pmatrix} 27 \\ 28 \\ 29 \\ 30 \\ 31 \\ 32 \\ 33 \\ 34 \\ 35 \\ 36 \\ 37 \\ 38 \\ 39 \end{pmatrix} \quad p_4 = \begin{pmatrix} 40 \\ 41 \\ 42 \\ 43 \\ 44 \\ 45 \\ 46 \\ 47 \\ 48 \\ 49 \\ 50 \\ 51 \\ 52 \end{pmatrix}$$

On fusionne ensuite les 4 paquets en les intercalant parfaitement. On obtient alors le paquet $s = \{1, 14, 27, 40, 2, 15, 28, 41, 3, 16, 29, 42, \cdots, 52\}$. Il faut remarquer que p et s sont deux représentations de la matrice (p_1, p_2, p_3, p_4), l'une avec le *stride* $(1, 13)$, l'autre avec le *stride* $(4, 1)$. La question que l'on peut alors se poser est comment passer efficacement de p à s. Une solution est de factoriser la permutation qui envoie p sur s. Cette permutation envoie toujours 1 sur 1, et 52 sur 52, on peut donc ne s'intéresser qu'à la permutation des 50 cartes situées entre la première et la dernière carte. On peut vérifier que cette permutation de $\mathbb{S}_{50}$ s'écrit

$$\sigma_{13,4} : i \to 4i \pmod{51},$$

ce qui se factorise en 12 cycles disjoints

$$\begin{aligned} &\big(3 \quad 39 \quad 48 \quad 12\big) \cdot \big(9 \quad 15 \quad 42 \quad 36\big) \cdot \big(27 \quad 45 \quad 24 \quad 6\big) \cdot \big(30 \quad 33 \quad 21 \quad 18\big) \cdot \\ &\big(1 \quad 13 \quad 16 \quad 4\big) \cdot \big(37 \quad 22 \quad 31 \quad 46\big) \cdot \big(43 \quad 49 \quad 25 \quad 19\big) \cdot \big(10 \quad 28 \quad 7 \quad 40\big) \cdot \\ &\big(35 \quad 47 \quad 50 \quad 38\big) \cdot \big(20 \quad 5 \quad 14 \quad 29\big) \cdot \big(26 \quad 32 \quad 8 \quad 2\big) \cdot \big(44 \quad 11 \quad 41 \quad 23\big). \end{aligned}$$

Comme ces cycles sont disjoints, on peut les appliquer au tas de carte initial de façon indépendante, en commençant par n'importe lequel des éléments de ces cycles, appelé graine. Le problème devient alors de calculer pour chaque permutation

$$\sigma_{k,n} : i \to ki \pmod{kn - 1}$$

un ensemble de graines. Ce genre de problème a été étudié en mathématiques, entre autre par Diaconis *et al.* (1983).

La factorisation du *k-perfect shuffle* est un problème résolu dans Ellis *et al.* (2002). La méthode est basée sur le constat suivant.

Soit C un cycle de $\sigma_{k,n}$, et $a \in C$. On peut alors écrire $C = \{a, ka, \cdots, k^{r-1}a\}$, où r est le plus petit entier vérifiant $k^r a = a$. On calcule alors g le plus grand diviseur commun à a et $m = kn - 1$. En posant $d = m/g$, on a alors $\gcd(a/g, d) = 1$, dont on déduit $a/g \in (\mathbb{Z}/d\mathbb{Z})^\times$. Comme $k^r a = a \pmod{m}$, on a

$$k^r \frac{a}{g} = \frac{a}{g} \pmod{d},$$

et donc $k^r = 1 \pmod{d}$. Comme, de plus, r est le plus petit entier vérifiant cette propriété, k est d'ordre r dans $(\mathbb{Z}/d\mathbb{Z})^\times$. On en déduit que

$$\frac{a}{g}\{1, k, \cdots k^{r-1}\}$$

est une classe de $\frac{a}{g} \in (\mathbb{Z}/d\mathbb{Z})^\times$ selon $\langle k\rangle_d = \{1, k, \cdots k^{r-1}\}$.

On peut montrer que réciproquement, pour tout $d|m$, toute classe d'un élément de $(\mathbb{Z}/d\mathbb{Z})^\times$ selon $\langle k\rangle_d$ multipliée par m/d est un cycle de $\sigma_{k,n}$. Il reste alors à calculer ces classes, un algorithme pour ce faire est décrit dans Ellis *et al.* (2002).

Par exemple, pour $m = 51$, il existe 3 diviseurs non-triviaux : $\{3, 17, 51\}$. Commençons avec $d = 3$. On a $\langle 13\rangle_3 = \{1\}$. De plus $(\mathbb{Z}/3\mathbb{Z})^\times = \{1, 2\}$, ce qui, comme $51/3 = 17$, nous donne les cycles de longueur 1 (17) et (34). Pour $d = 17$, on a $\langle 13\rangle_{17} = \{1, 13, 16, 4\}$, et $(\mathbb{Z}/17\mathbb{Z})^\times = \{1, \cdots, 16\}$, ce qui donne les cycles

$$\begin{array}{l}
\Big(3 \quad 39 \quad 48 \quad 12\Big), \Big(6 \quad 27 \quad 45 \quad 24\Big), \Big(9 \quad 15 \quad 42 \quad 36\Big), \Big(12 \quad 3 \quad 39 \quad 48\Big), \\
\Big(15 \quad 42 \quad 36 \quad 9\Big), \Big(18 \quad 30 \quad 33 \quad 21\Big), \Big(21 \quad 18 \quad 30 \quad 33\Big), \Big(24 \quad 6 \quad 27 \quad 45\Big), \\
\Big(27 \quad 45 \quad 24 \quad 6\Big), \Big(30 \quad 33 \quad 21 \quad 18\Big), \Big(33 \quad 21 \quad 18 \quad 30\Big), \Big(36 \quad 9 \quad 15 \quad 42\Big), \\
\Big(39 \quad 48 \quad 12 \quad 3\Big), \Big(42 \quad 36 \quad 9 \quad 15\Big), \Big(45 \quad 24 \quad 6 \quad 27\Big), \Big(48 \quad 12 \quad 3 \quad 39\Big).
\end{array}$$

Il faudra alors retirer les redondances, comme $\Big(3 \quad 39 \quad 48 \quad 12\Big) = \Big(39 \quad 48 \quad 12 \quad 3\Big)$. On obtient alors les 4 cycles

$$\Big(3 \quad 39 \quad 48 \quad 12\Big), \Big(6 \quad 27 \quad 45 \quad 24\Big), \Big(9 \quad 15 \quad 42 \quad 36\Big), \Big(18 \quad 30 \quad 33 \quad 21\Big).$$

Pour $d = 51$, on a $\langle 13\rangle_{51} = \{1, 13, 16, 4\}$. De plus

$$\begin{aligned}
(\mathbb{Z}/51\mathbb{Z})^\times = \{&1, 2, 4, 5, 7, 8, 10, 11, 13, 14, 16, 19, 20, 22, 23, 25, 26, \\
&28, 29, 31, 32, 35, 37, 38, 40, 41, 43, 44, 46, 47, 49, 50\}.
\end{aligned}$$

On a donc les 8 cycles suivants, sans écrire les doublons

$$\begin{array}{l}
\Big(1, 13, 16, 4\Big), \Big(2, 26, 32, 8\Big), \Big(5, 14, 29, 20\Big), \Big(7, 40, 10, 28\Big), \\
\Big(11, 41, 23, 44\Big), \Big(19, 43, 49, 25\Big), \Big(22, 31, 46, 37\Big), \Big(35, 47, 50, 38\Big).
\end{array}$$

On a ainsi retrouvé les 12 cycles facteurs de la transposition $\sigma_{13,4}$.

Pour transposer un tenseur en s'inspirant de ce que l'on vient d'exposer pour les matrices, il faut modifier quelque peu l'analogie du mélange de carte. On peut s'imaginer mélanger des paquets de paquets de cartes, pour un tenseur d'ordre 3, ou, pour un tenseur d'ordre 4, des paquets de paquets de paquets de cartes, et ainsi de suite. Imaginons que l'on veuille inverser l'ordre de deux modes voisins. On peut alors aisément se ramener au cas de la transposition matricielle. Pour un tenseur d'ordre 3 par exemple, vu comme un paquet de paquets de cartes. Échanger les modes 1 et 2, revient à mélanger chacun des paquets du paquet de paquets. Échanger les modes 2 et 3 revient à faire du *perfect shuffle* avec les paquets, sans toucher aux cartes. Ce genre de considération se généralise bien. Si l'on dispose d'un algorithme

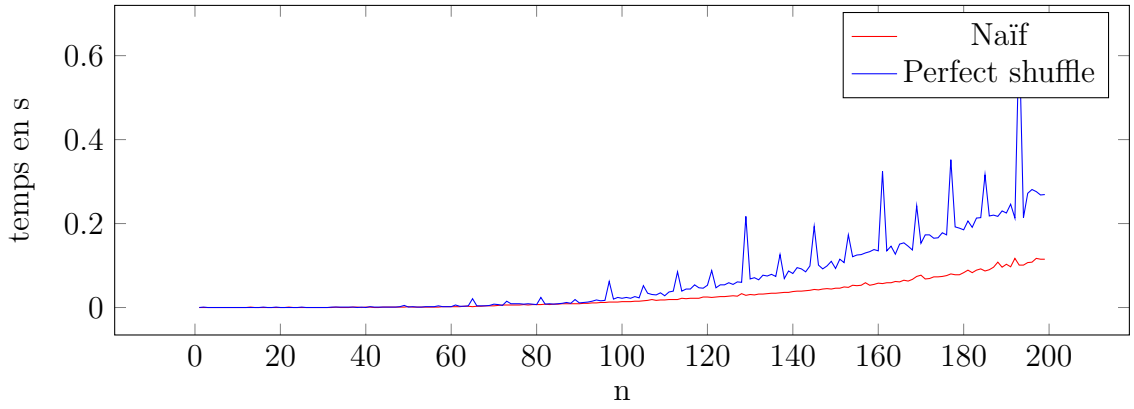


FIGURE 6.7 – Transposition d'un tenseur $n \times n \times n$

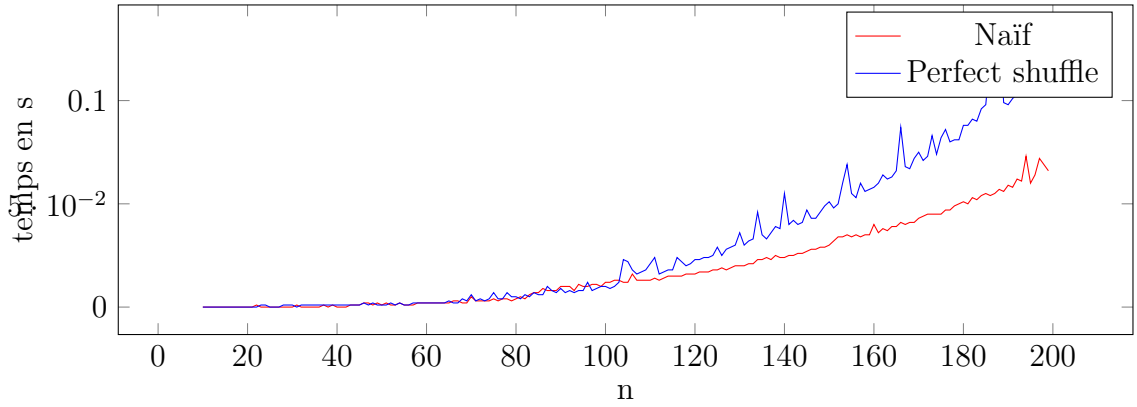


FIGURE 6.8 – Transposition d'un tenseur $n \times n/2 \times (n+1)$

de transposition *in place* pour les matrices, on pourra échanger deux modes voisins. Et comme toute permutation s'écrit comme produit de transpositions voisines, on peut procéder à n'importe quelle transposition tensorielle de cette façon. Au final, on obtient un algorithme relativement rapide, qui mériterait d'être encore un peu optimisé, mais qui évite les copies temporaires. On peut voir dans les figures 6.7 et 6.8 que la durée des transpositions *in place* n'est pas croissante avec la taille des tenseurs. C'est dû au fait que le nombre de graines pour décomposer une transposition dépend en quelque sorte du nombre de facteurs premiers de la taille totale du tenseur. On peut aussi remarquer que l'algorithme naïf est globalement plus rapide que l'algorithme par *perfect shuffle*. Le problème de ce dernier est que, contrairement à l'algorithme naïf, il ne travaille pas sur des données contiguës en mémoire. Cela le ralentit considérablement en pratique, le processeur perdant beaucoup de temps à attendre que les données soient transférées de la mémoire principale au cache (on parle de défaut de cache, ou *cache miss* en anglais).

