Caractérisation 3D de grains à partir d'images 2D
Biomasse

Alaa, Rayen, Amadou

CentraleSupélec

18 avril 2025

Granulométrie et Caractérisation 3D

Contexte :

- Utilisation d'un granulomètre à CentraleSupélec pour caractériser des grains de poudre (bois, etc.).
- Les mesures sont 2D (photographies, traitement d'image) donnant des distributions de caractéristiques (surface, circularité, etc.).

Problématique :

- Peut-on remonter aux caractéristiques 3D (forme, volume) à partir de simples observations 2D ?
- Grand intérêt industriel : prévision de la coulabilité, mise en forme, procédés de transformation.

Exemple introductif : particules ellipsoïdales

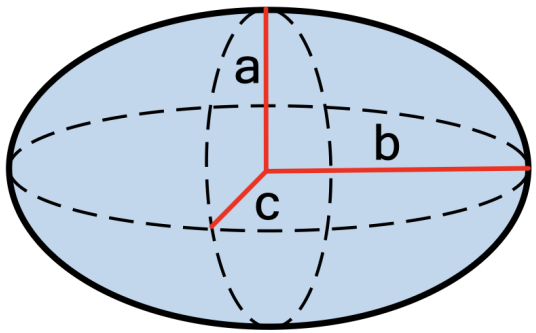
Hypothèses simplificatrices :

- On modélise chaque particule comme une **ellipsoïde** défini par (a, b, c), avec

$$\frac{x^2}{a^2} + \frac{y^2}{b^2} + \frac{z^2}{c^2} = 1.$$

- Pour garantir une *unicité* de la représentation et éviter les ambiguïtés, on impose :

$$a \geq b \geq c > 0.$$



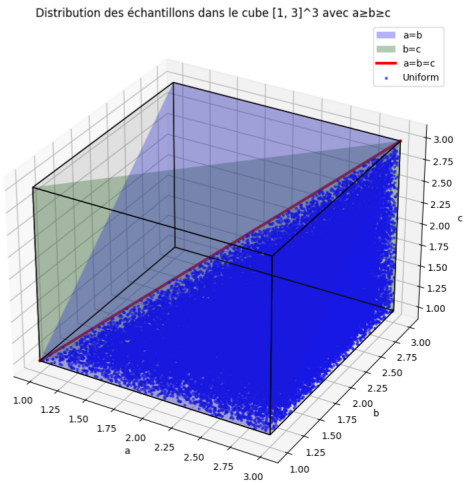
Génération des ellipsoïdes

Modélisation des axes 3D

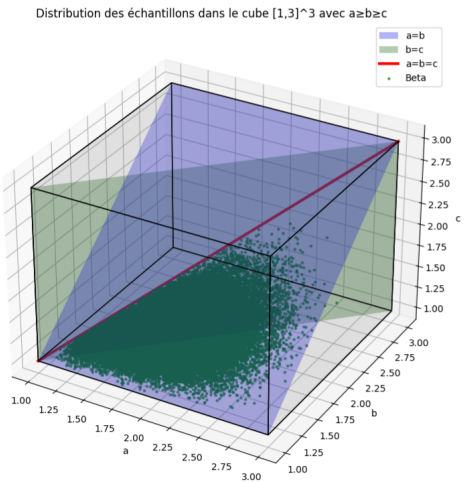
Chaque particule est représentée par un ellipsoïde dont les demi-axes (a, b, c) sont tirés dans la région

$$[u,\ u + l]^3,$$

avec la contrainte $a \geq b \geq c$.



Uniforme

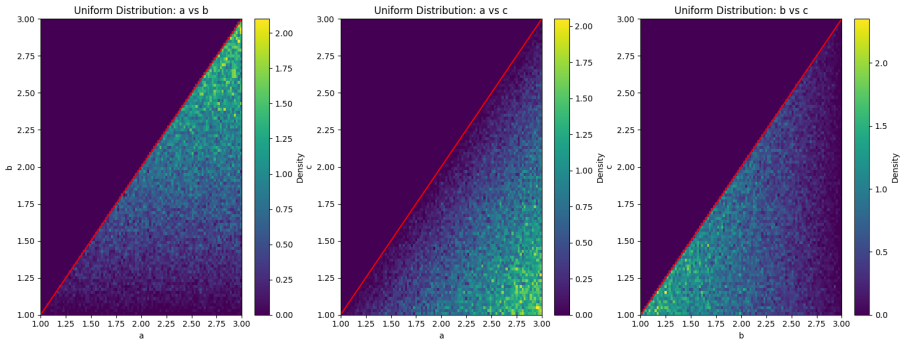


Beta avec $\alpha = 2,\ \beta = 5$

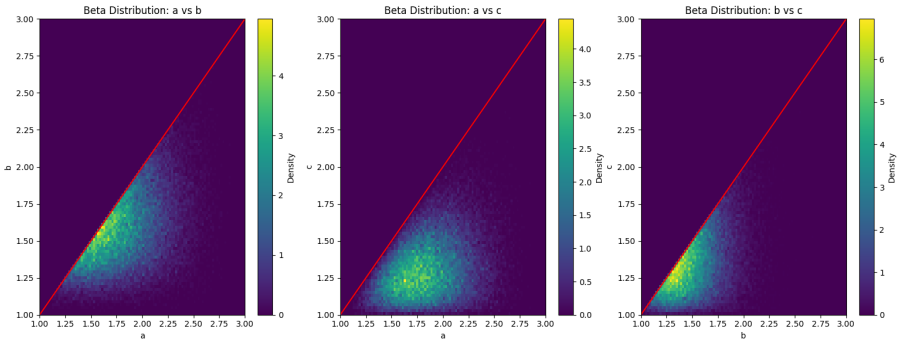
Distributions conjointes des axes 3D

Observation

Les lois initiales (Uniforme ou Beta) dans $[u, u + l]^3$ induisent des distributions conjointes pour les paires d'axes (a, b), (a, c) et (b, c).



