Identification de mélanges gaussiens via des méthodes tensorielles

Vicnesh Venedittan encadré par Rima Khouja

Mai 2023

Abstract

Les modèles de mélange ont été introduits dans le cadre du clustering. Le clustering consiste à identifier des sous populations dans une population donnée. Par exemple, une entreprise peut chercher à identifier des catégories de clients afin de proposer des produits adaptés à chaque profil.

Lorsque chaque catégorie peut être modélisée via une distribution gaussiene multivariée, la population globale suit ce qu'on appelle *un modèle de mélange*. Celui ci est caractérisé par la moyenne et la variance de chaque sous population et la proportion de la population total appartenant à chaqu catégorie.

Identifier le modèle consiste à estimer chacune de ces caractéristiques. La méthode la plus connue pour résoudre ce problème est l'algorithme expectation-maximisation (EM). Une approche plus récente consiste à utiliser des méthodes de moments, qui utilisent l'analyse tensorielle.

Le but de ce stage est donc d'utiliser les techniques de décomposition tensorielle pour ce problème d'identification de mélange gaussien. Après une introduction à des notions bases sur ce sujet : mélange gaussien, moments d'une variable aléatoire, tenseur, décomposition tensorielle. On présente l'analyse d'un article récent autour de ce sujet [PKK22]. Une partie de ce rapport sera enfin consacrée à l'implémentation (sur le logiciel Python) des algorithmes présentés dans ce papier et à leur validation expérimentale.

Introduction

Un modèle de mélange gaussien (Gaussian Mixture Model en anglais, abrégé GMM) est un modèle statistique utilisé pour modéliser des données qui peuvent être considérées comme provenant d'un mélange de distributions gaussiennes.

L'idée est qu'un modèle de mélange gaussien permet de décrire une population sous forme d'un mélange de plusieurs sous-populations. Chaque sous-population est décrite par une distribution gaussienne avec sa propre moyenne et sa propre matrice de covariance.

Le modèle de mélange gaussien est largement utilisé dans le domaine de l'apprentissage automatique pour résoudre de nombreux problèmes, tels que la segmentation d'image, la classification, la reconnaissance de la parole, la modélisation de la densité de probabilité, la compression de données et bien d'autres.

On a alors pour but d'estimer les paramètres de chacun des différentes sous-populations caractériser par sa moyenne et sa matrice de covariance. Parmis plusieurs méthodes la plus connue et répandue reste l'algorithme espérance-maximisation (expectation-maximization algorithm en anglais, souvent abrégé EM) l'algorithme EM est une méthode d'optimisation avec une approche itérative qui alterne deux étapes : l'étape dite (E) expectation et de (M) maximization.

On considère $X = (x_1, x_2, ..., x_p)$ données observées de loi $\mathbb{P}_X$ admettant pour loi de densité $f(X, \theta)$ de paramètre θ (qui peuvent être l'espérance, la variance etc). On considère la vraisemblance : $\mathcal{L}(X, \theta) = \sum_{i=1}^{p} log[f(x_i, \theta)]$.

L'étape (E) consiste à évaluer l'espérance de la vraisemblance, elle utilise les valeurs courantes des paramètres du modèle et les données observées pour estimer la distribution des variables latentes manquantes.

Dans cette étape, on utilise les résultats obtenus à l'étape (E) pour déterminer les estimations des paramètres du modèle qui maximisent la vraisemblance. En d'autres termes, on ajuste les paramètres du modèle de manière à ce que la probabilité d'observer les données soit maximisée.

Une autre manière d'estimer les paramètres est la méthode des moments.

C'est une méthode qui utilise les moments empiriques calculés à partir des données observées et les identifie avec les moments théoriques qui sont en termes des variables latentes manquantes. Cela permet d'obtenir une estimation de la distribution qui correspond aux données observées.

En statistique, les données peuvent êtres multidimensionnelles comme les images par exemples qui peuvent avoir trois dimensions (hauteur, largeur, profondeur). Une manière de les représenter est d'utiliser les tenseurs permettant de généraliser la notion de vecteur ou de matrice. Ainsi, une matrice est un tenseur de dimension deux, un vecteur est un tenseur de dimension un. Un avantage des tenseurs est qu'ils peuvent avoir des décompositions uniques dans des conditions beaucoup plus permissives que dans le cas des décompositions matricielles, où la propriété d'unicité, si importante dans un problème d'identification, est très difficile à obtenir.

En particulier, le moment d'ordre d est un tenseur symétrique d'ordre d, ce qui explique l'utilisation des tenseurs dans la méthode des moments.

L'utilisation d'une approche tensorielle pour l'identification d'un modèle de mélange gaussien présente plusieurs avantages. Tout d'abord, elle permet de prendre en compte l'interaction entre les variables, ce qui peut conduire à une meilleure représentation des données et à une meilleure performance du modèle. Ensuite, elle permet de traiter des données de grande dimension. Enfin, l'approche tensorielle peut faciliter l'interprétation des résultats en identifiant des groupes de variables qui ont une corrélation et qui contribuent de manière significative à la variation des données.

Plusieurs travaux ont considéré la méthode des moments avec l'approche tensorielle pour traiter le problème GMM, tels que [HK13, KMM22]. Le plus récent est le papier intitulé "Tensor moments of Gaussian mixture models: theory and applications" [PKK22], qui cherche à minimiser l'erreur entre les moments empiriques et théoriques, en élaborant une computation très efficace pour les gradients de la fonction de coût. Le calcul des moments, qui représente la partie de calcul la plus coûteuse dans une méthode des moments, est réalisé de manière implicite en introduisant la notion de symétrisation tensorielle, c'est-à-dire la transformation d'un tenseur en un tenseur symétrique, ainsi que l'utilisation des polynômes de Bell.

Dans un premier temps, nous allons introduire les notions de tenseurs et

de mélanges gaussiens. Dans un second temps, nous résumerons l'article en présentant les principaux résultats. Nous fournirons également une implémentation en Python de quelques algorithmes présentés dans le papier, ainsi que quelques exemples pour leur validation expérimentale.