6.5 Analyse en composantes indépendantes

Une des applications de la diagonalisation des tenseurs les plus étudiées lors de cette thèse a été l'analyse en composantes indépendantes. Le modèle prévoit n

signaux sources $s_i : [0,1] \to \mathbb{R}$ statistiquement indépendants les uns des autres, dont on observe un mélange

$$x(t) = As(t) + n(t) \in R^n, t \in [0,1]\,,$$

où A est une matrice $n \times n$ inversible, et n est un bruit gaussien. L'ICA consiste à retrouver la matrice de mélange A.

6.5.1 Méthodes de calcul

Il existe plusieurs façons de résoudre le problème. Elles diffèrent généralement sur le critère d'indépendance utilisé. Toutes s'accordent sur le fait que la matrice de covariance ne suffit pas, et utilisent généralement des statistiques d'ordre supérieur. On s'intéressera plus particulièrement aux cumulants. Contrairement aux moments d'ordre supérieur, le cumulant d'un vecteur aléatoire aux composantes indépendantes est diagonal.

Le bruit est gaussien, et ne contribue donc pas au cumulant d'ordre 4 des signaux observés. Celui-ci s'écrit

$$\mathcal{C}_4 = \sum_{i=1}^{n} \kappa_i a_i \otimes a_i \otimes a_i \otimes a_i.$$

En diagonalisant C_4, on retrouve donc les colonnes de A, ainsi que le cumulant d'ordre 4 de chaque source. On perd dans le processus l'ordre des sources (on peut changer arbitrairement l'ordre dans la somme ci-dessus), ainsi que l'amplitude de chacune des sources (on peut multiplier chaque vecteur m_i par une constante $\alpha \neq 0$ choisie arbitrairement, à condition de diviser κ_i par α^4).

Plusieurs des algorithmes d'ICA n'utilisent pas l'algorithme de décomposition CP basique 8, mais préfèrent s'intéresser à une collection de matrices issues du cumulant. L'algorithme JADE, pour Joint Approximate Diagonalization Estimation (Cardoso et Souloumiac, 1993), auquel nous nous sommes particulièrement intéressés lors de cette thèse, fait partie de ces derniers. Les matrices à diagonaliser sont générées comme contractions du cumulant par rapport à une série de matrices, souvent une base de $\mathbb{R}^{n\times n}$, notées $\{M_1, \cdots M_{n^2}\}$. Ces contractions s'écrivent ainsi

$$y_{i,j} = \sum_{k,l} c_{i,j,k,l} m_{k,l}.$$

On génère donc n^2 matrices Y_i. On peut réécrire ces contractions comme

$$\begin{aligned}
y^k_{i_1,i_2} &= \sum_{j_1,j_2} c_{i_1,i_2,j_1,j_2} m_{j_1,j_2} \\
&= \sum_{j_1,j_2} \left(\sum_l \kappa_l a_{l,i_1} a_{l,i_2} a_{l,j_1} a_{l,j_2} \right) m^k_{j_1,j_2} \\
&= \sum_l \left[\kappa_l a_{l,i_1} a_{l,i_2} \sum_{j_1,j_2} a_{l,j_1} a_{l,j_2} m^k_{j_1,j_2} \right] \\
&= \sum_l \left(\kappa_l a_{\cdot,l} M^k a^T_{\cdot,l} \right) a_{l,i_1} a_{l,i_2}.
\end{aligned}$$

En posant $d_{ll}^k = \kappa_l a_{.,l} M^k a_{.,l}^T$, on en déduit que A permet de diagonaliser l'ensemble des contractions Y^k générées

$$Y^k = AD^k A^T,\ 1 \leq k \leq n^2.$$

Un des algorithmes possible de décomposition tensorielle passe donc par la diagonalisation simultanée de contractions du tenseur étudié par une base matricielle (De Lathauwer *et al.*, 2004). Dans JADE, on commence par blanchir les données, de façon à chercher une matrice A orthogonale, ce qui facilite le calcul numériquement. Sans ce blanchiment, il faut recourir à une décomposition de Schur simultanée De Lathauwer *et al.* (1996). Plutôt que d'utiliser une base matricielle de $\mathbb{R}^{n\times n}$ pour générer les contractions, il est aussi possible d'utiliser un ensemble de matrices aléatoires (Kuleshov *et al.*, 2015). Cela restreint potentiellement le nombre de matrices à diagonaliser simultanément.

6.5.2 Exemples d'applications

L'ICA permet donc de séparer un mélange de signaux en plusieurs sources aussi indépendantes que possible. En pratique, la notion d'indépendance est trop forte, on lui préfère dans JADE une mesure de la diagonalité du cumulant d'ordre 4. Il est difficile d'imaginer ce que représente une série de signaux dont le cumulant d'ordre 4 est concentré sur la diagonale. Mais en pratique, cela marche plutôt bien. Par exemple dans la figure 6.9, les trois spectres mélangés sont retrouvés pratiquement à l'identique par JADE, modulo l'ordre et l'amplitude des spectres. On donne, à titre de comparaison, les résultats d'autres algorithmes basés sur d'autres critères d'indépendance que JADE. Leurs résultats sont nettement inférieurs pour cet exemple. L'ICA permet aussi de s'essayer au dé-mélange de piles d'images. On déplie chacune des images de la pile à dé-mélanger en un vecteur, puis on applique JADE. Là encore, en pratique cela marche bien, comme on peut le constater sur la figure 6.10.

6.5.3 Pertinence pour le dé-mélange hyper-spectral

L'analyse en composantes indépendantes ne semble guère adaptée au dé-mélange hyper-spectral. En effet cette dernière ne s'intéresse aucunement à la positivité de la décomposition, et est donc très limitée dans son interprétabilité, comme constaté dans Nascimento et Dias (2005b). De plus, les abondances comme les endmembers n'ont aucune raison d'être indépendants les uns des autres. Pourtant plusieurs travaux utilisent cette technique, et il semblait donc important d'y consacrer quelques lignes.

Si l'ICA ne se prête pas telle quelle à l'analyse hyper-spectrale, il est cependant possible de l'utiliser pour calculer les sommets d'un simplexe contenant des points uniformément répartis. Cette observation faite dans Anderson *et al.* (2013) n'est cependant guère utile, les algorithmes de détection des sommets du simplexe, tels SPA ou VCA, retrouvant aisément ces sommets dans ce cas.

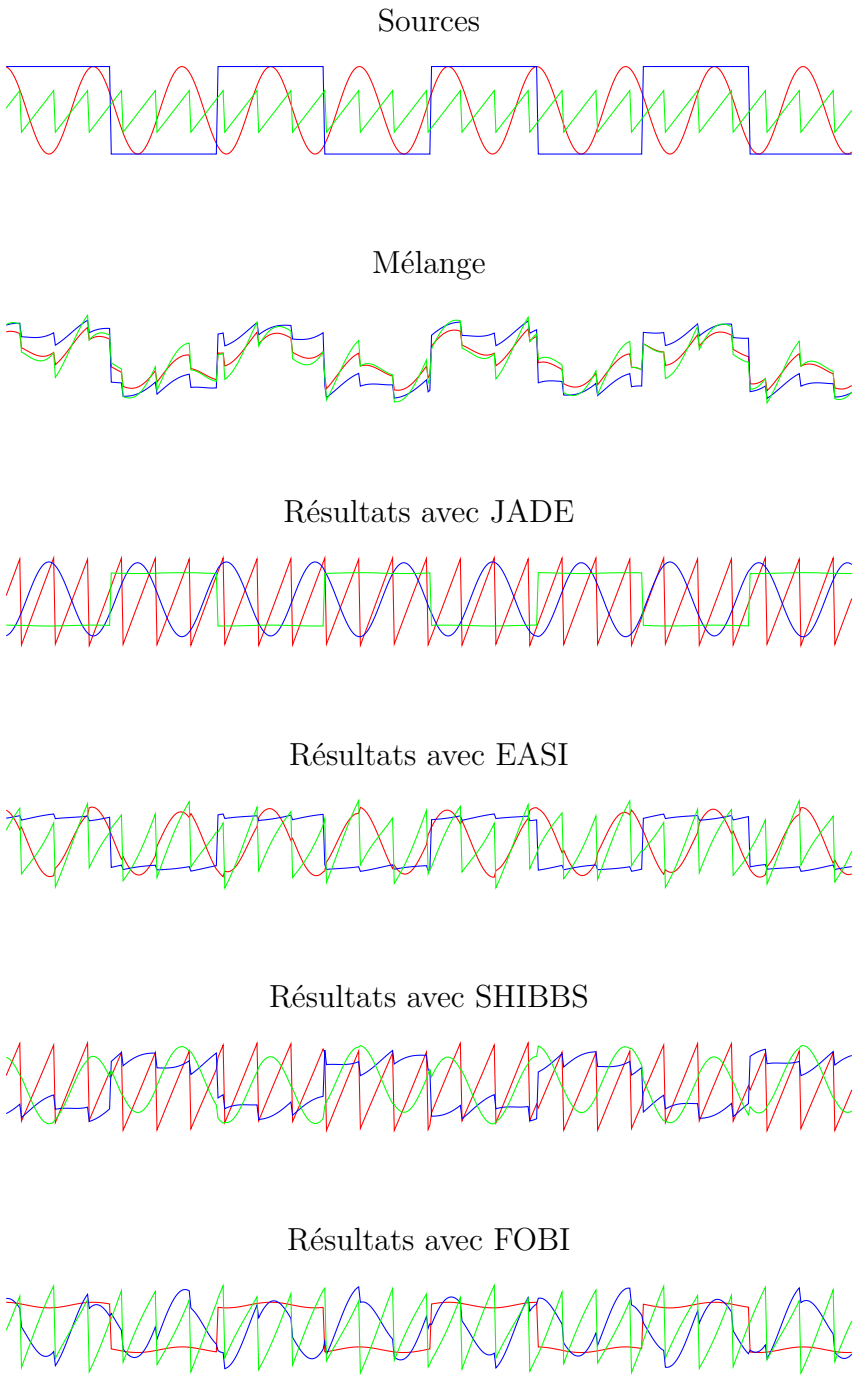


FIGURE 6.9 – Les 3 signaux sources, leur mélange, puis leur dé-mélange avec, respectivement JADE, EASI, SHIBBS, et FOBI.

FIGURE 6.10 – Une pile d'image, mélangée, puis dé-mélangée avec JADE.

6.6 Diagonalisation simultanée

Dans certains cas, on peut donc diagonaliser un tenseur en se contentant de diagonaliser une série de ses contractions, plutôt que de chercher à le diagonaliser directement. On présente dans ce paragraphe trois techniques de diagonalisation simultanée. La première est basée sur des mathématiques relativement simples à exposer, les deux suivantes impliquent des arguments plus théoriques que l'on n'exposera que très sommairement.