Distributions conjointes (Uniforme)



Distributions conjointes (Beta)

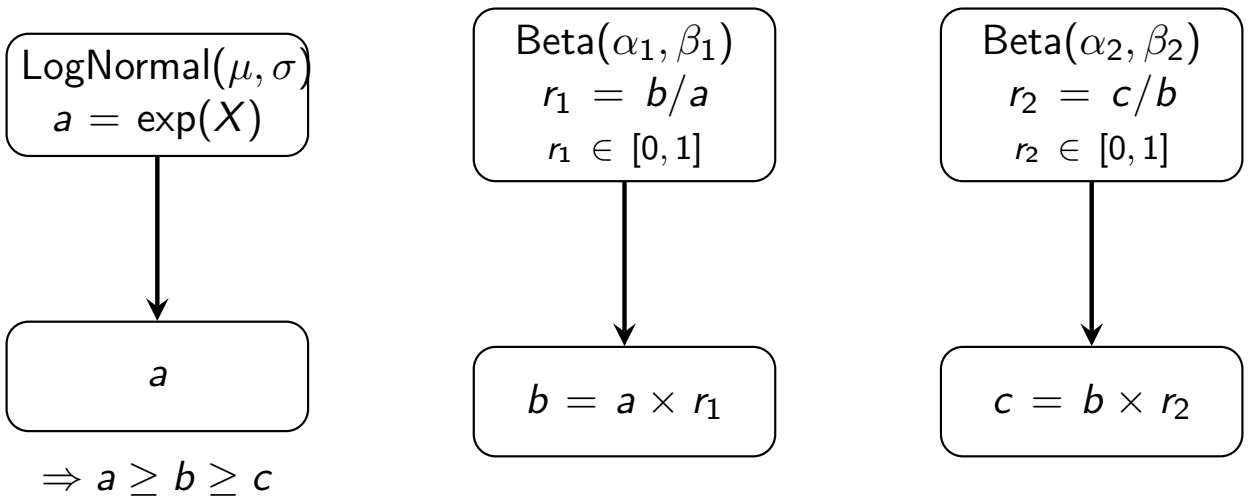
Génération avec LogNormBeta

Principe LogNormBeta

- Le grand axe a est tiré d'une loi log-normale : $\log(a) \sim N(\mu, \sigma)$.
- Les rapports $r_1 = \frac{b}{a}$ et $r_2 = \frac{c}{b}$ sont tirés de lois Beta classiques, donc $r_1, r_2 \in [0, 1]$.
- On définit alors :

$$b = a \times r_1, \quad c = b \times r_2,$$

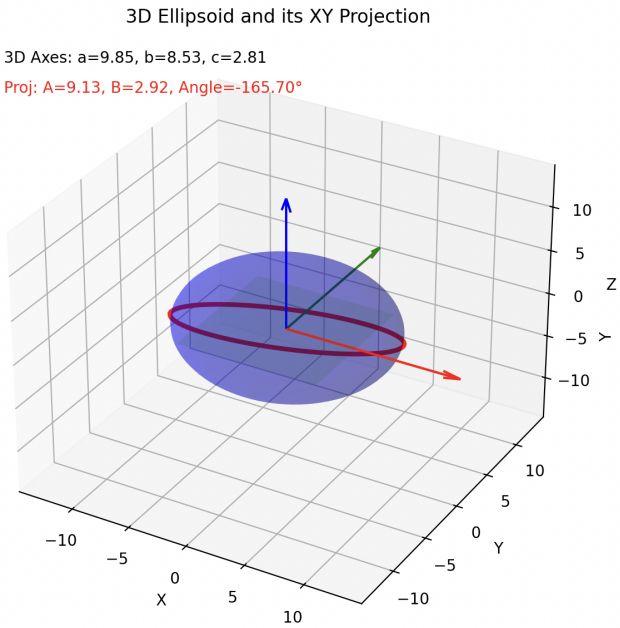
assurant ainsi $a \geq b \geq c$.

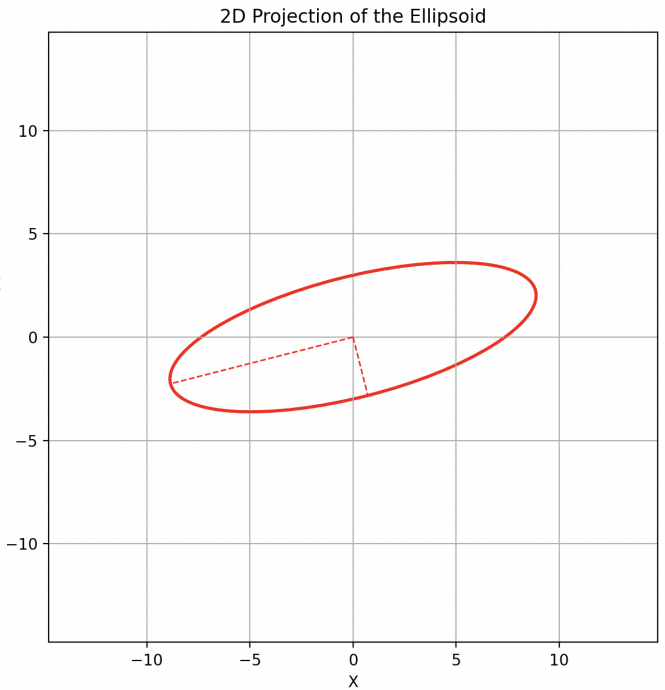


Rotation et Projection des particules

Processus

- Chaque ellipsoïde 3D subit une rotation aléatoire via une classe dédiée (`Generator`).
- La projection sur le plan XY génère une ellipse caractérisée par ses axes A (max) et B (min).