Définition 1 (Loi gaussienne multivariée)**.** *Soit $X \in \mathbb{R}^n$ un vecteur aléatoire, X suit une distribution gaussienne multivariée de paramètre $\mu \in \mathbb{R}^n$ et $\Sigma \in \mathbb{R}^{n \times n}$ définie positive, s'il admet pour densité :*

$$f(X, \mu, \Sigma) = \frac{1}{\sqrt{(2\pi)^n |\Sigma|}} exp(-\frac{1}{2}(X-\mu)^t \Sigma^{-1}(X-\mu))$$

Définition 2 (Modèle de mélange Gaussien)**.** *Soit $\lambda = (\lambda_1, ..., \lambda_m)$ tel que $\sum_{i=1}^m \lambda_i = 1$, puis $\mu = (\mu_1, ..., \mu_m)$ et $\Sigma = (\Sigma_1, ..., \Sigma_m)$. X suit une loi de mélange gaussien à m composante de dimension p si elle admet pour densité :*

$$f(X, \mu, \Sigma) = \sum_{i=1}^{m} \lambda_i f_i(X, \mu_i, \Sigma_i).$$

Où pour tout $i \in 1, ..., m$, f_i suit une loi gaussienne multivariée de paramètre (μ_i, Σ_i).

Exemple 1

Pour $\lambda = (0.3, 0.3, 0.4)$, $\mu = \left(\begin{pmatrix}0\\0\end{pmatrix} \begin{pmatrix}3\\3\end{pmatrix} \begin{pmatrix}-3\\3\end{pmatrix}\right)$ et $\Sigma = (diag(1.5), diag(1.5), diag(2))$.

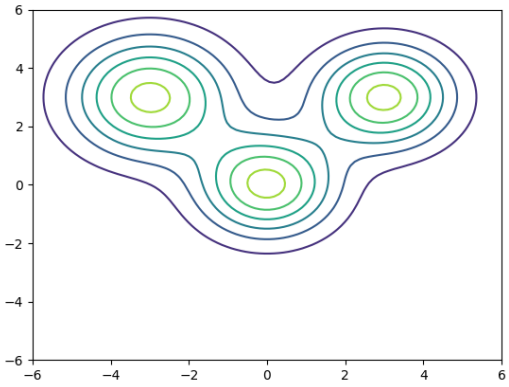


Figure 1: Les trois mélanges gaussiens de Exemple .

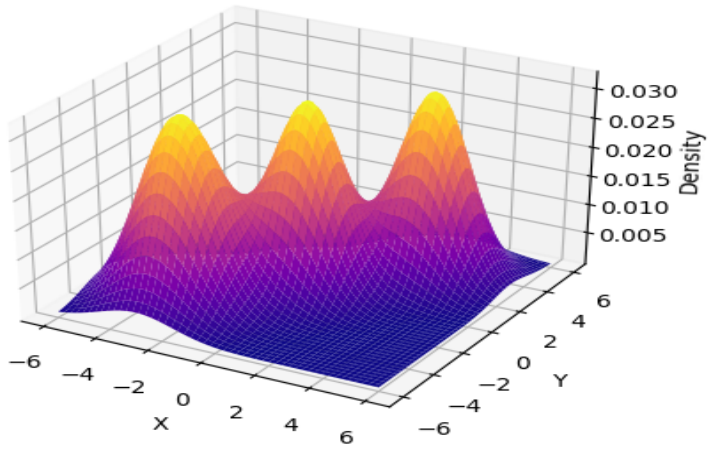


Figure 2: La fonction densité de l'Exemple .

Définition 3 (Tenseur)**.** *Soit* $\mathbb{R}^{n_1} \otimes \ldots \otimes \mathbb{R}^{n_d}$ *l'espace vectoriel du produit tensoriel de* $\mathbb{R}^{n_1}, \ldots, \mathbb{R}^{n_d}$*. L'espace vectoriel* $\mathbb{R}^{n_1} \otimes \ldots \otimes \mathbb{R}^{n_d}$ *est défini par la propriété universelle [Bou98] telle que toute application multilinéaire* f *sur* $\mathbb{R}^{n_1} \times \ldots \times \mathbb{R}^{n_d}$ *s'élève à une application linéaire sur* $\mathbb{R}^{n_1} \otimes \ldots \otimes \mathbb{R}^{n_d}$*. Un élément*

$\mathcal{T}$ de l'espace vectoriel des tableaux à d dimensions, noté $\mathbb{R}^{n_1 \times ... \times n_d}$, peut être écrit sous la forme d'un tableau $\mathcal{T} = [t_{i_1,...,i_d}]_{1 \leq i_1 \leq n_1,...,1 \leq i_d \leq n_d}$, où $t_{i_1,...,i_d} \in \mathbb{R}$ est l'élément $(i_1, \ldots, i_d)$ du tableau. Un tenseur peut être représenté par un tableau multidimensionnel dans $\mathbb{R}^{n_1 \times ... \times n_d}$, par rapport à une base fixe sur $\mathbb{R}^{n_1}, \ldots, \mathbb{R}^{n_d}$. Dans ce qui suit, on considère la base canonique sur $\mathbb{R}^{n_1}, \ldots, \mathbb{R}^{n_d}$.

Notation: On note $\mathcal{T}^d_{(n_1,...,n_d)} = \mathbb{R}^{n_1 \times ... \times n_d}$ l'ensemble des tenseurs réels d'ordre d de dimension $(n_1, ..., n_d)$. Si $n_1 = ... = n_d = n$ on le notera $\mathcal{T}^d_n$.

Définition 4 (Produit extérieur)**.** *Soit $(\mathcal{A}, \mathcal{B}) \in \mathcal{T}^{d_1}_{(n_1,...,n_{d_1})} \times \mathcal{T}^{d_2}_{(m_1,...,m_{d_2})}$, le produit tensoriel noté $\otimes$ de $\mathcal{A}$ et $\mathcal{B}$ est un tenseur de $\mathcal{T}^{d_1+d_2}_{(n_1,...,n_{d_1},m_1,...,m_{d_2})}$ tel que :*

$$(\mathcal{A} \otimes \mathcal{B})(i_1, ..., i_{d_1}, j_1, ..., j_{d_2}) = a_{i_1,...,i_{d_1}} b_{j_1,...,j_{d_2}},$$

$\forall 1 \leq i_k \leq n_k$, $\forall 1 \leq j_l \leq m_l$, *tel que* $k \in \{1, \ldots, d_1\}$, $l \in \{1, \ldots, d_2\}$.