6.6.1 Approche classique par Jacobi

La diagonalisation simultanée décrite ici est une extension de la technique de Jacobi : on minimise un critère de diagonalité jointe avec une succession de rotations de Jacobi Cardoso et Souloumiac (1996). Soit un ensemble $A = \{A_k, k = 1, \cdots, K\}$ de K matrices complexes de taille $N \times N$. Si les matrices de A sont normales et commutent, on peut alors les diagonaliser simultanément, par une série de transformations unitaires. On commence par définir :

$$\text{off}\,(A) = \sum_{1 \leq i \neq j \leq N} |a_{ij}|^2 .$$

Le but de la diagonalisation jointe est alors de minimiser

$$\sum_{k=1}^{K} \text{off}\left(UA_kU^H\right)$$

avec U unitaire. On construit alors U comme un produit de rotations planes que l'on applique à toutes les matrices de A. La rotation complexe $R(i,j,c,s)$ égale à l'identité, sauf pour les coefficients $r_{ii} = c$, $r_{ij} = \bar{s}$, $r_{ji} = -s$ et $r_{jj} = \bar{c}$, $c, s \in \mathbb{C}$, vérifiant $|c|^2 + |s|^2 = 1$. On désire trouver s et c, qui, pour $i \neq j$, minimisent la fonction objectif

$$o(c,s) = \sum_{k=1}^{K} \text{off}\left(R(i,j,c,s)\, A_k R^H(i,j,c,s)\right).$$

Pour y parvenir, on introduit une matrice réelle symétrique G de taille 3×3, définie pour une paire d'indices (i,j) par

$$G = \sum_{k=1}^{K} h^H(A_k)\, h(A_k),$$

avec

$$h(A) = \begin{pmatrix} a_{ii} - a_{jj} & a_{ij} + a_{ji} & i(a_{ji} - a_{ij}) \end{pmatrix}$$

Pour tout ensemble de matrices A (pas nécessairement normales, ou commutantes), la fonction objectif $o(c,s)$ est minimisée, sous la contrainte $|c|^2 + |s|^2 = 1$ par

$$c = \sqrt{\frac{x+r}{2r}}, s = \frac{y - iz}{\sqrt{2r(x+r)}}, r = \sqrt{x^2 + y^2 + z^2},$$

où $(x,y,z)^T$ est un vecteur propre associé à la plus grande valeur propre de G.

Comme G est seulement de taille 3×3, on peut trouver une forme analytique de son vecteur propre dominant, mais la stabilité numérique d'une telle expression n'est pas démontrée. On note pour finir qu'il semble intéressant de lancer l'algorithme avec une des matrices de A déjà diagonalisée.

6.6.2 Approche algèbres involutives

Cette approche,développée dans Maehara et Murota (2011), un peu plus théorique, permet la diagonalisation par blocs. Elle se base sur les algèbres involutives, dont on esquisse rapidement les quelques définitions et propriétés nécessaires à son exposition. Ces dernières sont définies comme des sous ensembles de $\mathcal{M}_n(\mathbb{C})$ vérifiant les propriétés suivantes. Pour tout $A, B \in \mathcal{T}, \alpha, \beta \in \mathbb{C}$, on a

$$\begin{cases} \alpha A + \beta B \in \mathcal{T} \\ AB \in \mathcal{T} \\ A^* \in \mathcal{T} \end{cases}$$

La sous-algèbre $\mathcal{T}$ est simple si elle n'a d'autre idéaux que $\{0\}$ et elle-même. Elle est irréductible si elle ne possède d'autre espace invariant que $\{0\}$ et $\mathbb{C}^n$. Toute sous-algèbre involutive de $\mathcal{M}_n(\mathbb{C})$ se décompose comme somme directe de sous-algèbres

simples de $\mathcal{M}_{n_i}(\mathbb{C})$, où $n_i \leq n$ et toute sous-algèbre simple de $\mathcal{M}_n(\mathbb{C})$ se décompose elle même comme somme directe de n_2 copies d'une sous algèbre irréductible de $\mathcal{M}_{n_1}(\mathbb{C})$, et $n = n_1 n_2$. Enfin, une sous algèbre irréductible de $\mathcal{M}_n(\mathbb{C})$ est isomorphe à $\mathcal{M}_n(\mathbb{C})$. On peut donc décomposer toute algèbre involutive comme la somme directe de sous-algèbres irréductibles : il existe une matrice Q unitaire telle que

$$Q^* \mathcal{T} Q = \bigoplus_{i=1}^{l} \mathcal{M}_{n_i}(\mathbb{C}),$$

avec $n_1 + \cdots + n_l \leq n$. On peut donc écrire une matrice de $\mathcal{T}$ comme une matrice diagonale par blocs, avec l blocs de taille $n_1, \cdots, n_l$. Il reste à définir le centre d'une sous-algèbre

$$\text{center}\,(\mathcal{T}) = \{X \in \mathcal{M}_n, \forall A \in \mathcal{T}, AX - XA = 0\}.$$

On a par exemple $\text{center}\,(\mathcal{M}_n(\mathbb{C})) = \mathbb{C} I_n$, et $\text{center}\,(\mathbb{C} I_n) = \mathcal{M}_n(\mathbb{C})$. On a aussi pour tout Q unitaire,

$$\text{center}\,(Q^* \mathcal{T} Q) = Q^* \,\text{center}\,(\mathcal{T})\, Q.$$

Ainsi, pour une sous-algèbre $\mathcal{T}$ décomposable comme somme directe de sous-algèbres isomorphes à $\mathcal{M}_{n_i}(\mathbb{C})$, le centre de $\mathcal{T}$ se décompose comme

$$\text{center}\,(\mathcal{T}) = \bigoplus_{i=1}^{l} \mathbb{C} I_{n_i}.$$

Ainsi, diagonaliser une matrice générique du centre de $\mathcal{T}$, revient à diagonaliser par blocs une matrice générique de $\mathcal{T}$, cf. Maehara et Murota (2011).

Soit $\mathcal{T}$ l'algèbre involutive engendrée par une collection de matrices à diagonaliser simultanément par blocs $A_1, \cdots, A_m$. Pour générer une matrice aléatoirement dans $\text{center}\,(\mathcal{T})$, on s'intéresse donc à la fonction suivante

$$f(X) = \sum_{i=1}^{m} \|A_i X - X A_i\|_F^2 = \|(I_n \otimes A_i - A_i \otimes I_n)\,\text{vec}\, X\|_2^2 .$$

La solution moindres carrés est caractérisée par $\nabla f = 0$, on calcule donc le noyau de T, matrice de $M_{n^2}(\mathbb{C})$, définie comme

$$T = \sum_{i=1}^{m} (I_n \otimes A_i - A_i \otimes I_n)^T (I_n \otimes A_i - A_i \otimes I_n) .$$

Pour obtenir une matrice générique, on peut ainsi prendre une combinaison aléatoire d'une base matricielle du noyau de T. Cette matrice T étant potentiellement de taille très importante, on peut aussi tenter une descente de gradient initialisée aléatoirement sur f.

L'approche, intéressante car permettant une diagonalisation par blocs, demeure globalement plus lente que la diagonalisation par rotation de Jacobi. On pourrait d'ailleurs tout à fait imaginer une diagonalisation par blocs simultanée par rotation de Jacobi, en choisissant à chaque itération une rotation minimisant la norme des blocs hors-diagonale. Cela fait d'ailleurs remarquer que, contrairement à une

méthode type Jacobi, la méthode présentée dans ce paragraphe ne permet de diagonaliser simultanément par blocs qu'une collection de matrices diagonalisables simultanément par blocs (au moins de façon approximative). En effet, un des problèmes de la méthode est qu'elle devient triviale quand le centre de l'algèbre engendrée par notre collection de matrices est réduit à $\{0\}$. Dans ce cas, la diagonalisation par blocs résultante serait constituée de blocs $n \times n$.

6.6.3 Approche par géométrie différentielle

Une autre approche, dont l'idée de départ est finalement relativement proche de la diagonalisation de Jacobi, est de voir le problème de diagonalisation simultanée comme un problème d'optimisation.

$$\min_{Q^TQ=I_n} \left\| \text{off}\left(Q^T A_i Q\right)\right\|_2^2$$

où off A désigne le vecteur des coefficients de A hors diagonale. Plutôt que d'utiliser des techniques d'optimisation contrainte, on adapte des techniques classiques d'optimisation sans contraintes, comme une descente de gradient, la méthode de Newton, ou les méthodes *trust region*. Plutôt que de voir l'ensemble des matrices orthogonales comme un sous ensemble de $\mathcal{M}_n$ dont on essaierait de rester le plus proche possible en cherchant à la fois une solution au problème, on voit l'ensemble de ces matrices comme le seul accessible. On conçoit ainsi des algorithmes d'optimisation de manière à ce que leurs trajectoires restent dans cet ensemble, appelé variété de Stiefel dans la suite de ce paragraphe. Les méthodes d'optimisation doivent donc être adaptées à ce cadre, et il faut ainsi redéfinir certaines notions, comme le gradient, ou la hessienne. En effet, dans une règle de mise à jour très simple, comme celle d'une descente de gradient,

$$x_{k+1} = x_k - \alpha_k \nabla f(x_k),$$

il faut donner un sens précis à cette différence, car la somme de deux matrices orthogonales n'est pas nécessairement orthogonale. Il faut aussi définir le gradient $\nabla f(x_k)$, car à nouveau une définition classique comme

$$f(x+h) = f(x) + \nabla_x f \cdot h + h\epsilon(h)$$

s'adapte mal à ce cadre.

Variété lisse

Pour parer à ces problèmes, il faut donc introduire certaines notions de géométrie différentielle. Ce que l'on fait ici très sobrement. Une variété est un espace topologique localement homéomorphe à l'espace euclidien $\mathbb{R}^n$. Les homéomorphismes locaux, appelés cartes, définissent des systèmes de coordonnées locales. On définit alors un atlas comme une collection de cartes, vérifiant certaines régularités des transitions entre cartes. Quand ces transitions sont lisses, on parle de variété lisse.

Espace tangents et produits scalaires variant continuement sur la variété

En chaque point d'une surface de l'espace classique en 3 dimensions, on peut imaginer un plan tangent à cette surface, plan sur lequel vivrait l'ensemble des vecteurs tangents à ce point. Comme on souhaite définir les choses intrinsèquement, on modifie quelque peu cette image, en définissant un vecteur tangent à un point p d'une variété M comme une application linéaire v de $C^\infty(M)$ dans $\mathbb{R}$ vérifiant

$$v(fg) = f(p)v(g) + g(p)v(f),$$

pour toute fonction $f, g \in C^\infty(M)$. L'ensemble de ces vecteurs tangents forme un espace vectoriel nommé espace tangent en p, que l'on note T_pM. Pour pouvoir parler de gradient, il faut encore définir un produit scalaire sur chaque espace tangent T_pM. Une variété dotée d'une métrique g associant continuement à chaque point p un produit scalaire g_p sur T_pM est appelée variété Riemannienne. On peut alors définir le gradient d'une fonction $f \in C^\infty(M)$ en p comme le seul élément de T_pM vérifiant

$$v(f) = g_p(\text{grad}\, f(p), v), \forall v \in T_pM.$$

Le gradient d'une fonction $f \in C^\infty(M)$ est un exemple de fonction de M dans

$$TM = \bigsqcup_{p \in M} T_pM,$$

le fibré tangent de M.

Champs vectoriels

Ces fonctions $M \to TM$ sont appelées champs vectoriels. Tout champ vectoriel X se doit de vérifier $\pi \circ X = \text{id}$, où π est la projection naturelle de TM sur M. Une autre façon, équivalente, de voir un champ vectoriel sur M est comme une application linéaire $C^\infty(M) \to C^\infty(M)$ vérifiant

$$X(fg) = fX(g) + gX(f), \forall f, g \in C^\infty(M).$$

L'ensemble de ces champs sur M sera noté $\mathfrak{X}(M)$.