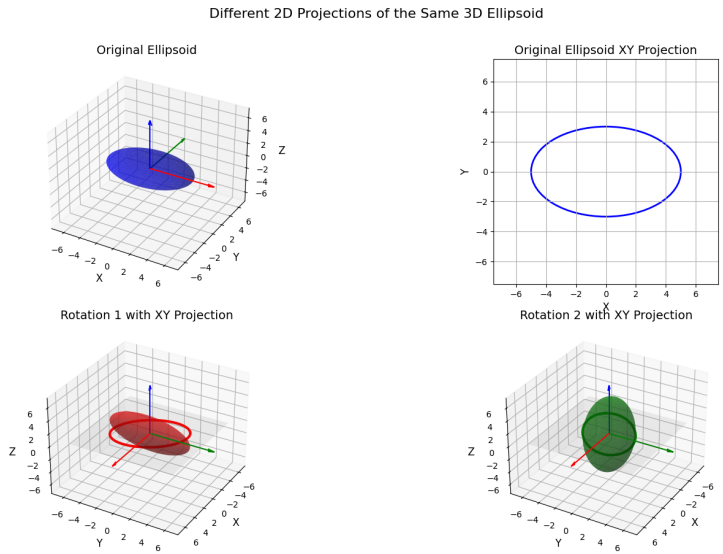




Défis du problème inverse

Problématique

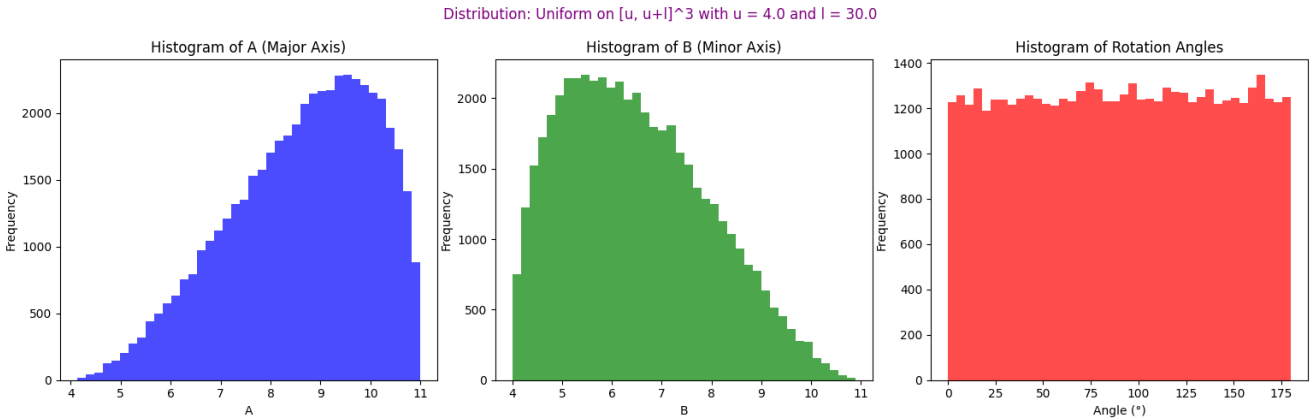
- La projection 3D $\rightarrow$ 2D est **non bijective** : différentes formes 3D peuvent produire des projections 2D identiques.
- Peut-on retrouver la distribution des axes 3D à partir des projections 2D observées ?



Problème inverse

Problématique

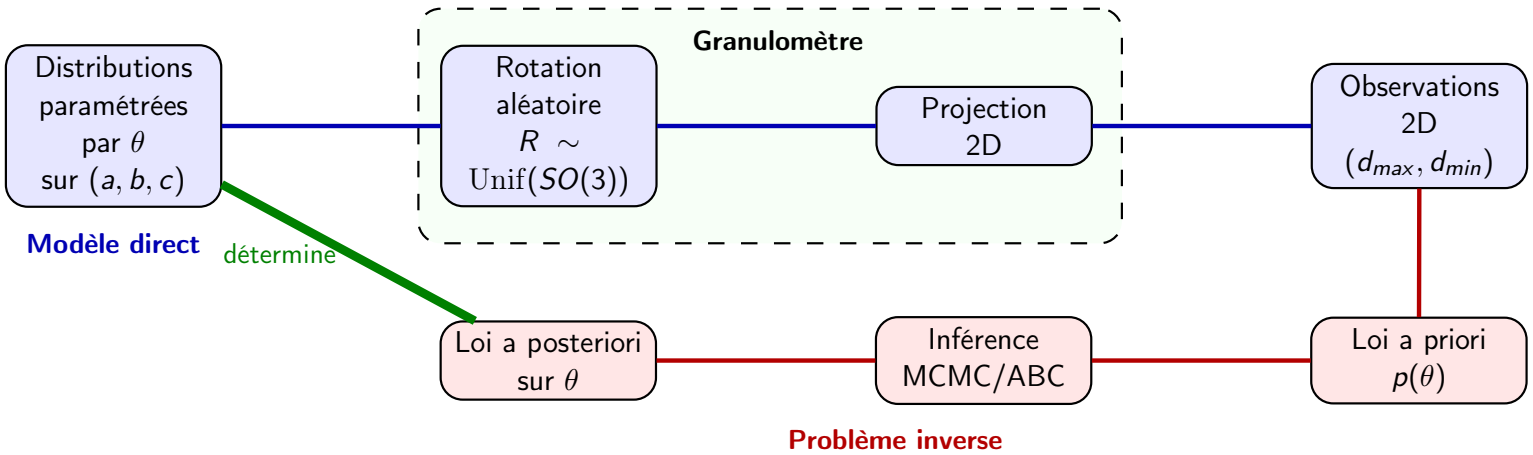
- La projection 3D $\rightarrow$ 2D est **non bijective** : différentes formes 3D peuvent produire des projections 2D identiques.
- Peut-on retrouver la distribution des axes 3D à partir des projections 2D observées ?