Pour tout $a \in \mathbb{R}^n$, on note $\underbrace{a \otimes a ... \otimes a}_{\times d}$ par $a^{\otimes d}$.

Exemple 2

$$a = \begin{bmatrix} 1 \\ 2 \end{bmatrix} \; b = \begin{bmatrix} 3 \\ 5 \end{bmatrix}, a \otimes b = \begin{bmatrix} 3 & 5 \\ 6 & 10 \end{bmatrix}$$

Définition 5 (Produit matriciel de Hadamard)**.** *Soit $A, B \in \mathbb{R}^{n \times m}$, le produit de Hadamar noté $*$ de A et de B est :*

$$A * B = \begin{pmatrix} a_{1,1}b_{1,1} & a_{1,2}b_{1,2} & ... & a_{1,m}b_{1,m} \\ a_{2,1}b_{2,1} & a_{2,2}b_{2,2} & & a_{2,m}b_{2,m} \\ ... & ... & ... & ... \\ a_{n,1}b_{n,1} & a_{n,2}b_{n,2} & ... & a_{n,m}b_{n,m} \end{pmatrix}$$

Définition 6 (Produit scalaire tensoriel)**.** *Soit $\mathcal{A}, \mathcal{B} \in \mathcal{T}^d_n$, on note le produit scalaire de $\mathcal{A}$ et $\mathcal{B}$ par :*

$$\langle \mathcal{A}, \mathcal{B} \rangle = \sum_{i_1=1}^{n} ... \sum_{i_d=1}^{n} a_{i_1,...,i_d} b_{i_1,...,i_d}.$$

En particulier, on a :

$$\langle \mathcal{A}, \mathcal{A} \rangle = ||\mathcal{A}||^2 = \sum_{i_1=1}^{n} ... \sum_{i_d=1}^{n} a^2_{i_1,...,i_d}.$$

Remarque 1

Pour tout $a, b \in \mathbb{R}^n$, $\langle a^{\otimes d}, b^{\otimes d} \rangle = \sum_{i_1=1}^{n} ... \sum_{i_d=1}^{n} a_{i_1} a_{i_2} ... a_{i_d} b_{i_1} ... b_{i_d} = \langle a, b \rangle^d$

Définition 7 (Tenseur symétrique)**.** *Soit $\mathcal{A} \in \mathcal{T}_n^d$. Pour tout $(j_1, j_2, ..., j_d) \in \{1, ..., n\}^d$, on dit que $\mathcal{A}$ est un tenseur symétrique si ses entrées $a_{j_1,j_2,...,j_d}$ ne changent pas sous n'importe quelle permutation de ces d indices, c'est-à-dire si :*

$$a_{j_1,j_2,...,j_p} = a_{j_{\sigma(1)},...,j_{\sigma(d)}},$$

pour tout σ appartenant à $\mathfrak{S}^d$, le groupe symétrique des permutations sur $\{1, \ldots, d\}$.

On note l'ensemble des tenseurs symétriques d'ordre d de dimension n par $\mathcal{S}_n^d$.

Résumé de l'article *Tensor moments of Gaussian mixture models: Theory and applications [PKK22]*

L'article utilise la méthode des moments pour traiter le problème d'identification des mélanges gaussiens. Il vise à identifier ce mélange en trouvant la correspondance entre les moments théoriques et les moments empiriques. Plus précisément, il résout un problème d'optimisation où la fonction de coût cherche à minimiser l'erreur entre les moments théoriques d'ordre d ($\tilde{\mathcal{M}}^{(d)}$) et les moments empiriques d'ordre d ($\mathcal{M}^{(d)}$) :

$$\frac{1}{2} \min \|\mathcal{M}^{(d)} - \tilde{\mathcal{M}}^{(d)}\|^2.$$

Pour cela, une méthode de calcul efficace des gradients de la fonction de coût a été développée. Les auteurs ont introduit la notion de symétrisation, qui transforme un tenseur multilinéaire en un tenseur symétrique, et ont utilisé cette notion pour développer une expression explicite et concise des moments d'ordre d. Ensuite, la fonction de coût a été décomposée en deux parties, et en

utilisant la nouvelle expression du moment d'ordre d ainsi que les polynômes de Bell, des expressions explicites et concises des gradients de ces fonctions ont été présentées. Le résultat prometteur de cette étude est qu'il est possible d'utiliser la méthode des moments pour le problème d'identification des mélanges gaussiens sans avoir besoin de calculer explicitement les moments. Cette partie du calcul, qui présente une complexité élevée, était considérée comme un point négatif de la méthode des moments par rapport à d'autres méthodes plus rapides, telles que l'algorithme EM, par exemple.

Dans la suite, nous présentons certains résultats de cet article. De plus, nous introduisons une implémentation en Python du code des algorithmes 1 et 2 décrits dans cet article. Afin de valider ce code, nous proposons un exemple de mélange gaussien où les paramètres latents ont été perturbés. Ce point perturbé est utilisé comme point initial pour des itérations de descente de gradient. Enfin, nous présentons des graphiques décroissants qui prouvent que l'erreur de la fonction de coût diminue.

Dans la définition suivante, nous introduisons l'opération Sym qui permet de transformer un tenseur qui n'est pas forcément symétrique en un tenseur symétrique.