Hessienne et connexion

Il manque un dernier ingrédient pour faire de l'optimisation, la hessienne. Pour la définir, il faut pouvoir comparer des vecteurs dans des espaces tangents de points voisins. On introduit pour cela la notion de connexion. Une connexion affine sur M est une fonction

$$\nabla : \mathfrak{X}(M) \times \mathfrak{X}(M) \to \mathfrak{X}(M) : (X, Y) \to \nabla_X Y,$$

vérifiant, pour $X, Y, Z \in \mathfrak{X}(M)$, $f, g \in C^\infty(M)$ et $a, b \in \mathbb{R}$,

$$\begin{aligned} \nabla_{fX+gY} Z &= f\nabla_X Z + g\nabla_Y Z, \\ \nabla_X(aY + bZ) &= a\nabla_X Y + b\nabla_X Z, \\ \nabla_X(fY) &= (Xf)Y + f\nabla_X Y. \end{aligned}$$

Il existe une infinité de connexions ainsi définies. Sur une variété riemannienne, il en existe en revanche une unique vérifiant les conditions supplémentaires

$$\begin{aligned}&\nabla_X Y - \nabla_Y X = [X, Y]\,,\\ &Z\langle X, Y\rangle = \langle \nabla_Z X, Y\rangle + \langle X, \nabla_Z Y\rangle,\end{aligned}$$

appelée connexion de Levi-Civita, ou simplement connexion riemannienne. On définit alors la hessienne d'une fonction $f \in C^\infty$ en $x \in M$ comme une application linéaire de T_xM dans T_xM.

$$\operatorname{Hess} f(x)[v] = (\nabla_X \operatorname{grad} f)(x),$$

où X est un champ vectoriel vérifiant $X(x) = v$.

Pour implémenter une descente de gradient, il manque encore une notion permettant de se déplacer sur une variété en partant de $x_k \in M$ dans la direction donnée par un vecteur de $T_{x_k}M$. On commence par introduire le champ vectoriel des vecteurs tangents à une courbe. Pour une courbe $\gamma \in \mathcal{C}_1$ définie sur $[a, b], a, b \in \mathbb{R}$, on pose $\gamma_t : [a - t, b - t] \to M : u \mapsto \gamma(t + u)$, ce qui définit $\dot\gamma(t) = [\gamma_t] \in T_{\gamma(t)}M$. On dit que γ est une géodésique si, pour un champ vectoriel X vérifiant

$$X(\gamma(t)) = \dot\gamma(t), \forall t \in [a, b],$$

on a $\nabla_X X = 0$ pour tout $t \in [a, b]$. Pour tout $x \in M$, et pour tout $\xi \in T_xM$, il existe une unique géodésique γ définie sur $0 \in I$ telle que $\gamma(0) = x$, et $\dot\gamma(0) = \xi$. On nomme application exponentielle la fonction

$$\operatorname{Exp}_x : \begin{cases} T_xM \to M \\ \xi \mapsto \gamma(1) \end{cases}.$$

Comme 1 n'est pas systématiquement dans I, la fonction Exp_x n'est pas forcément définie sur tout M.

Rétractations et transport parallèle

L'application exponentielle permet de transposer l'algorithme de Newton aux variétés riemanniennes. Son principal défaut est que son calcul n'est pas aisé. On lui préférera donc souvent la notion de rétractation, application lisse du fibré tangent TM vers M, notée R. Pour tout $x \in M$, chaque restriction R_x de cette application à T_xM doit vérifier

1. $R_x(0_x) = x$, où 0_x est le zéro de T_xM,

2. $\mathrm{D}\, R_x(0_x) = \mathrm{id}_{T_xM}$, relation pour laquelle on identifie $T_{0_x}T_xM \cong T_xM$.

On peut remarquer que l'exponentielle est une rétractation.

Algorithme 9 Descente de gradient sur une variété lisse

1: Choisir un point initial $x_0 \in M$
2: **for** $k \in \{0, 1, 2, \cdots, N\}$ **do**
3: Calculer une direction de descente $\xi_k \in T_{x_k}$ et la distance à parcourir dans cette direction $\alpha_k > 0$.
4: Calculer l'itération suivante par $\xi_{k+1} = R_{x_k}(\alpha_k \xi_k)$.
5: **end for**

Optimisation sur les variétés

On a maintenant tout ce qu'il faut pour décrire une descente de gradient sur une variété. L'algorithme typique est décrit dans 9. En choisissant $\xi_k = \operatorname{grad} f(x_k)$, on définit une descente de gradient, pour laquelle il faut déterminer un pas $\alpha_k > 0$. On peut, comme dans les espaces euclidiens, déduire le pas de certaines approximations (Gauss-Newton, Gradient conjugué, Fletcher-Reeves, Polak-Ribière...), ou utiliser l'itération de Newton. Si on choisit Newton, il faut alors résoudre, de façon analogue au cas euclidien,

$$\operatorname{Hess} f(x_k)\xi_k = -\operatorname{grad} f(x_k).$$

Pour que la solution soit une direction de descente, il faut que la Hessienne soit définie positive, dans le sens $\langle \xi, \operatorname{Hess} f(x_k)[\xi] \rangle > 0, \forall \xi \neq 0_{x_k}$. C'est systématiquement le cas, lorsque l'on utilise la connexion riemannienne, si $\operatorname{Hess} f(x_k)$ a des valeurs propres strictement positives. Dans la pratique, quand ce n'est pas le cas, on pourra par exemple rendre la hessienne définie positive artificiellement, en lui ajoutant un multiple de l'identité. La vitesse de convergence peut être influencée par ce genre de pratique.

Un exemple : la variété de Stiefel

Le but de toute cette introduction à la géométrie différentielle est la diagonalisation jointe. Pour rappel, elle consiste à trouver une matrice orthogonale diagonalisant au mieux un ensemble de matrices. Pour proposer un algorithme type descente de gradient résolvant le problème, il faut donc s'intéresser à la variété de Stiefel, variété constituée des matrices orthogonales d'un certain rang. On commence par déterminer l'expression du plan tangent en $X \in \operatorname{St}(p, n)$. Comme $\operatorname{St}(p, n) = \ker F$, où

$$F : \begin{cases} \mathbb{R}^{n \times p} \to \mathbb{R}^{p \times p} \\ X \mapsto X^T X - I \end{cases},$$

on peut obtenir $T_X \operatorname{St}(p, n)$ comme $\ker \mathrm{D}\, F(X)$. On a donc

$$T_X \operatorname{St}(p, n) = \{Z \in \mathbb{R}^{n \times p}, X^T Z + Z^T X = 0\}.$$

Une autre façon d'obtenir le plan tangent est de différencier une courbe de $\operatorname{St}(p, n)$ passant par X. On peut alors obtenir une expression différente de la précédente,

$$T_X \operatorname{St}(p, n) = \{X\Omega + X_\perp K, \Omega^T = -\Omega, K \in \mathbb{R}^{n \times (n-p)}\},$$

où $X_\perp$ est une base de $(\text{span}(X))^\perp$. On peut alors calculer un gradient en le projetant sur l'espace tangent. En effet, si on choisit la métrique riemannienne héritée de $\mathbb{R}^{n\times p}$, qui à $\xi, \eta \in T_X \,\text{St}(p,n)$ associe $\text{tr}\Big(\xi^T\eta\Big)$, on peut alors montrer que

$$(T_X \,\text{St}(p,n))^\perp = \{XS, S \in \mathcal{S}_p\},$$

et donc que la projection orthogonale de Y sur $T_X \,\text{St}(p,n)$ s'écrit

$$P_X = (I - XX^T)Y + X \,\text{skew}\Big(X^TY\Big),$$

où $\text{skew}(M) = \frac{1}{2}(M - M^T)$. Pour calculer le gradient d'un champ scalaire f sur $\text{St}(p,n)$ en X, il suffit alors de projeter son gradient classique, celui de $\mathbb{R}^{n\times p}$, via P_X.

Quand M est une sous-variété riemannienne de $\bar{M}$, leurs connexions riemanniennes sont liées par

$$\nabla_{\eta_x}\xi = P_x(\bar{\nabla}_{\eta_x}\xi),$$

pour tout $\eta_x \in T_xM$, et $\xi \in \mathfrak{X}(M)$. On en déduit que la connexion riemannienne de $\text{St}(p,n)$ est

$$\nabla_{\eta_X}\xi = P_X(\text{D}\,\xi(X)[\eta_X]),$$

avec $X \in \text{St}(p,n)$, $\eta_X \in T_X \,\text{St}(p,n)$, et $\xi \in \mathfrak{X}(\text{St}(p,n))$.

Une rétractation possible sur $\text{St}(p,n)$ est définie comme le facteur Q de la décomposition QR. Pour garantir que cette rétractation soit bien une fonction, on utilise l'unicité d'une telle décomposition, quand elle porte sur des matrices de rang plein, et que l'on s'impose des coefficients positifs sur la diagonale de R. Pour $\xi \in T_X \,\text{St}(p,n)$, on a ainsi défini

$$R_X(\xi) = \text{qf}(X + \xi).$$

Une autre technique de diagonalisation jointe On a maintenant tout ce qu'il faut pour diagonaliser simultanément des matrices. L'objectif est de minimiser la somme des coefficients hors diagonale d'un ensemble de matrices $A_1, \cdots A_K$. On pose donc

$$f : \text{St}(p,n) \to \mathbb{R}, X \mapsto \sum_{k=1}^{K} \Big\|\text{diag}(X^TA_kX)\Big\|_F^2.$$

Le but du jeu est alors de résoudre $\min f(X)$, sur son domaine de définition. On peut étendre ce domaine à $\mathbb{R}^{n\times p}$, ce qui définit alors une fonction $\bar{f}$, dont on peut calculer le gradient de façon classique. On a

$$\text{grad}\Big(\bar{f}\Big) = -4\sum_{k=1}^{K} A_kX \,\text{diag}(X^TA_kX),$$

que l'on projette sur $T_X \,\text{St}(p,n)$ pour obtenir

$$\text{grad}(f) = -4\sum_{k=1}^{K} A_kX \,\text{diag}(X^TA_kX) - X \,\text{sym}\left(X^TA_kX \,\text{diag}(X^TA_kX)\right).$$

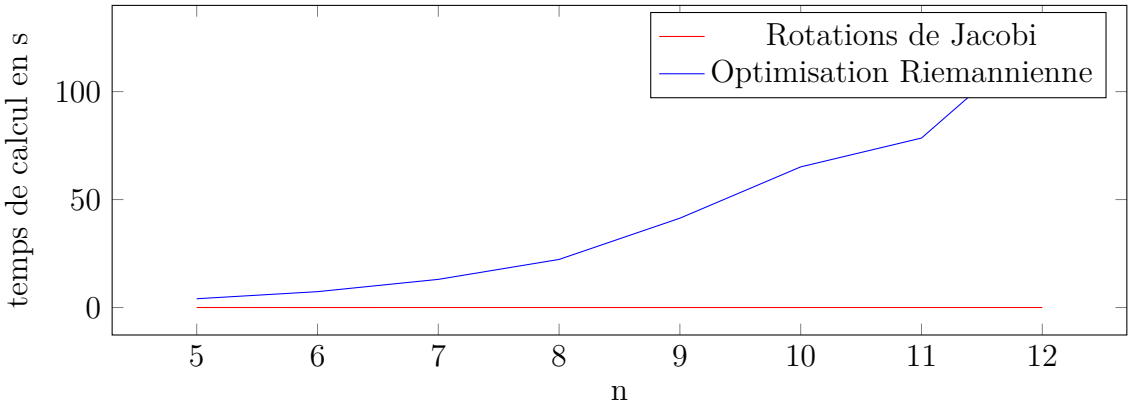


FIGURE 6.11 – Temps de calcul pour la diagonalisation simultanée de n matrices de taille $n \times n$

On obtient alors la hessienne comme

$$\operatorname{Hess} f(X)[\xi] = P_X(\mathrm{D}\operatorname{grad} f(X)[\xi]).$$

Comme $X \operatorname{sym}(\xi^T \operatorname{grad} \bar{f}(X)) + X \operatorname{sym}(X^T \mathrm{D}\operatorname{grad} \bar{f}(X)[\xi]) \in T_X \operatorname{St}(p,n)^{\perp}$, on a

$$\operatorname{Hess} f(X)[\xi] = P_X(\mathrm{D}\operatorname{grad} \bar{f}(X)[\xi]) - \xi \operatorname{sym}(X^T \operatorname{grad} \bar{f}(X)),$$

avec

$$\mathrm{D}\operatorname{grad} \bar{f}(X)[\xi] = -4 \sum_{k=1}^{K} A_k \left(\xi \operatorname{diag}(X^T A_k X) + 2X \operatorname{diag}(X^T A_k \xi) \right).$$

Pour plus de précisions, on peut se référer à l'article de Sato (2014), qui utilise l'algorithme de Newton en utilisant une astuce réduisant la taille de la hessienne, ou à celui de Absil Theis *et al.* (2009) qui utilise un algorithme *trust–region*. On donne une idée des temps de calcul via la méthode de Newton dans la figure 6.11. Le constat est sans appel, la méthode riemannienne est pour l'instant beaucoup plus lente. Cela est dû au calcul de la décomposition en valeurs propres de la hessienne, de taille $n^2 \times n^2$ à chaque itération. Les algorithmes d'optimisation riemannienne se passant de hessienne semblent donc bien plus intéressants.

