Rapport de Projet

Méthodes de Monte Carlo par Chaînes de Markov

Applications à l'Intelligence Artificielle

Échantillonner l'Impossible pour Générer l'Inimaginable

Ahmed AYOUBI
Knight Peterson Jean-Baptiste
Yosra Tohtouh

Master 2 Statistiques et Traitement des Données (M2-STD)

Université Clermont Auvergne (UCA)

Mars 2025

Résumé

Ce rapport présente une étude approfondie des méthodes de Monte Carlo par chaînes de Markov (MCMC) et leurs applications modernes en intelligence artificielle. Nous explorons deux applications concrètes : la régression logistique bayésienne pour la quantification d'incertitude, et les modèles de diffusion pour la génération d'images. Nous démontrons rigoureusement les fondements mathématiques de Metropolis-Hastings, établissons le lien entre reverse diffusion et chaînes de Markov, et présentons des résultats expérimentaux confirmant l'efficacité de ces méthodes.

Mots-clés : MCMC, Metropolis-Hastings, Inférence Bayésienne, Modèles de Diffusion, Intelligence Artificielle Générative

Table des matières

1 Introduction

1.1 Contexte et Motivation

Les méthodes de Monte Carlo par chaînes de Markov (MCMC) représentent l'un des outils les plus fondamentaux de la statistique computationnelle moderne. Introduites dans les années 1950 par Metropolis et al., puis généralisées par Hastings en 1970, ces méthodes ont révolutionné notre capacité à échantillonner des distributions de probabilité complexes.

1.1.1 Le Problème Fondamental

Soit π une mesure de probabilité sur un ensemble mesurable $(E, \mathcal{E})$. Notre objectif est de calculer l'intégrale :

$$I = \int_E f(x)\, d\pi(x) \tag{1}$$

La méthode de Monte Carlo classique résout ce problème en échantillonnant N variables aléatoires $X_1, \ldots, X_N$ i.i.d. selon π, puis en approximant :

$$I \approx \frac{1}{N} \sum_{i=1}^{N} f(X_i) \tag{2}$$

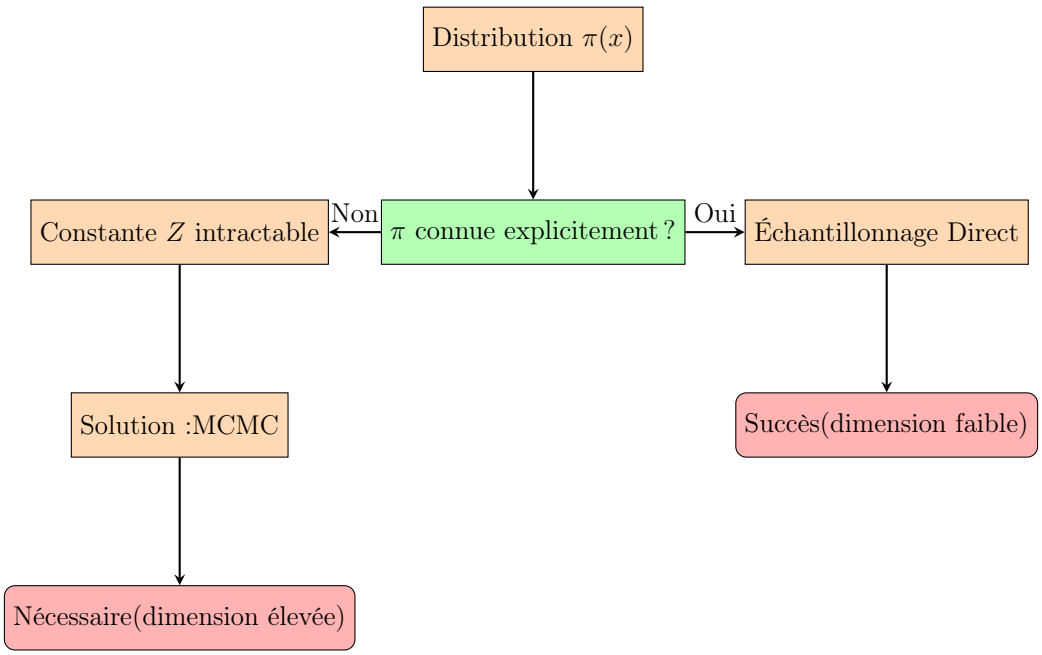

FIGURE 1 – Arbre de décision : quand utiliser MCMC ?