Définition 8 (Opération Sym)**.** *Pour tout* $\mathcal{A} \in \mathcal{T}_n^d$, $(j_1, j_2, ..., j_d) \in \{1, ..., n\}^d$*:*

$$Sym[\mathcal{A}](j_1, j_2, ..., j_d) = \frac{1}{d!} \sum_{\sigma \in \mathfrak{S}^d} a_{j_{\sigma(1)}, ..., j_{\sigma(d)}}.$$

Proposition 1: [Pereira, Kileel, Kolda, 2022]

On définit $\Phi : \bigcup_{d=0}^{\infty} \mathcal{T}_n^d \to \bigcup_{d=0}^{\infty} \mathbb{R}[z_1, ..., z_n]_d$ (l'ensemble des polynômes homogènes réels) tel que pour tout $\mathcal{V} \in \mathcal{T}_n^d$ et $z \in \mathbb{R}^n$:

$$\Phi[\mathcal{V}](z) = \langle \mathcal{V}, z^{\otimes d} \rangle.$$

- L'application Φ est bijective pour la restriction sur $\mathcal{S}_n^d$. Pour tout $\mathcal{A} \in \mathcal{T}_n^d$:

  $$\Phi[\mathcal{A}] = \Phi[Sym(\mathcal{V})].$$

  En particuliers si $\mathcal{A}, \mathcal{B} \in \mathcal{S}_n^d$, et $\Phi[\mathcal{A}] = \Phi[\mathcal{B}]$ alors $\mathcal{A} = Sym[\mathcal{B}]$.

- Pour tout vecteur $v \in \mathbb{R}^n$ et pour toutes matrices $M \in \mathbb{R}^{n \times n}$, les polynômes homogènes $\Phi[v], \Phi[M]$ s'écrivent :

  $$\Phi[v](z) = v^T z \text{ et } \Phi[M](z) = z^T M z.$$

- Pour tout $\mathcal{A} \in \mathcal{T}_n^{d_1}, \mathcal{B} \in \mathcal{T}_n^{d_2}$, on a :

  $$\Phi[\mathcal{A} \otimes \mathcal{B}] = \Phi[\mathcal{A}]\Phi[\mathcal{B}].$$

Ainsi, un moment d'ordre d pour une distribution gaussiénne multivariée peut s'écrire en utilisant l'opération Sym comme le suivant:

Théorème 1: [Pereira, Kileel, Kolda, 2022]

Soit $X \sim \mathcal{N}(\mu, \Sigma)$, le moment d'ordre d est $\mathcal{M}^{(d)} = \mathbb{E}(X^{\otimes d})$ est :

$$\mathcal{M}^{(d)} = \sum_{k=0}^{\lfloor d/2 \rfloor} C_{d,k} Sym(\mu^{\otimes d-2k} \otimes \Sigma^{\otimes k}),$$

avec : $C_{d,k} = \binom{d}{2k} \frac{2k!}{k! 2^k}$.

Essayons de calculer $\mathcal{M}^{(d)}$ pour $d = 1, 2, 3$:

$$\mathcal{M}^{(1)} = \mu$$
$$\mathcal{M}^{(2)} = \mu^{\otimes 2} + \Sigma$$

$$\mathcal{M}^{(3)} = C_{3,0}Sym(\mu^{\otimes 3} \otimes \Sigma^{\otimes 0}) + C_{3,1}Sym(\mu^{\otimes 3-2} \otimes \Sigma^{\otimes 1})$$
$$= \mu^{\otimes 3} + 3Sym(\mu \otimes \Sigma).$$

Après les moments suivant sont plus difficle à calculer.

Proof. L'idée de la preuve est d'utiliser la Proposition 1 et un résultat de [Gut09], qui donne une expression pour les moments dans le cas unidimensionnel :

Proposition 2: [Gut09, Théorème 3.1]

If $Y \sim \mathcal{N}(\mu, \sigma)$ then

$$\mathbb{E}[Y^d] = \sum_{k=1}^{\lfloor d/2 \rfloor} \binom{d}{2k} \frac{2k!}{k!2^k} \mu^{d-2k} \sigma^k.$$

On utilise la Propostion 1 et on se ramène au cas multivarié :

$$\Phi[\mathbb{E}(X^{\otimes d})](z) = \langle z^{\otimes d}, \mathbb{E}(X^d) \rangle = \mathbb{E}(\langle z^{\otimes d}, X^{\otimes d} \rangle) = \mathbb{E}((z^t X)^d).$$

On a $z^t X \sim \mathcal{N}(z^t \mu, z^t \Sigma z)$ par propriété des vecteurs gaussiens. On applique alors la Proposition 1:

$$\Phi[\mathbb{E}(X^{\otimes d})](z) = \mathbb{E}((z^t X)^d) = \sum_{k=0}^{\lfloor d/2 \rfloor} \binom{d}{2k} \frac{2k!}{k!2^k} (z^t \mu)^{d-2k} (z^t \Sigma z)^k =$$
$$\sum_{k=0}^{\lfloor d/2 \rfloor} \binom{d}{2k} \frac{2k!}{k!2^k} \Phi[\mu^{\otimes d-2k} \otimes \Sigma^{\otimes k}](z).$$

□

Ensuite, on peut déduire qu'un moment d'ordre d d'un mélange de gaussiens peut s'écrire comme le suivant:

Théorème 2

Si $X \sim \sum_{j=1}^{m} \lambda_j \mathcal{N}(\mu_j, \Sigma_j)$, le moment d'ordre d est $\mathcal{M}^{(d)} = \mathbb{E}(X^{\otimes d})$ est :

$$\mathcal{M}^{(d)} = \sum_{j=1}^{m} \sum_{k=0}^{\lfloor d/2 \rfloor} \lambda_j C_{d,k} Sym(\mu^{\otimes d-2k} \otimes \Sigma^{\otimes k}),$$

avec : $C_{d,k} = \binom{d}{2k} \frac{2k!}{k!2^k}$.

Soit $x_1, \ldots, x_p$ un ensemble de points à n variables suivant un modèle de m mélange gaussien $\mathcal{N}(\lambda, \mu, \Sigma)$, tel que $\lambda = (\lambda_1, \ldots, \lambda_m)$, les λ_i sont les proportions de chaque distribution gaussienne dans le mélange ($\sum_{i=1}^{m} \lambda_i = 1$), $\mu := [\mu_1, \ldots, \mu_m]$ et $\Sigma := [\Sigma_1, \ldots, \Sigma_m]$ tel que (μ_i, Σ_i) est la moyenne et la covariance du ième mélange pour $i \in \{1, \ldots, m\}$. On note par $\theta := (\lambda, \mu, \Sigma)$.