Exemple introductif : Synthèse de l'approche

Objectif : Comprendre le lien entre la distribution des formes 3D des particules et les observations 2D via le granulomètre. **Pourquoi cet exemple est pertinent ?**

- Simuler un processus complet : de la loi sur (a, b, c) jusqu'à la projection observée (d_{max}, d_{min}).
- Illustrer simplement les mécanismes avec des distributions modélisées.
- Offrir une base générale pouvant être étendue à toute forme dont la projection peut être calculée.



Plan de la Présentation

Objectif Global

Modéliser la forme 3D et estimer ses paramètres

- On pose un **modèle probabiliste** pour la forme 3D.
- Les paramètres de ce modèle (loi) sont incertains $\rightarrow$ **approche bayésienne**.
- On met à jour ces paramètres à partir de données 2D réelles.

Approche Méthodologique

- **Simulation** 3D $\rightarrow$ 2D (projection avec rotation aléatoire).
- **MCMC** pour explorer l'espace des paramètres et ajuster la loi a posteriori.

Notations Principales

- E : variable aléatoire décrivant la forme 3D (ex. distribution de grains).
- R : rotation aléatoire, $R \sim \mathrm{Unif}(SO(3))$.
- Y : caractéristiques 2D (projection du grain), $Y = P(E, R)$.
- θ : paramètre décrivant la loi de E (ex. π_θ).

$$p(E, R) = p(E)p(R), \quad R \text{ indépendant de } E.$$

$$\text{Observation : } Y = P(E, R).$$

Cadre Bayésien

A priori et Vraisemblance

- $p_T(\theta)$: prior sur θ.
- $p_{Y|T=\theta}(y)$: probabilité d'observer y sachant θ.

Posterior (loi a posteriori) :

$$p_{T|Y}(\theta \mid y) = \frac{p_{Y|T=\theta}(y)\, p_T(\theta)}{p_Y(y)}.$$

Problème : Calculer ou approximer $p_{Y|T=\theta}(y)$ est souvent complexe.
Solution : *Méthodes de simulation* (Approximate Bayesian Computation, MCMC avec un générateur 3D→2D, etc.).

Pourquoi une approche avec MCMC ?

- L'espace des paramètres θ peut être de grande dimension (formes complexes).
- Les postérieurs peuvent être multi-modaux ou non analytiques.
- MCMC permet d'échantillonner (au lieu de calculer l'intégrale) :

$$\theta^{(1)}, \theta^{(2)}, \ldots, \theta^{(N)} \sim p_{T|Y}(\theta \mid y).$$

- On peut ensuite estimer la valeur moyenne, l'intervalle de confiance, etc.

Méthode MCMC : Vue d'ensemble

Objectif : approximer une loi a posteriori $p(\theta \mid y)$ lorsqu'elle est intractable analytiquement.

Étapes de l'approche bayésienne par MCMC :

1. **Définir un prior** $p(\theta)$ sur les paramètres inconnus θ.
2. **Construire un forward model** : à partir de θ, simuler des données synthétiques y_{sim} via un générateur 3D$\rightarrow$2D.
3. **Définir une vraisemblance** (ou une distance) entre y_{sim} et les données observées y.
4. **Appliquer un algorithme MCMC** pour échantillonner depuis $p(\theta \mid y)$.

Algorithme Metropolis-Hastings (MCMC)

Principe : construire une chaîne de Markov dont la loi limite est la postérieure $p(\theta \mid y)$.

Pseudo-code :

- Initialiser $\theta^{(0)}$
- **Pour** $t = 1$ à $N - iter$:
  - Proposer $\theta' \sim q(\theta' \mid \theta^{(t-1)})$ (loi de proposition, ex. gaussienne centrée)
  - Calculer le rapport :

$$\alpha = \min\left(1, \frac{p(y \mid \theta') \cdot p(\theta')}{p(y \mid \theta^{(t-1)}) \cdot p(\theta^{(t-1)})}\right)$$

  - Tirer $u \sim \mathrm{Unif}(0, 1)$:
    - si $u < \alpha$, accepter $\theta^{(t)} = \theta'$
    - sinon, $\theta^{(t)} = \theta^{(t-1)}$

Résultat : une séquence $\{\theta^{(t)}\}_{t=1}^{N}$ distribuée selon la loi postérieure.

Application MCMC : estimation d'une loi 3D

Expérience numérique :

- On simule des observations (D_{max}, D_{min}) à partir d'une distribution 3D connue (uniforme sur $[u, u + l]$).
- On essaye ensuite de retrouver les paramètres (u, l) à partir de ces données 2D.
- Pour cela, on utilise notre package `mcmc` développé spécifiquement.