7 Mise en œuvre et implémentation

Dans les chapitres précédents, nous avons souvent écarté le côté implémentation. Cela a pourtant représenté une partie importante de cette thèse. Nous essayons dans ce chapitre de donner un aperçu de cette facette du travail.

7.1 Démonstrateur

Une part importante de ce travail de thèse a consisté à tester différents algorithmes élaborés pour résoudre les diverses problématiques décrites au fil des chapitres précédents. Pour une part importante de ceux-ci, l'implémentation a dépassé le cadre du simple prototype Matlab/Scilab. Pour pouvoir comparer les solutions proposées de façon plus réaliste, en situation, la plupart ont été implémentées en C++. Cela implique plusieurs choses.

7.1.1 Librairies de calcul

Cela demande d'abord de développer des classes de calcul matriciel/tensoriel. Pour plusieurs des fonctions proposées par ces classes, comme le produit matriciel, ou certains algorithmes de parcours de graphes par exemple, il existe souvent déjà des implémentations efficaces disponibles dans certaines librairies de calcul. Il s'agit alors de comparer les méthodes proposées, et de s'assurer que les licences des librairies utilisées autorisent leur utilisation dans le cadre d'un logiciel commercial. Plusieurs de ces librairies sont en effet placées sous une licence assez restrictive, type GNU, qui exige de tout logiciel les incluant d'être eux mêmes libres, c'est à dire que leur code source doit être disponible publiquement. On doit alors s'en passer.

Un exemple important de ces fonctions de base est le produit matriciel. Une version naïve est très simple à développer : trois boucles imbriquées suffisent pour atteindre une complexité de $O(n^3)$. Il existe d'autres algorithmes de complexité moindre, mais ils ne sont généralement pas ceux implémentés en pratique. Ces derniers sont en effet plus difficile à mettre en œuvre, et leur gain en complexité asymptotique n'est intéressant que pour le produit de matrices gigantesques. Pour gagner en temps de calcul, les différentes implémentations disponibles sont donc attentives à des considérations plus pratiques. Elles veillent par exemple à la localité des données

dans le cache du processeur, exploitent les différents cœurs des processeurs modernes en parallélisant l'algorithme, transforment la structure en mémoire des matrices à multiplier en les ré-écrivant bloc par bloc, déroulent certaines des boucles, ou encore tirent parti des instructions SIMD disponibles sur certains processeurs permettant en une seule instruction de calculer plusieurs opérations. Ces différentes considérations impliquent donc de devoir écrire plusieurs implémentations, chacune adaptée à un type précis de processeur. Ces dernières sont généralement écrites sur mesure en assembleur. Un compte-rendu de ce type de travail est donné dans Goto et Geijn (2008).

Il est ainsi généralement inintéressant de re-développer soi même ce type de fonctionnalité élémentaire. Le même genre de problème existe pour d'autres opérations algébriques de base. Ces différentes opérations sont spécifiés dans BLAS, pour Basic Linear Algebra Subprograms. On y retrouve par exemple le produit scalaire, le produit matrice-vecteur, le produit externe, les *updates* de rang 1, ou le produit matriciel. L'interface proposée par BLAS est relativement peu pratique à utiliser telle quelle. C'est pourquoi il existe différents *wrappers* de ces fonctions facilitant leur utilisation. Le prototype de la fonction BLAS de produit matriciel, par exemple, est en effet donné comme :

```
void cblas_dgemm( const enum CBLAS_ORDER Order,
                  const enum CBLAS_TRANSPOSE TransA,
                  const enum CBLAS_TRANSPOSE TransB,
                  const blasint M,
                  const blasint N,
                  const blasint K,
                  const double alpha,
                  const double *A,
                  const blasint lda,
                  const double *B,
                  const blasint ldb,
                  const double beta,
                  double *C,
                  const blasint ldc );
```

Ce genre d'interface complexe, au nom peu évocateur, est susceptible d'entraîner des erreurs dans le code appelant, et il est assez pénible de devoir décliner cette longue liste d'arguments à chaque produit matriciel. Nous avons donc développé une classe de calcul matriciel encapsulant ces fonctions.

Les différentes fonctions de bases décrites par BLAS sont implémentées par diverses librairies, dont :

- NetlibBlas : l'implémentation officielle de référence, avec une licence très permissive,
- OpenBlas : une implémentation efficace basée sur GotoBlas, sous licence BSD,
- MKL : une autre implémentation efficace, commerciale, pour laquelle il existe toutefois des licences gratuites sans *royalties*.

Il existe encore d'autres librairies, certaines étendant les spécifications de BLAS. BLIS (Van Zee et van de Geijn, 2015) propose par exemple la possibilité de spécifier un *stride* plus général, ce qui peut être intéressant pour la contraction de tenseurs.

D'autres procédures importantes en algèbre linéaire, absentes de BLAS, sont regroupées dans LAPACK, librairie sous licence BSD modifiée. On y trouve par exemple des méthodes d'inversion de matrice, de décomposition QR, ou de diagonalisation. Nous avons mentionné section 1.3.2 que l'on pouvait calculer une décomposition en valeurs propres à partir de rotations de Jacobi. Cette méthode est en fait très sous-optimale. Il existe une autre méthode, plus difficile à mettre en œuvre, qui commence par réduire la matrice étudiée à une forme tri-diagonale, avant de calculer les valeurs et vecteurs propres par une approche type "diviser pour régner". Cette procédure, ainsi que beaucoup d'autres procédures d'algèbre linéaire décrites dans Golub et Van Loan (2012), font également partie de LAPACK.

Un autre exemple de primitive de calcul qu'il est préférable d'importer d'une librairie est la transformée de Fourier rapide. La FFT version Cooley–Tuckey pour un signal de taille $n = km$ s'écrit simplement comme

$$\mathrm{DFT}_{km} = (\mathrm{DFT}_k \otimes I_m)\, T_m^n\, (I_k \otimes \mathrm{DFT}_m)\, L_k^n,$$

où T_m^n est une matrice diagonale, et L_k^n est une matrice permettant de transposer une matrice $k \times (n/k)$. Mais à nouveau, pour un calcul aussi efficace que possible, il est préférable de passer par une librairie de calcul. Plusieurs sont disponibles :

- FFTW, pour Fastest Fourier Transform in the West, est l'une des plus connues, mais est sous licence GNU,
- MKL propose également des fonctions de calcul de FFT, réputées rapides elles-aussi,
- Kiss-FFT, qui est loin d'être la plus rapide, mais qui a l'avantage d'être très simple, est sous licence BSD.

Enfin, pour certaines fonctionnalités, nous avons estimé que ce qui était proposé dans les librairies existantes ne nous convenait pas. C'est par exemple le cas pour le calcul tensoriel, pour lequel la plupart des librairies disponibles sont conçues autour du concept d'indice, et sont orientées vers des applications issues de la physique (type relativité générale ou renormalisation, cf. Itensor [1] et Ftensor [2]). Nous avons préféré développer cette classe par nous mêmes, en nous appuyant largement sur les *templates*, et en nous inspirant de la classe matrice de Eigen (Guennebaud *et al.*, 2010), ainsi que des idées développées par Aragón (2014). Nous avons aussi développé certains algorithmes d'optimisation, comme ADMM, ou L-BFGS. L'un des points de vigilance lorsque l'on code un tel algorithme, utilisé à de nombreuses reprises dans le code par la suite, est son exactitude.

1. Disponible en ligne à http://itensor.org
2. Disponible en ligne à http://www.wlandry.net/Projects/FTensor

7.1.2 Tests unitaires

Pour s'assurer que le code développé est correct, nous avons dû procéder régulièrement à des tests unitaires. Les méthodes disponibles dans les classes de calcul ne doivent en effet comporter aucune erreur, car elles constituent la base sur laquelle on construit d'autres algorithmes, souvent beaucoup plus élaborés. Ces méthodes doivent donc être exemptes de tout doute concernant la validité de leurs résultats. On présente dans ce paragraphe quelques idées de tests unitaires, à travers deux exemples. L'un relativement simple, concernant le produit matriciel, et l'autre un peu plus compliqué, concernant la décomposition en valeurs singulières.

Exemple 7.1.1. *Pour tester le produit matriciel, on pourra par exemple vérifier que le résultat du produit entre une matrice A fixée et une matrice B fixée elle aussi est conforme au résultat attendu. On peut aussi essayer de multiplier deux matrices aux dimensions incompatibles, et vérifier que la fonction signale bien qu'il y a un problème. On peut encore générer trois matrices aléatoires $A \in \mathbb{R}^{m\times k_1}, B \in \mathbb{R}^{k_1\times k_2}$ et $C \in \mathbb{R}^{k_1\times n}$, et $\lambda \in \mathbb{R}$, et vérifier que certaines des propriétés du produit matriciel sont bien vérifiées. On regardera par exemple si l'on a bien*

$$\begin{cases} (AB)^T = B^T A^T, \\ (AB)C = A(BC), \\ \lambda(AB) = (\lambda A)B = A(\lambda B), \\ A(B+C) = AB + BC, \\ (A+B)C = AC + BC, \\ \mathrm{tr}(AB) = \mathrm{tr}(BA). \end{cases}$$

Ces égalités matricielles ne seront la plupart du temps qu'approximativement vraies. Du fait des imprécisions numériques, pour vérifier si une matrice A est égale à une autre B, on se contentera de vérifier si $\|A - B\|_F < 10^{-14}$.

Comme les matrices peuvent être représentées de différentes façons, sous forme row-major ou column-major, on testera les propriétés évoquées dans cet exemple dans les deux formats.