Pour trouver le paramètre latent θ le papier propose de résoudre un problème d'optimisation qui minimise l'erreur entre le moment théorique d'ordre d $\tilde{\mathcal{M}}^{(d)}$ et le moment empirique d'ordre d $\mathcal{M}^{(d)}$:

$$\min \frac{1}{2} \|\mathcal{M}^{(d)} - \tilde{\mathcal{M}}^{(d)}\|^2 := \min F. \quad (1)$$

Tel que le moment théorique $\tilde{\mathcal{M}}^{(d)}$ est donnée par Théorème 2 et le moment empirique d'ordre d est donné par $\mathcal{M}^{(d)} = \frac{1}{p} \sum_{k=1}^{p} x_k^{\otimes d}$. En appliquant la propriété $\|a-b\|^2 = \|a\|^2 + \|b\|^2 - 2\langle a, b \rangle$, on décompose F en deux fonctions F_1 et F_2 tel que

$$F = F_1 - \frac{1}{p} F_2,$$

avec $F_1 = ||\tilde{\mathcal{M}}^{(d)}||^2$ et de $F_2 = \frac{1}{2} \sum_{i=1}^{p} \langle \tilde{\mathcal{M}}^{(d)}, x_i^{\otimes d} \rangle$.

Le papier présente un calcul concis des gradients des fonctions F_1 et F_2 par rapport à θ pour le cas de mélange de gaussiens sphériques (c'est-à-dire $\Sigma_i = \sigma_i^2 I$). Ces gradients calculés constituent l'élément principal d'une méthode d'optimisation du premier ordre qui peut être utilisée pour résoudre le problème de minimisation (1). Ainsi, dans ce qui suit, nous présentons le code que nous avons implémenté en Python pour les gradients de F_1 et F_2 (Algorithmes 1 et 2 dans le papier).

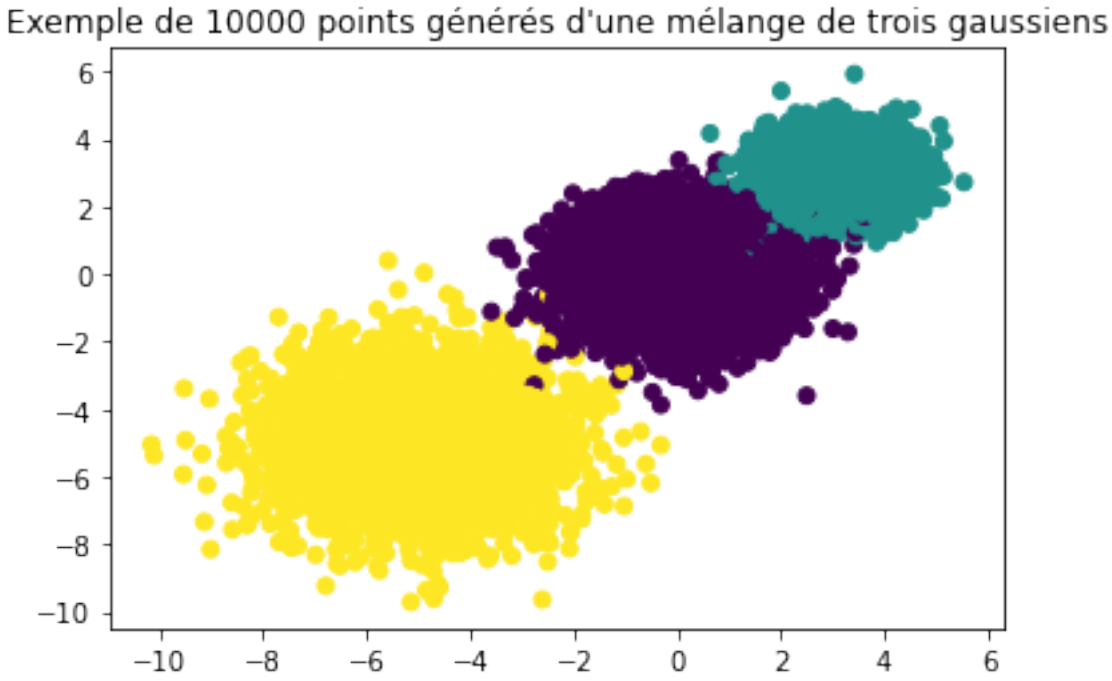
```
import numpy as np
from math import comb
import matplotlib.pyplot as plt
import numpy.linalg as LA

# one_zero(k) renvoie 1 si k>1 sinon renvoie 0

def one_zero(k):
    if k>1:
        return 1
    else:
        return 0

# two_zero(k) renvoie 1 si k>2 sinon renvoie 0

def two_zero(k):
    if k>2:
        return 1
    else:
        return 0


def ph(A, k):
  #entrée tenseur et un entier
  #Retour la produit d'hadamard de A k fois
    n = A.shape[0]
    m = A.shape[1]
    res = A.copy()
    for i in range(k-1):
        res = np.multiply(res, A)

    return res

def F1(theta, d):
    lamb, A, D = theta
    m = A.shape[1]
    Bs = [np.ones((m,m))]
    Ks = []
```

```
for k in range(1, d+1):
    Kk = np.zeros((m, m))
    km1_fact = np.math.factorial(k-1)
    if k%2 == 1:
        Kk = np.math.factorial(k)*(np.transpose(np.multiply(ph(D,k-1), A)))
        @(np.multiply(ph(D,k-1), A))
    else:
        V = (np.transpose(ph(D,k)))@(np.multiply(ph(D,k-2), ph(A,2)))
        Kk = np.math.factorial(k-1)*(np.transpose(ph(D,k))@ph(D,k)) +
        (np.math.factorial(k)/2)*(V+np.transpose(V))
    Ks.append(Kk)
    Bk = np.zeros((m,m))
    for r in range(0, k):
        Bk = Bk + comb(k-1, r)*(np.multiply(Bs[r], Ks[k-r-1]))
    Bs.append(Bk)
f = lamb@(Bs[d])@np.transpose(lamb)
W_lamb = 2*lamb@Bs[d]
W_A = 0
W_D = 0
for k in range(1, d+1):
    B = np.multiply(Bs[d-k], np.transpose(lamb)@lamb)
    k_fact = np.math.factorial(k)
    if k%2 == 1:
        T_D = np.math.factorial(k)*np.multiply(ph(D,k-1), A)
        U_D = one_zero(k)*(k-1)*np.multiply(ph(D,k-2), A)
        W_D = W_D + 2*comb(d, k)*np.multiply(T_D@B, U_D)
        T_A = np.math.factorial(k)*np.multiply(ph(D,k-1), A)
        U_A = ph(D,k-1)
        W_A = W_A + 2*comb(d, k)*np.multiply(T_A@B, U_A)
    else:
        T_D1 = np.math.factorial(k)*ph(D,k) +
        k*(np.math.factorial(k)/2)*(np.multiply(ph(D,k-2), ph(A,2)))
        U_D1 = ph(D,k-1)
        T_D2 = (np.math.factorial(k)/2)*ph(D,k)
        U_D2 = two_zero(k)*(k-2)*(np.multiply(ph(D,k-3), ph(A,2)))
        W_D = W_D + 2*comb(d,k)*(np.multiply(T_D1@B, U_D1)
        + np.multiply(T_D2@B, U_D2))
        T_A = np.math.factorial(k)*ph(D,k)
```

```
            U_A = np.multiply(ph(D,k-2), A)
            W_A = W_A + 2*comb(d, k)*(np.multiply(T_A@B, U_A))
    return (f,W_A,W_D,W_lamb)



def F2(X,theta,d):
    lamb, A, D = theta
    p = X.shape[1]
    m = lamb.shape[1] #la matrice lamb est en ligne
    n = A.shape[0]
    V = np.transpose(X)@A
    Z = ph((np.transpose(X)),2)@ph(D,2)
    if d == 1:
        R2 = np.zeros((p,m))
        R1 = np.ones((p,m))
    else:
        R2 = np.ones((p,m))
        R1 = V
    R2_copy = R2.copy()
    R1_copy = R1.copy()
    for k in range(2,d):
        R3 = R2_copy
        R2 = R1_copy
        R1 = np.multiply(R2,V)+(k-1)*np.multiply(R3,Z)
        R3_copy = R3.copy()
        R2_copy = R2.copy()
        R1_copy = R1.copy()
    T = X@R1
    W_A = d*T@np.diag(lamb[0,:])
    U = (d-1)*np.multiply((ph(X,2)@R2),D)
    W_D = d*U@np.diag(lamb[0,:])
    W_lamb = (np.multiply(T,A)+np.multiply(U,D))[0]
    f = W_lamb @ np.transpose(lamb)

    return (f,W_A,W_D,W_lamb)
```