Configuration :

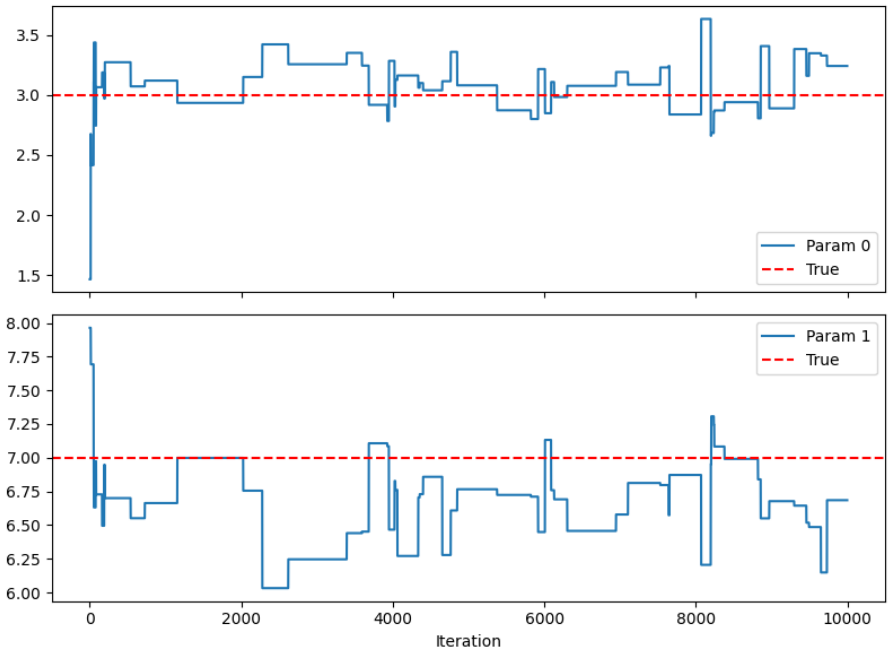
- Vraie loi : $U \sim \mathrm{Unif}(3, 10)$ donc $u = 3$, $l = 7$.
- A priori : $u, l \in [0, 10]$.
- Méthode : MCMC avec 10 000 itérations, noyau KDE.

Résultat obtenu :

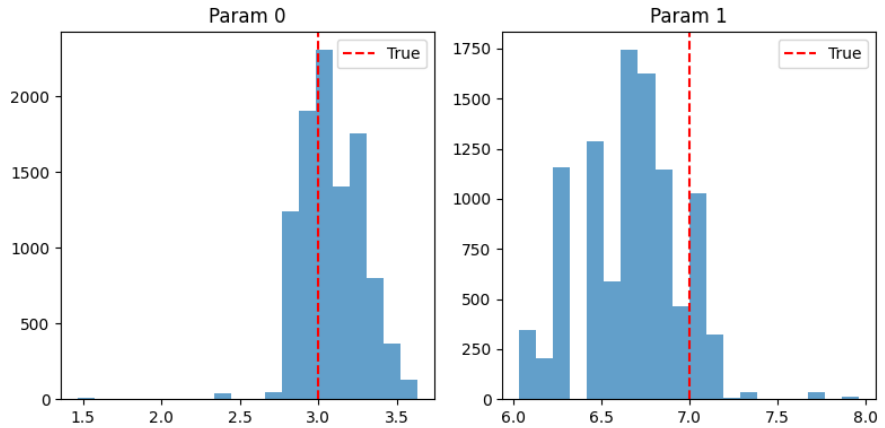
$$\hat{u} \approx 3.07, \quad \hat{l} \approx 6.68$$

$\rightarrow$ très proche de la vérité !

Visualisation des résultats MCMC



Trace des chaînes MCMC (Paramètres u et l)



Distribution postérieure (Histogrammes de u et l)

Les chaînes convergent bien, et la distribution postérieure est concentrée autour des vraies valeurs.

Discussion Générale sur la méthode MCMC

- **Avantages** :
  - Approche bayésienne robuste pour gérer l'incertitude.
  - Flexible (peut s'étendre à des formes 3D plus complexes).
- **Limites** :
  - Coût de calcul élevé (simulation 3D→2D, MCMC).
  - Choix de la fonction de vraisemblance (approximation ou ABC) est sensible.
  - Le choix des hyperparmètres peut avoir un impact sur la convergence (par exemple converger vers un minimum local pour une bande passante (bw) très faible ou très élevée)

Principe de l'approche fréquentiste

Rappel formulation bayésienne :

$$p_{T|Y}(\theta \mid y) \;=\; \frac{p_{Y|T=\theta}(y)\; p_T(\theta)}{p_Y(y)}.$$

Au lieu de chercher toute la loi *a posteriori* $p_{T|Y}(\theta \mid y)$, l'approche **fréquentiste** cherche directement

$$\hat{\theta} \;=\; \arg\max_{\theta}\Big\{\, p_{Y|T=\theta}(y)\; p_T(\theta)\Big\}.$$

Conséquence :

- On définit un problème *d'optimisation explicite*,

$$\max_{\theta} \;\; \mathcal{L}(\theta, y).$$

- On a besoin d'une **métrique de distance** (ou de vraisemblance) entre les données simulées et réelles.
- Puis on applique un **algorithme d'optimisation** global ou local pour trouver le meilleur θ.