Exemple 7.1.2. *Dans certains cas, les propriétés à tester sont évidentes. Pour tester la SVD d'une matrice $M = USV^T$ choisie aléatoirement, on vérifiera simplement que*

$$\begin{cases} M = USV^T, \\ S = \mathrm{diag}\,\mathrm{diag}\, S, \\ U^T U = I, \\ V^T V = I. \end{cases}$$

7.1.3 Interface

Afin de tester les différentes méthodes à l'intérieur de Mountains, nous avons développé une interface avec le logiciel. Cela a nécessité que les structures de données développées dans les classes de calcul soient compatibles avec celles utilisées dans

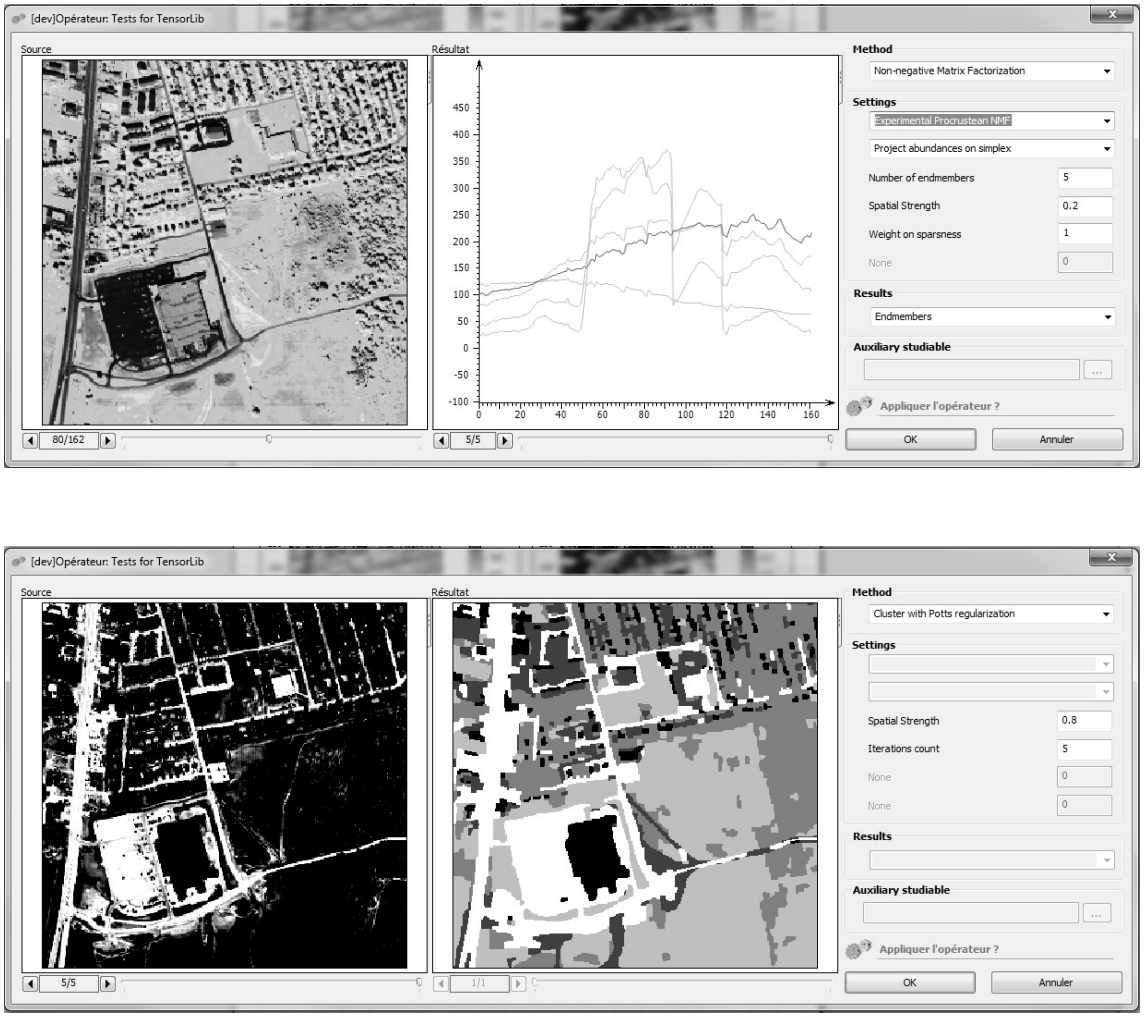


FIGURE 7.1 – Aperçus de l'interface

Mountains. Le fonctionnement de Mountains est basé sur des opérateurs/études et des étudiables. Un étudiable Mountains peut être par exemple un profil, une image, une surface, un hypercube, etc. En fonction du type de l'étudiable, certains opérateurs/études sont disponibles. On imagine en effet difficilement faire du dé-mélange hyper-spectral sur un profil. Ce genre de distinction est respectée dans le démonstrateur. Cela permet aussi de différencier les modalités d'application de certaines techniques développées en fonction du type de l'étudiable. L'analyse en composantes indépendantes s'applique par exemple différemment sur les séries de spectres et sur les hypercubes. Une fenêtre permettant de sélectionner quelle méthode appliquer, ainsi que toutes les modalités relatives à cette dernière, a été développée. Même si cette dernière ne propose qu'une interface minimaliste, il faut veiller à ce que pour chaque méthode disponible, les choix concernant les paramètres et les options sélectionnés soient cohérents.

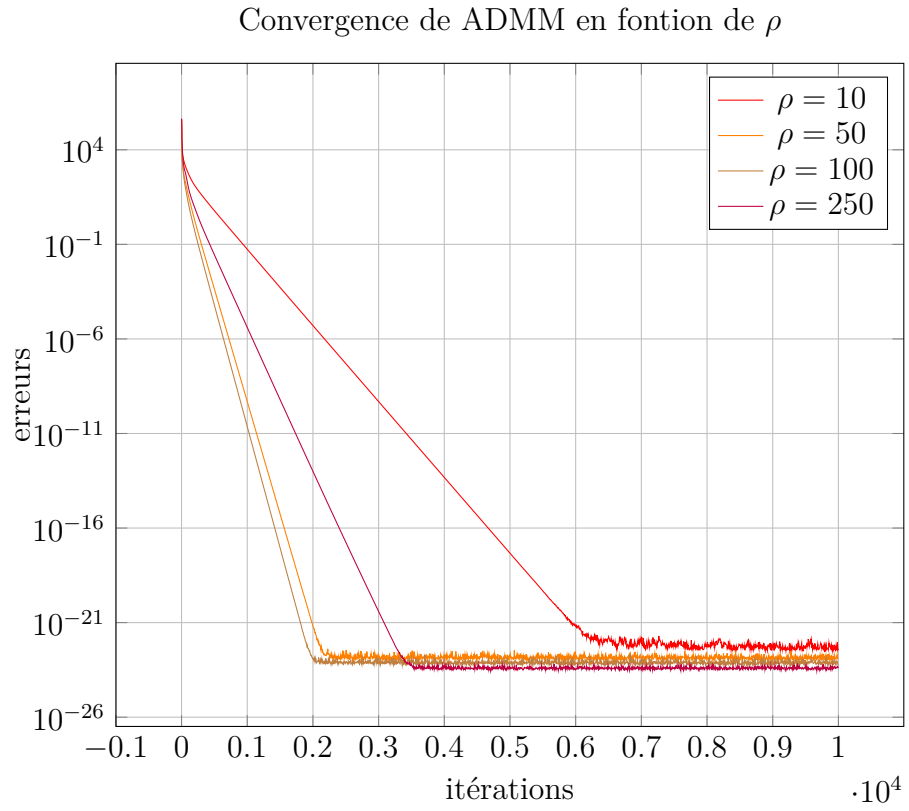


FIGURE 7.2 – Convergence d'un algorithme ADMM pour une NMF en fonction de ρ.

7.2 Réglage des paramètres et sélection des algorithmes

Certains algorithmes demandent un réglage fin de leurs paramètres, paramètres parfois difficilement interprétables. Un exemple récurrent est le réglage du paramètre de lagrangien augmenté dans les algorithmes ADMM utilisés. Dans les bonnes conditions, ADMM converge, quelle que soit la valeur de ce paramètre. Cela peut sembler rassurant. Cependant, en pratique, cette valeur a une influence importante sur la vitesse de convergence de l'algorithme. Et on peut difficilement laisser à l'utilisateur le soin d'effectuer ce réglage, sans lui expliquer les détails d'implémentation d'un algorithme ADMM. De plus, il semble difficile de calculer de façon théorique le meilleur réglage possible de ce paramètre. De telles études existent seulement dans certains cas particuliers, comme la programmation quadratique (Ghadimi *et al.*, 2015). Le réglage se fait donc souvent de façon expérimentale. On teste simplement un algorithme un certain nombre de fois avant de trouver le bon réglage. Certaines publications se contentent d'ailleurs souvent d'indiquer une valeur semblant bien fonctionner lors d'expérimentations. C'est le cas dans par exemple dans Zhang (2010); Warren et Osher (2015); Iordache *et al.* (2012); Faivre et Dombry (2017).

Une multitude d'algorithmes ont été développés, et tous ne seront pas dispo-

nibles pour l'utilisateur. On peut en effet imaginer que disposer d'une vingtaine d'algorithmes pour la NMF par exemple ne serait pas forcément d'un grand secours pour les utilisateurs. Il faudrait expliquer les différences entre ceux-ci, différences quelquefois assez subtiles. On peut d'ailleurs imaginer que le choix des algorithmes disponibles ne se fera pas uniquement sur une base purement technique, en ne prenant en compte que la vitesse de convergence ou la qualité des résultats, mais que des critères comme la difficulté d'explication du fonctionnement de ces algorithmes, ou même leur réputation devront également être pris en compte. Pouvoir expliquer de façon claire et concise les différents critères et garanties offerts par un algorithme participe à la lisibilité du logiciel.

7.3 Encore d'autres problèmes

Digital Surf n'est pas une entreprise focalisée uniquement sur l'analyse hyperspectrale. Aussi durant cette thèse, des problèmes très divers, issus de questions parfois assez éloignées du sujet de cette thèse ont été rencontrés. On en donne ici quelques exemples, retenus pour leur intérêt mathématique.

Un des modules disponibles dans Mountains analyse les contours de formes diverses, et permet de comparer ces formes entre elles. Cela engendre une multitude de problèmes d'optimisation, une des idées sous-jacentes étant souvent de positionner au mieux les deux formes l'une par rapport à l'autre, avant de mesurer certains écarts entre ces deux formes. On doit donc résoudre un problème d'optimisation où la variable doit appartenir au groupe orthogonal, voire même au groupe spécial orthogonal. Ce type de problème, souvent appelé problème de Procuste, se résout relativement aisément grâce à une SVD quand il ne présente pas de points aberrants. On peut rencontrer encore d'autres problèmes liés à l'analyse des contours. L'un d'entre eux consiste à ajuster une forme géométrique à un nuage de points. Pour une ellipse par exemple, si tous les points à ajuster lui appartiennent exactement, le problème se résout par une simple SVD.

Une autre difficulté soulevée par l'étude des contours consiste à calculer l'intersection de certaines courbes. On peut par exemple avoir besoin de calculer les intersections de deux ellipses E_1 et E_2, décrites par

$$\begin{cases} E_1 = \{(x,y) \in \mathbb{R}^2, (x\ y\ 1) A_1 (x\ y\ 1)^T = 0\}, \\ E_2 = \{(x,y) \in \mathbb{R}^2, (x\ y\ 1) A_2 (x\ y\ 1)^T = 0\}. \end{cases}$$

Cela peut se faire sans trop de calcul, et de façon exacte, en s'intéressant aux valeurs critiques du faisceau de matrices $\mu A_1 + \lambda A_2$. En choisissant une des valeurs critiques réelles, on produit une ellipse dégénérée dont les points d'intersection avec E_1 ou E_2 sont les mêmes que les points recherchés. Il ne reste donc plus qu'à calculer l'intersection entre une paire de droites et une ellipse, un problème bien plus simple que le problème initial.