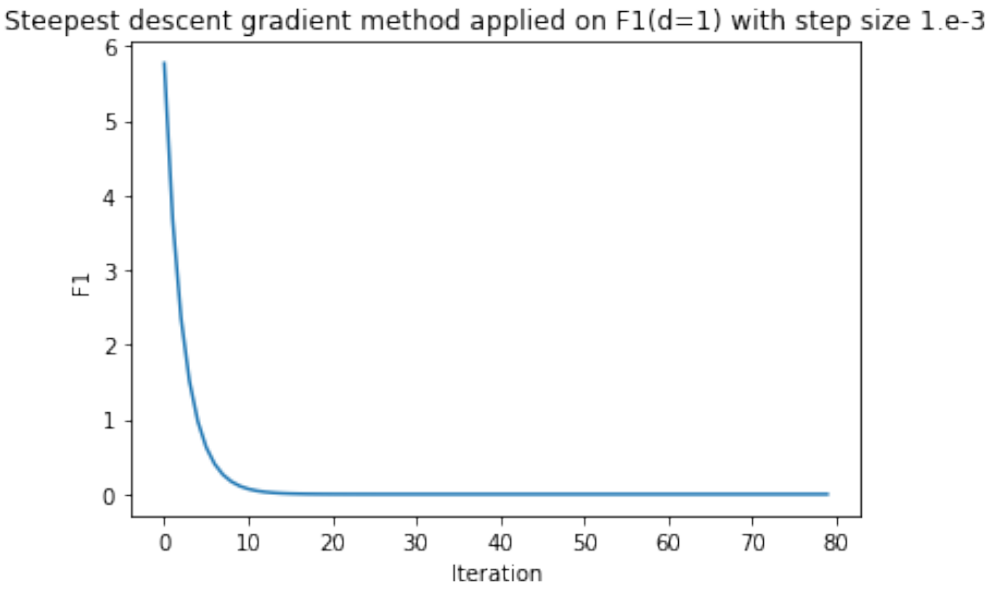
Pour valider le code on présente un exemple de mélange de trois gaussiens dont les paramètres sont: $\lambda_1 = 0.5,\ \lambda_1 = 0.2,\ \lambda_1 = 0.3,\ \mu_1 = [0\ 0],\ \mu_2 = [3\ 3],\ \mu_3 = [-5\ -5],\ \ \Sigma_1 = diag(1,1),\ \Sigma_2 = diag(0.5,0.5),\ \Sigma_3 = diag(2,2)$.