Distance via noyau KDE

Distance KDE :

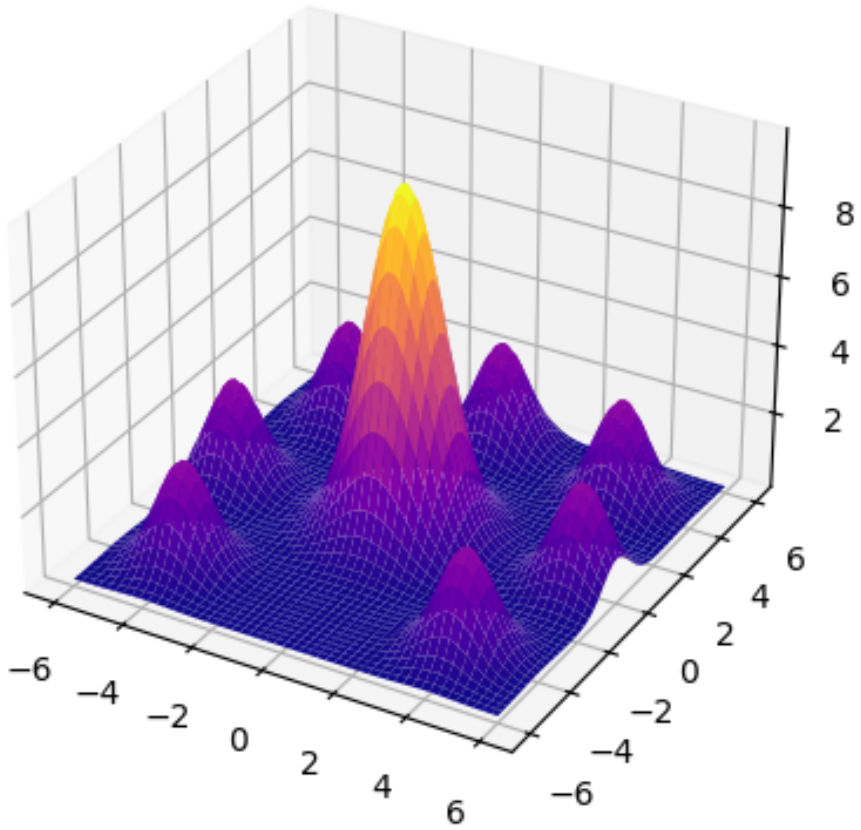
- Pour un θ, on génère des données simulées y_{sim} à l'aide du modèle 3D $\rightarrow$ 2D.
- On construit une loi empirique via un **estimateur à noyau** (KDE) sur y_{sim}.
- On calcule la **vraisemblance** de y_{obs} sous cette densité estimée, puis on la *négative* (pour avoir un «coût» à minimiser) :

$$D(\theta) \;=\; -\sum_{i=1}^{N} \log\big(\widehat{p}_\theta(y_{\text{obs},i})\big).$$

Pourquoi cette distance est-elle pertinente ?

- Si y_{sim} est «assez gros» et bien tiré pour θ, la KDE *converge* vers la vraie loi engendrée par θ.
- Minimiser le *negative log-likelihood* revient à trouver les paramètres expliquant le mieux y_{obs}.

Exemple de fonction de vraisemblance



Exemple de fonction de vraisemblance

Problème d'optimisation multi-modal

- Même si on calcule la vraisemblance/Distance comme ci-dessus, la topologie peut être **complexe** : plusieurs maxima locaux.
- Sur la figure, on voit un exemple 3D d'une vraisemblance en fonction de deux paramètres seulement : présence d'un maximum global et de maxima locaux.

D'où l'importance d'utiliser, par exemple, des **optimiseurs globaux** pour éviter de rester piégé dans un maximum local.

Algorithmes d'optimisation utilisés

Plusieurs stratégies pour minimiser la distance :

- *Minimize (local)* :
  - Méthodes type **Nelder-Mead**, **L-BFGS-B**, etc.
  - Rapides mais risquent de stagner dans un minimum local.
- *DifferentialEvolution (global)* :
  - **Algorithme évolutionnaire** : population de solutions, mutations, etc.
  - Moins sensible aux minima locaux, mais plus lourd en calcul.
- *PSO (Particle Swarm Optimization)* :
  - **Essaim de particules** cherchant le meilleur global.
  - Tir aléatoire + inertie, converge globalement mais coûteux.

Expérience : Loi LognormBetaBeta

Hypothèse : on a la loi `LognormBetaBeta` qui génère (a, b, c) avec :

$$a = \exp(\mathcal{N}(\mu, \sigma)), \quad b = a \cdot \mathrm{Beta}(\alpha_1, \beta_1), \quad c = b \cdot \mathrm{Beta}(\alpha_2, \beta_2).$$

Ici, $a \geq b \geq c$ par construction.

Observation :

- Comme b et c dépendent séquentiellement de a, on obtient souvent un unique maximum global.
- On simule les données avec

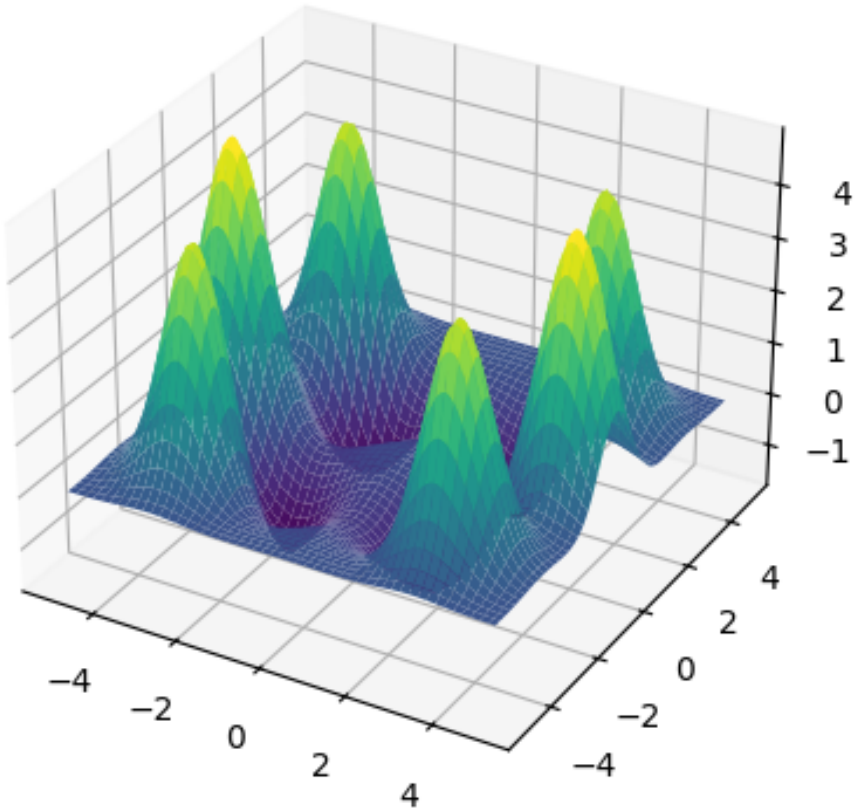
$$\theta^* = [0.5, 0.3, 2.0, 5.0, 2.0, 5.0].$$