D'autres courbes que les ellipses interviennent également comme contours. Certaines applications nécessitent par exemple le calcul de courbes parallèles ainsi que de leurs intersections. Deux courbes sont parallèles si ce sont les lieux des extrémités d'un segment de longueur constante se déplaçant de façon orthogonale à sa

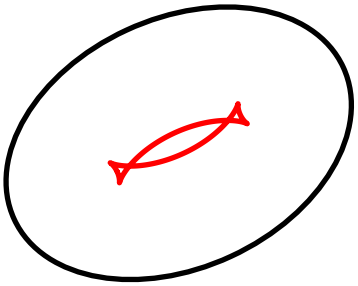


FIGURE 7.3 – Courbe parallèle à une éllipse

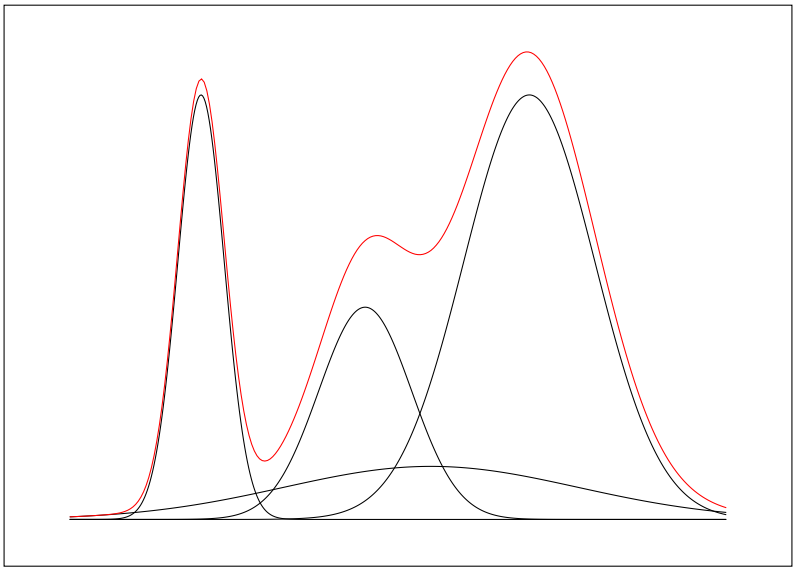


FIGURE 7.4 – Un signal décomposé en somme de gaussiennes.

direction. Nous avons dessiné un exemple de courbe parallèle à une ellipse sur la figure 7.3. De telles courbes, appelées parfois tétracuspides ou toroïdes, sont décrites par des équations de degré huit. Il est donc difficile de calculer algébriquement leurs intersections.

Un dernier exemple de problème rencontré, hors du strict cadre du sujet, consiste en la décomposition d'un spectre en somme de gaussiennes. Un exemple d'une telle décomposition est présenté figure 7.4. Quand on ne connaît pas à priori le nombre de gaussiennes, il faut l'estimer. Là encore, on a à faire à un problème d'optimisation intéressant.

Conclusion

Les travaux menés durant cette thèse ont eu pour objet principal l'exploration des diverses techniques d'analyse hyper-spectrale.

Nous avons commencé par étudier les limites de *k-means*, une technique de *clustering* permettant de regrouper chacun des spectres d'un cube hyper-spectral dans des classes homogènes. Après avoir expliqué les raisons de l'existence de ces limites, nous avons introduit une nouvelle méthode basée sur l'optimisation SDP dont nous avons étudié de façon théorique l'exactitude, lorsque les données à traiter sont issues d'un mélange de gaussiennes. Cette nouvelle méthode a fait l'objet d'un article soumis pour publication à *Statistics and Computing*.

Nous avons ensuite abordé le problème de dé-mélange hyper-spectral, dont l'objectif est de décomposer un hyper-cube comme le produit de signatures spectrales de base par une matrice décrivant les proportions de chacune de ces signatures pour chacun des spectres du cube. Ce problème correspond au problème de factorisation en matrices positives, appelé aussi NMF, dont nous avons décrit les principaux algorithmes de résolution disponibles aujourd'hui. La complexité de ce problème ne nous a pas permis de proposer une relaxation efficace du problème, comme cela a été le cas pour le *clustering*. Nous avons alors proposé une méthode de régularisation du problème exploitant la régularité spatiale affichée par les cubes. Ce travail a donné lieu à un autre article, lui aussi soumis pour publication.

Nous avons finalement abordé plusieurs autres problèmes d'analyse hyper-spectrale. Nous avons proposé une méthode de classification bâtie sur les résultats obtenus par dé-mélange, exploitant là encore l'information spatiale contenue typiquement dans un cube. Enfin, nous nous sommes intéressés à des méthodes de dé-mélange non-linéaires en étudiant certaines techniques de réduction non-linéaires de la dimension.

Certaines des techniques développées lors de cette thèse sont récemment sorties du cadre strictement expérimental, pour intégrer un opérateur qui sera bientôt disponible dans Mountains, le logiciel d'analyse d'images développé par Digital Surf.

Bibliographie

Aaronson, S., Kuperberg, G. et Granade, C. (2015). The complexity zoo.

Aloise, D., Deshpande, A., Hansen, P. et Popat, P. (2009). Np-hardness of euclidean sum-of-squares clustering. *Machine learning*, 75(2):245–248.

Anandkumar, A., Ge, R., Hsu, D. et Kakade, S. (2013). A tensor spectral approach to learning mixed membership community models. *In Conference on Learning Theory*, pages 867–881.

Anandkumar, A., Ge, R., Hsu, D., Kakade, S. M. et Telgarsky, M. (2014). Tensor decompositions for learning latent variable models. *Journal of Machine Learning Research*, 15:2773–2832.

Anderson, J., Goyal, N. et Rademacher, L. (2013). Efficient learning of simplices. *In Conference on Learning Theory*, pages 1020–1045.

Aragón, A. M. (2014). A c++ 11 implementation of arbitrary-rank tensors for high-performance computing. *Computer Physics Communications*, 185(6):1681–1696.

Araújo, M. C. U., Saldanha, T. C. B., Galvao, R. K. H., Yoneyama, T., Chame, H. C. et Visani, V. (2001). The successive projections algorithm for variable selection in spectroscopic multicomponent analysis. *Chemometrics and Intelligent Laboratory Systems*, 57(2):65–73.

Arora, S. et Barak, B. (2009). *Computational complexity : a modern approach*. Cambridge University Press.

Arora, S., Ge, R., Kannan, R. et Moitra, A. (2012). Computing a nonnegative matrix factorization–provably. *In Proceedings of the forty-fourth annual ACM symposium on Theory of computing*, pages 145–162. ACM.

Arthur, D. et Vassilvitskii, S. (2007). k-means++ : The advantages of careful seeding. *In Proceedings of the eighteenth annual ACM-SIAM symposium on*

Discrete algorithms, pages 1027–1035. Society for Industrial and Applied Mathematics.

Bahmani, B., Moseley, B., Vattani, A., Kumar, R. et Vassilvitskii, S. (2012). Scalable k-means++. *Proceedings of the VLDB Endowment*, 5(7):622–633.

Bauckhage, C. (2014). A purely geometric approach to non-negative matrix factorization. *In LWA*, pages 125–136.

Bioucas-Dias, J. M. (2009). A variable splitting augmented lagrangian approach to linear spectral unmixing. *In Hyperspectral Image and Signal Processing : Evolution in Remote Sensing, 2009. WHISPERS'09. First Workshop on*, pages 1–4. IEEE.

Bioucas-Dias, J. M., 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.

Boucheron, S., Lugosi, G. et Massart, P. (2013). *Concentration inequalities*. Oxford University Press, Oxford. A nonasymptotic theory of independence, With a foreword by Michel Ledoux.

Boumal, N., Voroninski, V. et Bandeira, A. (2016). The non-convex burer-monteiro approach works on smooth semidefinite programs. *In Advances in Neural Information Processing Systems*, pages 2757–2765.

Boutsidis, C. et Gallopoulos, E. (2008). Svd based initialization : A head start for nonnegative matrix factorization. *Pattern Recognition*, 41(4):1350–1362.

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.

Boykov, Y., Veksler, O. et Zabih, R. (2001). Fast approximate energy minimization via graph cuts. *IEEE Transactions on pattern analysis and machine intelligence*, 23(11):1222–1239.

Bro, R. (1997). Parafac. tutorial and applications. *Chemometrics and intelligent laboratory systems*, 38(2):149–171.

Burer, S. (2009). On the copositive representation of binary and continuous nonconvex quadratic programs. *Mathematical Programming*, 120(2):479–495.

Burer, S. et Monteiro, R. D. (2003). A nonlinear programming algorithm for solving semidefinite programs via low-rank factorization. *Mathematical Programming*, 95(2):329–357.

Candès, E. J., Li, X., Ma, Y. et Wright, J. (2011). Robust principal component analysis ? *Journal of the ACM (JACM)*, 58(3):11.

Cardoso, J.-F. et Souloumiac, A. (1993). Blind beamforming for non-gaussian signals. *In IEE proceedings F (radar and signal processing)*, volume 140, pages 362–370. IET.

Cardoso, J.-F. et Souloumiac, A. (1996). Jacobi angles for simultaneous diagonalization. *SIAM journal on matrix analysis and applications*, 17(1):161–164.

Chrétien, S., Dombry, C. et Faivre, A. (2016). A semi-definite programming approach to low dimensional embedding for unsupervised clustering. *arXiv preprint arXiv :1606.09190*.

Çivril, A. et Magdon-Ismail, M. (2009). On selecting a maximum volume submatrix of a matrix and related problems. *Theoretical Computer Science*, 410(47-49):4801–4811.

Combettes, P. L. et Pesquet, J.-C. (2011). Proximal splitting methods in signal processing. *In Fixed-point algorithms for inverse problems in science and engineering*, pages 185–212. Springer.

Cook, S. A. (1971). The complexity of theorem-proving procedures. *In Proceedings of the third annual ACM symposium on Theory of computing*, pages 151–158. ACM.

Craig, M. D. (1994). Minimum-volume transforms for remotely sensed data. *IEEE Transactions on Geoscience and Remote Sensing*, 32(3):542–552.

De Lathauwer, L., De Moor, B. et Vandewalle, J. (1996). Independent component analysis based on higher-order statistics only. *In Statistical Signal and Array Processing, 1996. Proceedings., 8th IEEE Signal Processing Workshop on (Cat. No. 96TB10004*, pages 356–359. IEEE.

De Lathauwer, L., De Moor, B. et Vandewalle, J. (2004). Computation of the canonical decomposition by means of a simultaneous generalized schur decomposition. *SIAM journal on Matrix Analysis and Applications*, 26(2):295–327.

Diaconis, P., Graham, R. et Kantor, W. M. (1983). The mathematics of perfect shuffles. *Advances in applied mathematics*, 4(2):175–196.

Ellis, J. A., Fan, H. et Shallit, J. (2002). The cycles of the multiway perfect shuffle permutation. *Discrete Mathematics & Theoretical Computer Science*, 5(1): 169–180.

Faivre, A. et Dombry, C. (2017). Total variation regularized non-negative matrix factorization for smooth hyperspectral unmixing. *arXiv preprint arXiv :1706.09242*.

Fiorini, S., Massar, S., Pokutta, S., Tiwary, H. R. et Wolf, R. D. (2015). Exponential lower bounds for polytopes in combinatorial optimization. *Journal of the ACM (JACM)*, 62(2):17.

Frommhold, L. (2006). *Collision-induced absorption in gases*, volume 2. Cambridge University Press.

Ghadimi, E., Teixeira, A., Shames, I. et Johansson, M. (2015). Optimal parameter selection for the alternating direction method of multipliers (admm) : quadratic problems. *IEEE Transactions on Automatic Control*, 60(3):644–658.

Gillis, N. et Vavasis, S. A. (2014). Fast and robust recursive algorithmsfor separable nonnegative matrix factorization. *IEEE transactions on pattern analysis and machine intelligence*, 36(4):698–714.

Goemans, M. X. et Williamson, D. P. (1995). Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *Journal of the ACM (JACM)*, 42(6):1115–1145.

Golub, G. H. et Van Loan, C. F. (2012). *Matrix computations*, volume 3. JHU Press.

Goto, K. et Geijn, R. A. (2008). Anatomy of high-performance matrix multiplication. *ACM Transactions on Mathematical Software (TOMS)*, 34(3):12.

Green, A. A., Berman, M., Switzer, P. et Craig, M. D. (1988). A transformation for ordering multispectral data in terms of image quality with implications for noise removal. *IEEE Transactions on geoscience and remote sensing*, 26(1):65–74.

Guédon, O. et Vershynin, R. (2016). Community detection in sparse networks via grothendieck's inequality. *Probability Theory and Related Fields*, 165(3-4):1025–1049.