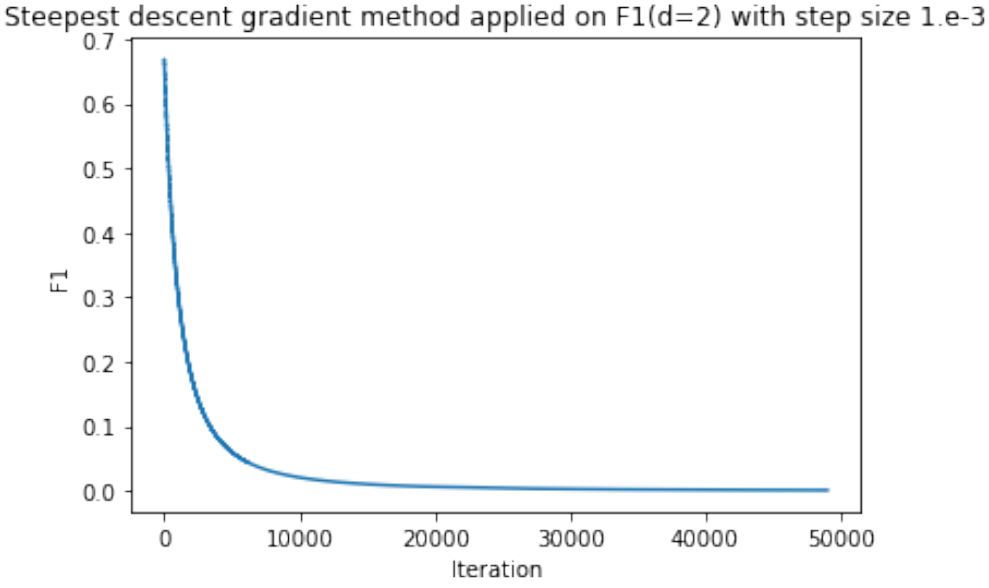
On génère $p = 10000$ points suivants ce mélange, voici une illustration:

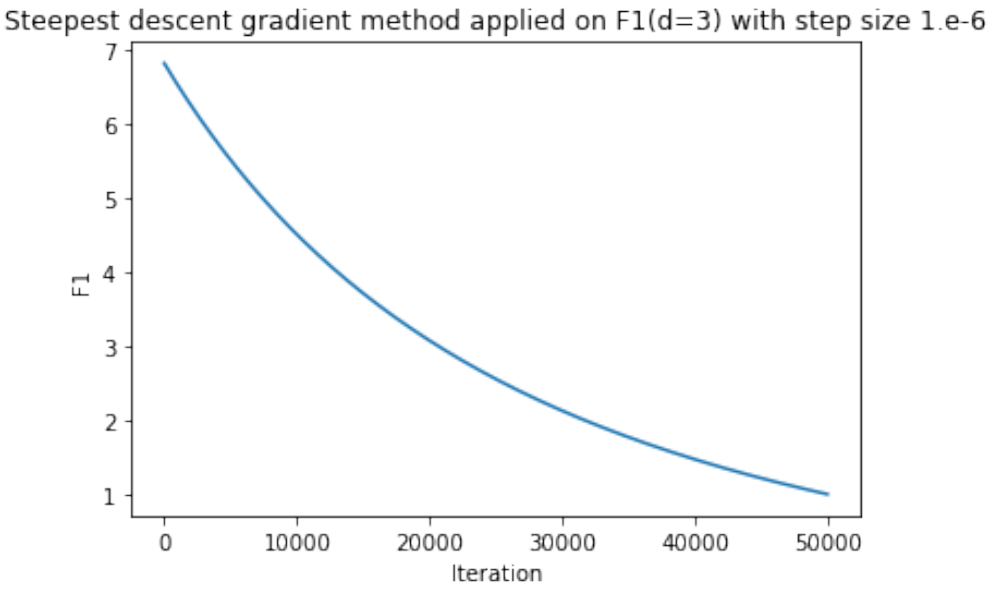


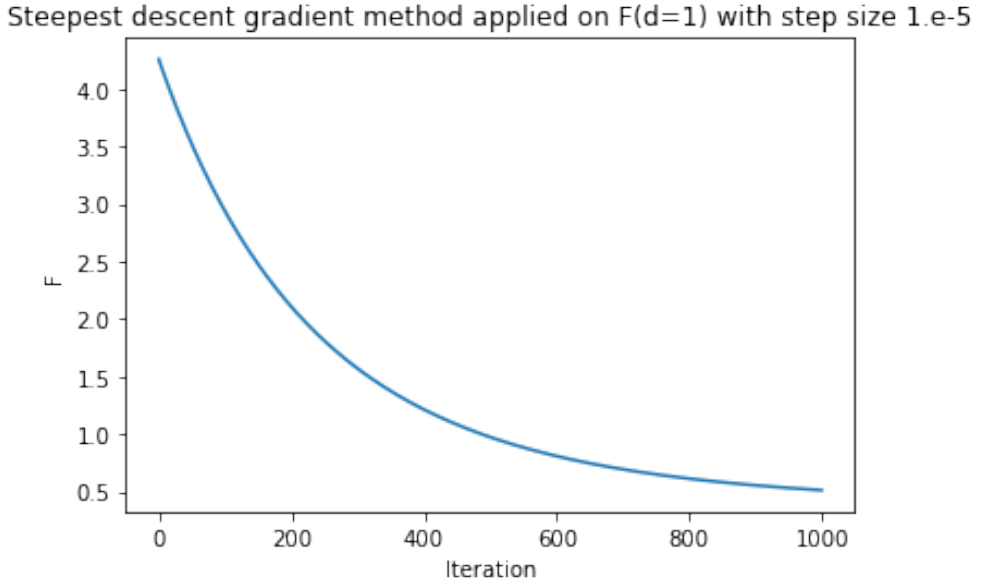
On ajoute au point θ une perturbation d'ordre 1×10^{-3} et on effectue des itérations de descente de gradient. Pour rappel, soit f une fonction à minimiser, la méthode de descente de gradient à partir d'un point initial x_0 est définie par l'itération $k+1$: $x_{k+1} = x_k - \alpha \nabla_{x_k} f$, où $\nabla_{x_k} f$ représente le gradient de f au point x_k et α est un scalaire positif appelé "taille de pas" (step size en anglais).