- Estimation fréquentiste par min. de la distance KDE :

$$\widehat{\theta} \approx [0.501, 0.312, 1.906, 4.714, 2.681, 6.665]$$

(vs. MCMC : $[0.522, 0.357, 1.599, 3.305, 3.370, 8.282]$).

Exemple `TriLogNormal` : 6 paramètres indépendants



Exemple `TriLogNormal` : 6 paramètres indépendants

Idée : Chaque axe (a, b, c) a sa propre loi lognormale (paramètres μ_i, σ_i), puis on trie (a, b, c) en décroissant.

$$a = \text{Lognormal}(\mu_1, \sigma_1),\ \ b = \text{Lognormal}(\mu_2, \sigma_2),\ \ c = \text{Lognormal}(\mu_3, \sigma_3).$$

Conséquence :

- On a 6 paramètres à estimer ; la *fonction de vraisemblance* peut présenter **plusieurs maxima globaux** (schéma *image4*).
- Par exemple, en simulant avec

  $$\theta = [1.0, 0.5,\ \ 2.0, 0.8,\ \ 0.0, 0.9\,],$$

  on obtient en pratique

  $$\widehat{\theta} \approx [0.798,\ \ 0.560,\ \ 0.177,\ \ 0.618,\ \ 1.934,\ \ 0.794\,].$$

- Des variations plus importantes entre itérations si l'algorithme bascule entre plusieurs sommets.

Comparaison entre différentes lois

Idée : Générer des données avec une loi (ex. LogNormal) puis essayer de les *fitter* avec plusieurs lois (Uniform, Gamma, LogNormal).

- On s'attend à ce que la **distance minimale** soit obtenue avec la loi correcte (c'est-à-dire celle utilisée pour la génération).
- Les autres lois peuvent approcher plus ou moins bien le dataset (ex. queue plus épaisse, mode différent).

Application aux données réelles (1/2)

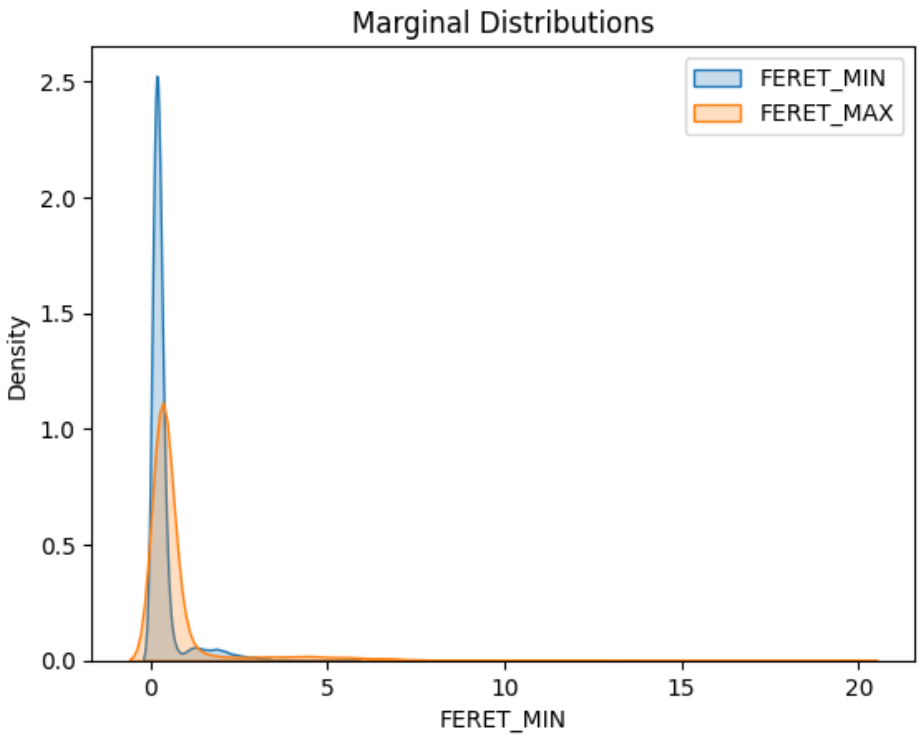
Procédure :

- Charger les données réelles ($d_{\text{max}}, d_{\text{min}}$) depuis le granulomètre.
- Appliquer la même *distance KDE + DifferentialEvolution* pour estimer Uniform, LogNormal, Gamma.

Informations sur les données :

- Moyenne de $d_{\text{min}} \approx 0{,}368$, minimum $d_{\text{min}} \approx 0{,}096$.
- Moyenne de $d_{\text{max}} \approx 0{,}8216$, maximum $d_{\text{max}} \approx 19{,}72$.
- Ces valeurs guident le choix des bornes pour l'algorithme d'optimisation.