Guennebaud, G., Jacob, B. *et al.* (2010). Eigen v3. http ://eigen.tuxfamily.org.

Halko, N., Martinsson, P.-G., Shkolnisky, Y. et Tygert, M. (2011). An algorithm for the principal component analysis of large data sets. *SIAM Journal on Scientific computing*, 33(5):2580–2594.

Hamerly, G. (2010). Making k-means even faster. *In SDM*, pages 130–140. SIAM.

Hartmann, J.-M., Boulet, C. et Robert, D. (2008). *Collisional effects on molecular spectra : laboratory experiments and models, consequences for applications*. Elsevier.

Heylen, R., Burazerovic, D. et Scheunders, P. (2011). Non-linear spectral unmixing by geodesic simplex volume maximization. *IEEE Journal of Selected Topics in Signal Processing*, 5(3):534–542.

Hoyer, P. O. (2004). Non-negative matrix factorization with sparseness constraints. *Journal of machine learning research*, 5(Nov):1457–1469.

Huang, K., Sidiropoulos, N. D. et Swami, A. (2014). Non-negative matrix factorization revisited : Uniqueness and algorithm for symmetric decomposition. *IEEE Transactions on Signal Processing*, 62(1):211–224.

Iordache, M.-D., Bioucas-Dias, J. M. et Plaza, A. (2012). Total variation spatial regularization for sparse hyperspectral unmixing. *Geoscience and Remote Sensing, IEEE Transactions on*, 50(11):4484–4502.

Iordache, M.-D., Plaza, A. et Bioucas-Dias, J. (2010). On the use of spectral libraries to perform sparse unmixing of hyperspectral data. *In Hyperspectral Image and Signal Processing : Evolution in Remote Sensing (WHISPERS), 2010 2nd Workshop on*, pages 1–4. IEEE.

Kaibel, V. (2011). Extended formulations in combinatorial optimization. *arXiv preprint arXiv :1104.1023*.

Keshava, N. et Mustard, J. F. (2002). Spectral unmixing. *IEEE signal processing magazine*, 19(1):44–57.

Kim, D., Sra, S. et Dhillon, I. S. (2007). Fast newton-type methods for the least squares nonnegative matrix approximation problem. *In Proceedings of the 2007 SIAM international conference on data mining*, pages 343–354. SIAM.

Kolda, T. G. et Bader, B. W. (2009). Tensor decompositions and applications. *SIAM review*, 51(3):455–500.

Krishnamurthy, V. et d'Aspremont, A. (2012). Convex algorithms for nonnegative matrix factorization. *arXiv preprint arXiv :1207.0318*.

Kuleshov, V., Chaganty, A. et Liang, P. (2015). Tensor factorization via matrix factorization. *In Artificial Intelligence and Statistics*, pages 507–516.

Kulis, B., Surendran, A. C. et Platt, J. C. (2007). Fast low-rank semidefinite programming for embedding and clustering. *In International Conference on Artificial Intelligence and Statistics*, pages 235–242.

Lee, D. D. et Seung, H. S. (2001). Algorithms for non-negative matrix factorization. *In Advances in neural information processing systems*, pages 556–562.

Li, J., Agathos, A., Zaharie, D., Bioucas-Dias, J. M., Plaza, A. et Li, X. (2015). Minimum volume simplex analysis : A fast algorithm for linear hyperspectral unmixing. *IEEE Transactions on Geoscience and Remote Sensing*, 53(9):5067–5082.

Lovász, L. (2003). Semidefinite programs and combinatorial optimization. *In Recent advances in algorithms and combinatorics*, pages 137–194. Springer.

Maehara, T. et Murota, K. (2011). Algorithm for error-controlled simultaneous block-diagonalization of matrices. *SIAM Journal on Matrix Analysis and Applications*, 32(2):605–620.

Marot, J. et Bourennane, S. (2008). Fast tensor signal filtering using fixed point algorithm. *In Acoustics, Speech and Signal Processing, 2008. ICASSP 2008. IEEE International Conference on*, pages 921–924. IEEE.

Moitra, A. (2016). An almost optimal algorithm for computing nonnegative rank. *SIAM Journal on Computing*, 45(1):156–173.

Mura, M. D., Villa, A., Benediktsson, J. A., Chanussot, J. et Bruzzone, L. (2011). Classification of hyperspectral images by using extended morphological attribute profiles and independent component analysis. *Geoscience and Remote Sensing Letters, IEEE*, 8(3):542–546.

Nascimento, J. M. et Dias, J. M. (2005a). Vertex component analysis : A fast algorithm to unmix hyperspectral data. *IEEE transactions on Geoscience and Remote Sensing*, 43(4):898–910.

Nascimento, J. M. et Dias, J. M. B. (2005b). Does independent component analysis play a role in unmixing hyperspectral data. Citeseer.

Netrapalli, P., Niranjan, U., Sanghavi, S., Anandkumar, A. et Jain, P. (2014). Non-convex robust pca. *In Advances in Neural Information Processing Systems*, pages 1107–1115.

Packer, A. (2002). Np-hardness of largest contained and smallest containing simplices for v-and h-polytopes. *Discrete and Computational Geometry*, 28(3):349–377.

Park, B., Windham, W., Lawrence, K. et Smith, D. (2007). Contaminant classification of poultry hyperspectral imagery using a spectral angle mapper algorithm. *Biosystems Engineering*, 96(3):323–333.

Parrilo, P. A. (2000). *Structured semidefinite programs and semialgebraic geometry methods in robustness and optimization*. Thèse de doctorat, California Institute of Technology.

Pataki, G. (1998). On the rank of extreme matrices in semidefinite programs and the multiplicity of optimal eigenvalues. *Mathematics of operations research*, 23(2):339–358.

Rudin, L. I., Osher, S. et Fatemi, E. (1992). Nonlinear total variation based noise removal algorithms. *Physica D : Nonlinear Phenomena*, 60(1):259–268.

Saad, Y. (2003). *Iterative methods for sparse linear systems*. SIAM.

Sato, H. (2014). Riemannian newton's method for joint diagonalization on the stiefel manifold with application to ica. *arXiv preprint arXiv :1403.8064*.

Tarabalka, Y., Chanussot, J., Benediktsson, J. A., Angulo, J. et Fauvel, M. (2008). Segmentation and classification of hyperspectral data using watershed. *In Geoscience and Remote Sensing Symposium, 2008. IGARSS 2008. IEEE International*, volume 3, pages III–652. IEEE.

Tarabalka, Y., Fauvel, M., Chanussot, J. et Benediktsson, J. A. (2010). Svm-and mrf-based method for accurate classification of hyperspectral images. *IEEE Geoscience and Remote Sensing Letters*, 7(4):736–740.

Theis, F. J., Cason, T. P. et Absil, P.-A. (2009). Soft dimension reduction for ica by joint diagonalization on the stiefel manifold. *In Independent Component Analysis and Signal Separation*, pages 354–361. Springer.

Thurau, C., Kersting, K., Wahabzada, M. et Bauckhage, C. (2012). Descriptive matrix factorization for sustainability adopting the principle of opposites. *Data Mining and Knowledge Discovery*, 24(2):325–354.

Van Zee, F. G. et van de Geijn, R. A. (2015). BLIS : A framework for rapidly instantiating BLAS functionality. *ACM Transactions on Mathematical Software*, 41(3):14 :1–14 :33.

Vavasis, S. A. (2009). On the complexity of nonnegative matrix factorization. *SIAM Journal on Optimization*, 20(3):1364–1377.

Velasco-Forero, S. et Angulo, J. (2013). Classification of hyperspectral images by tensor modeling and additive morphological decomposition. *Pattern Recognition*, 46(2):566–577.

Warren, R. et Osher, S. (2015). Hyperspectral unmixing by the alternating direction method of multipliers. *Inverse Problems and Imaging*, 14(3):00–00.

Weinberger, K. Q., Packer, B. et Saul, L. K. (2005). Nonlinear dimensionality reduction by semidefinite programming and kernel matrix factorization. *In AISTATS*.

Winter, M. E. (1999). N-findr : An algorithm for fast autonomous spectral endmember determination in hyperspectral data. *In SPIE's International Symposium on Optical Science, Engineering, and Instrumentation*, pages 266–275. International Society for Optics and Photonics.

Yannakakis, M. (1988). Expressing combinatorial optimization problems by linear programs. *In Proceedings of the twentieth annual ACM symposium on Theory of computing*, pages 223–228. ACM.

Yianilos, P. N. (1993). Data structures and algorithms for nearest neighbor search in general metric spaces. *In Proceedings of the Fifth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*.

Zhang, Y. (2010). An alternating direction algorithm for nonnegative matrix factorization. *Rice Technical Report*.

Liste des publications et communications

Publications

[1] Chrétien S., Dombry C. & Faivre A. (2017). A Semi-Definite Programming approach to low dimensional embedding for unsupervised clustering. *arXiv preprint*, (arXiv :1606.09190).

[2] Faivre A. & Dombry C. (2017). Total variation regularized non-negative matrix factorization for smooth hyperspectral unmixing. *arXiv preprint* (arXiv :1706.09242).

Communications

1er avril 2016 Relaxations de problèmes de classification.
Troisième Journée des Jeunes Chercheurs, Université de Bourgogne Franche-comté, Laboratoire de Mathématiques de Besançon, Besançon, France.

1er juin 2016 A SemiDefinite Programming approach to Gaussian Mixture based clustering.
48èmes Journées de Statistique, Université de Montpellier, Montpellier, France.

Analyse d'images hyperspectrales

Résumé

Les travaux de thèse effectués dans le cadre de la convention Cifre conclue entre le laboratoire de mathématiques de Besançon et Digital Surf, entreprise éditrice du logiciel d'analyse métrologique Mountains, portent sur les techniques d'analyse hyperspectrale. Sujet en plein essor, ces méthodes permettent d'exploiter des images issues de micro-spectroscopie, et en particulier de spectroscopie Raman. Digital Surf ambitionne aujourd'hui de concevoir des solutions logicielles adaptées aux images produites par ces appareils. Ces dernières se présentent sous forme de cubes de valeurs, où chaque pixel correspond à un spectre. La taille importante de ces données, appelées images hyperspectrales en raison du nombre important de mesures disponibles pour chaque spectre, obligent à repenser certains des algorithmes classiques d'analyse d'image.

Nous commençons par nous intéresser aux techniques de partitionnement de données. L'idée est de regrouper dans des classes homogènes les différents spectres correspondant à des matériaux similaires. La classification est une des techniques couramment utilisée en traitement des données. Cette tâche fait pourtant partie d'un ensemble de problèmes réputés trop complexes pour une résolution pratique : les problèmes NP-durs. L'efficacité des différentes heuristiques utilisées en pratique était jusqu'à récemment mal comprise. Nous proposons des argument théoriques permettant de donner des garanties de succès quand les groupes à séparer présentent certaines propriétés statistiques.

Nous abordons ensuite les techniques de dé-mélange. Cette fois, il ne s'agit plus de déterminer un ensemble de pixels semblables dans l'image, mais de proposer une interprétation de chaque pixel comme un mélange linéaire de différentes signatures spectrales, sensées émaner de matériaux purs. Cette déconstruction de spectres composites se traduit mathématiquement comme un problème de factorisation en matrices positives. Ce problème est NP-dur lui aussi. Nous envisageons donc certaines relaxations, malencontreusement peu convaincantes en pratique. Contrairement au problème de classification, il semble très difficile de donner de bonnes garanties théoriques sur la qualité des résultats proposés. Nous adoptons donc une approche plus pragmatique, et proposons de régulariser cette factorisation en imposant des contraintes sur la variation totale de chaque facteur.

Finalement, nous donnons un aperçu d'autres problèmes d'analyse hyperspectrale rencontrés lors de cette thèse, problèmes parmi lesquels figurent l'analyse en composantes indépendantes, la réduction non-linéaire de la dimension et la décomposition d'une image par rapport à une librairie regroupant un nombre important de spectres de référence.