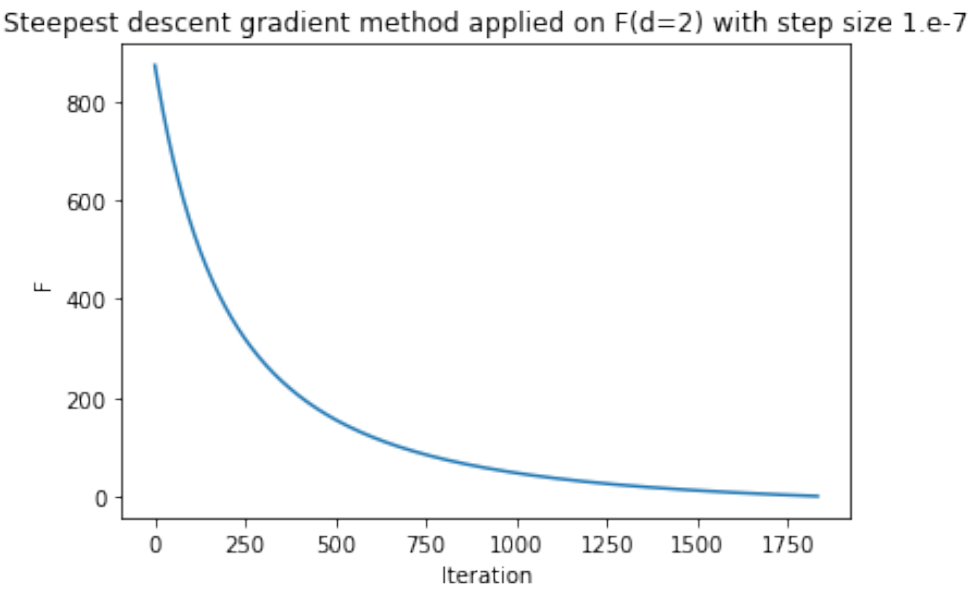
Dans ce qui suit, nous présentons quelques graphiques illustrant la décroissance de l'erreur de F_1 et F en fonction des itérations de la méthode de descente de gradient.

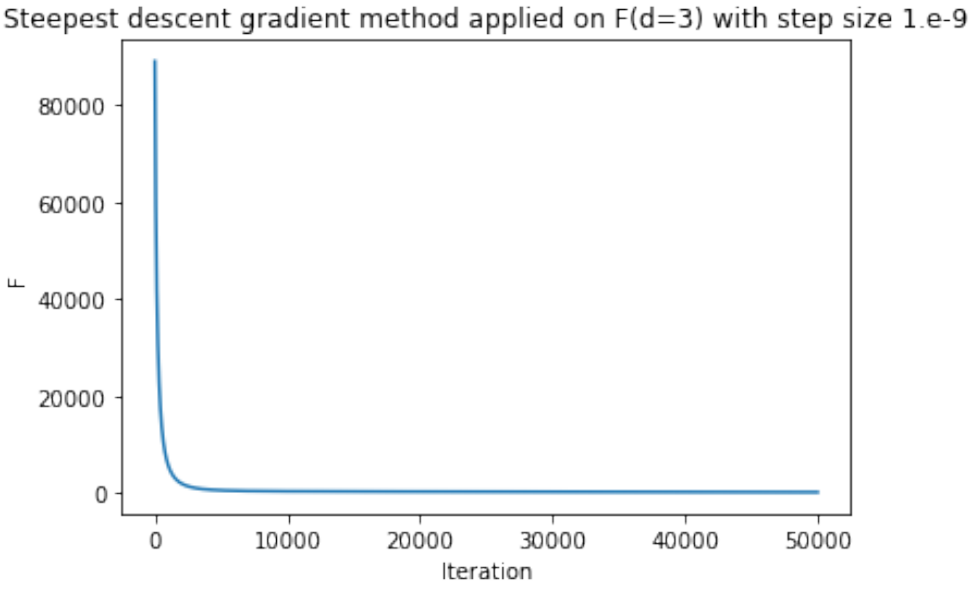

Steepest descent gradient method applied on F1(d=1) with step size 1.e-3
F1
Iteration
0
1
2
3
4
5
6
0
10
20
30
40
50
60
70
80


Steepest descent gradient method applied on F1(d=2) with step size 1.e-3
F1
Iteration
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0
10000
20000
30000
40000
50000


Steepest descent gradient method applied on F1(d=3) with step size 1.e-6
F1
Iteration
7
6
5
4
3
2
1
0
10000
20000
30000
40000
50000


Steepest descent gradient method applied on F(d=1) with step size 1.e-5
F
Iteration
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0
200
400
600
800
1000





Conclusion

Pour conclure, nous avons abordé le problème de l'identification des mélanges gaussiens en utilisant la méthode des moments, qui implique l'utilisation de

tenseurs d'ordre supérieur. Nous avons analysé un article récent sur ce sujet [PKK22], dans lequel l'idée principale était de résoudre ce problème en rapprochant le moment théorique du moment empirique du modèle, tout en développant un calcul efficace qui évite le calcul explicite des moments, ce qui est coûteux en pratique. Cela permet de réduire considérablement les coûts de calcul. Nous avons implémenté les algorithmes présentés dans cet article pour les gradients de la fonction de coût, et nous les avons validés avec un exemple synthétique d'un mélange de 3 gaussiennes en observant la décroissance de l'erreur entre le moment empirique et le moment théorique en appliquant la méthode de descente de gradient avec les gradients calculés. L'approche présentée dans cet article est prometteuse et ouvre la porte à davantage de travaux sur les méthodes de moments pour le problème d'identification des mélanges de distributions.

Bibliography

[Bou98] Nicolas Bourbaki. *Algebra I: Chapters 1–3, Elements of Mathematics*. Springer-Verlag, Berlin, 1998.

[Gut09] A. Gut. *An Intermediate Course in Probability*. Springer Texts in Statistics. Springer New York, 2009.

[HK13] Daniel Hsu and Sham M. Kakade. Learning mixtures of spherical gaussians: Moment methods and spectral decompositions. In *Proceedings of the 4th Conference on Innovations in Theoretical Computer Science*, ITCS '13, pages 11–20, New York, NY, USA, January 2013. Association for Computing Machinery.

[KMM22] Rima Khouja, Pierre-Alexandre Mattei, and Bernard Mourrain. Tensor decomposition for learning gaussian mixtures from moments. *Journal of Symbolic Computation*, 113:193–210, 2022.

[PKK22] João M. Pereira, Joe Kileel, and Tamara G. Kolda. Tensor moments of gaussian mixture models: Theory and applications, 2022.