Application aux données réelles (1/2)



Application aux données réelles (2/2)

Distribution empirique :

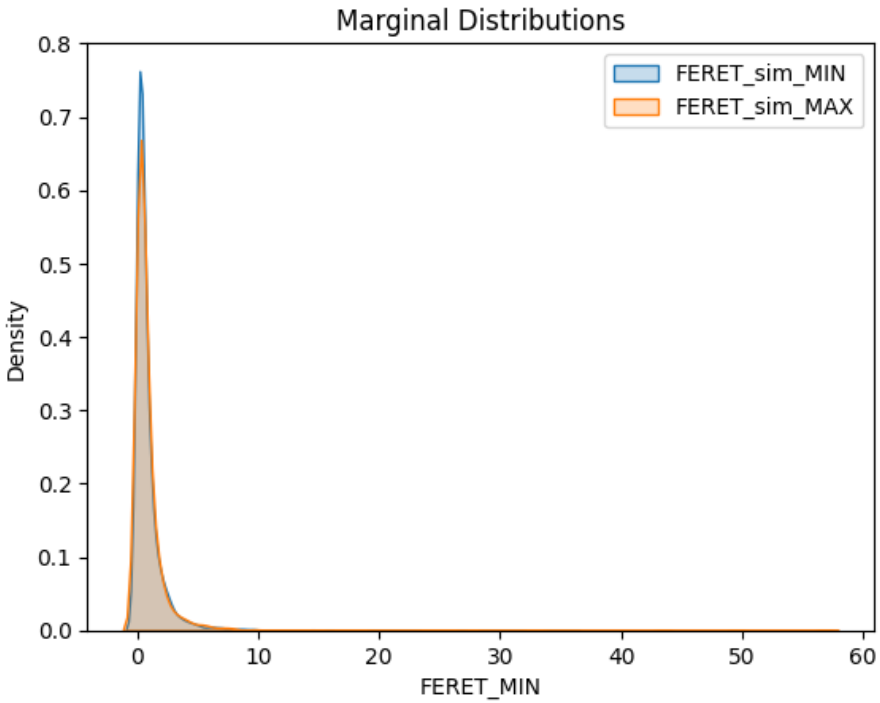
- Illustration de la répartition de d_{min} et d_{max}.
- Visuellement, on observe une forte concentration autour de valeurs centrales avec des *queues épaisses* ("fat tails"), suggérant qu'une loi LogNormale pourrait être la plus adaptée.

Résultats (exemple) :

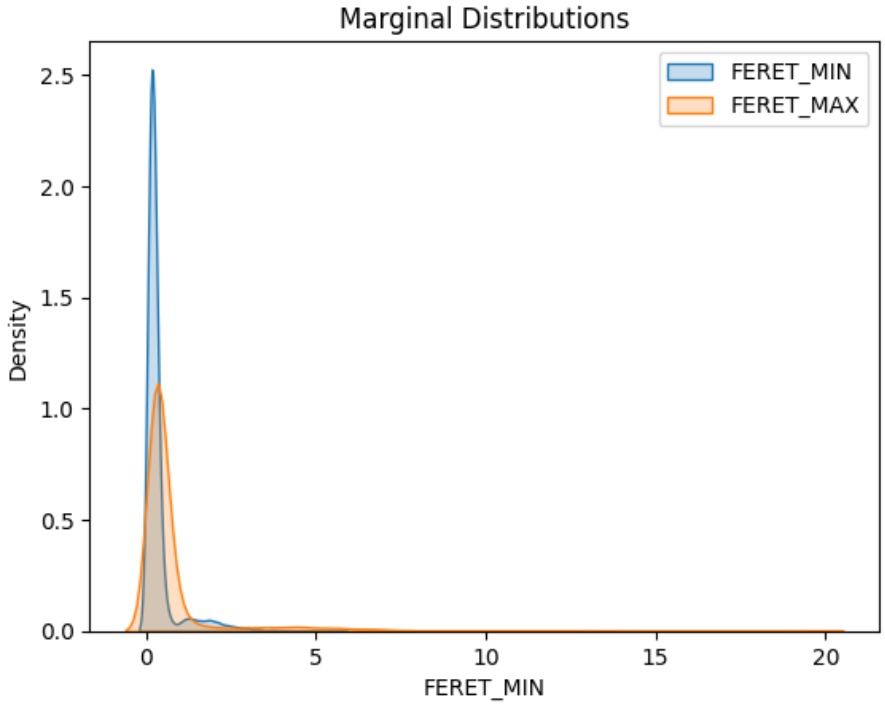
- $\widehat{\theta}_{\text{uni}} = [0.0226,\ 7.2491],$ distance ≈ 9899.6
- $\widehat{\theta}_{\text{lognorm}} = [-0.9715,\ 1.3051],$ distance ≈ 1321.55
- $\widehat{\theta}_{\text{gamma}} = [0.475,\ 1.915],$ distance ≈ 1796.88

Remarque : Les résultats confirment l'intuition : la LogNormale obtient la distance la plus faible.

Comparaison visuelle



Simulated Data marginal densities



Real Data marginal densities

Conclusion et perspectives

- La corrélation observée entre **FERET_MAX** et **FERET_MIN** est de **0.83**.
- Cela suggère qu'un modèle intégrant une **corrélation entre les 3 paramètres initiaux** pourrait être plus pertinent.
- Cette forte corrélation provient probablement du fait que ces deux dimensions sont **scalées de manière similaire** selon leur **distance à la caméra** au moment de la prise de vue.
- **Prochaine étape** : inclure explicitement cette dépendance dans la modélisation pour capturer plus fidèlement la structure réelle des données.