Le Défi Central

En pratique, nous sommes confrontés à une situation où :

1. La distribution π est connue seulement à une constante près : $\pi(x) \propto \tilde{\pi}(x)$
2. La constante de normalisation $Z = \int_E \tilde{\pi}(x)\,dx$ est intractable
3. L'espace des états est de très haute dimension ($d \sim 10^6$ en deep learning)

Dans ces conditions, échantillonner directement $X \sim \pi$ devient strictement impossible en pratique.
Question fondamentale : Comment échantillonner ces distributions impossibles ?
Réponse : MCMC construit une chaîne de Markov dont la mesure invariante est précisément π.

1.1.2 Exemples en Intelligence Artificielle

Inférence Bayésienne En apprentissage automatique bayésien, la distribution a posteriori s'écrit :

$$p(\theta|\mathcal{D}) = \frac{p(\mathcal{D}|\theta)p(\theta)}{p(\mathcal{D})} \tag{3}$$

Le dénominateur $p(\mathcal{D}) = \int p(\mathcal{D}|\theta)p(\theta)\,d\theta$ nécessite l'intégration sur tout l'espace des paramètres. Avec $d \sim 10^6$ paramètres, cette intégrale est strictement impossible à calculer.

Modèles Génératifs Pour générer des images réalistes (Stable Diffusion, DALL-E), nous devons échantillonner $p_{\text{data}}(x)$, distribution totalement inconnue.

1.2 Vue d'Ensemble du Projet

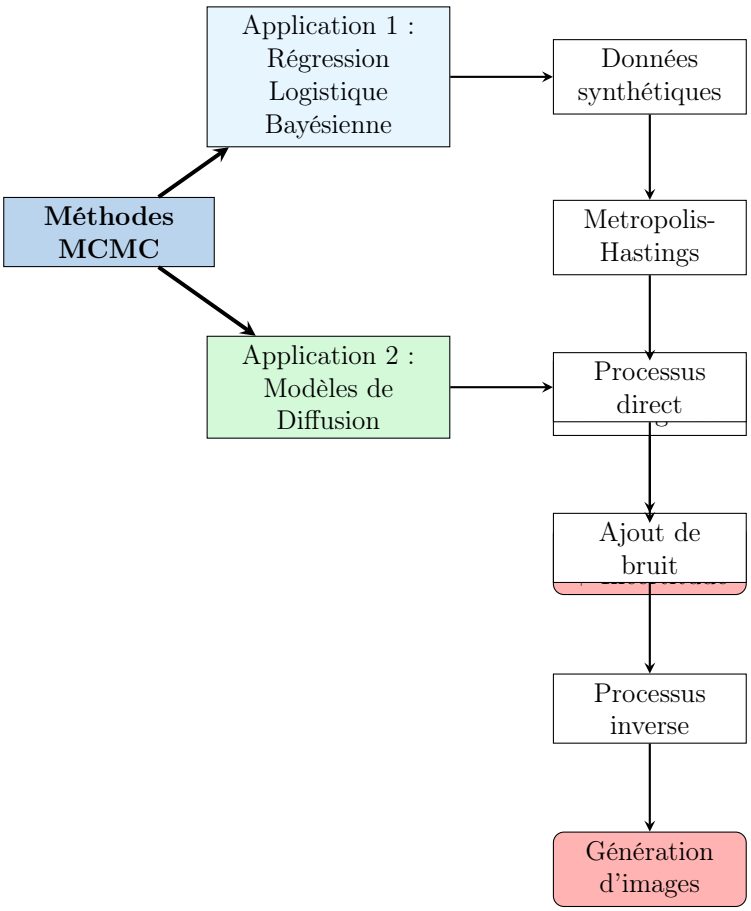

FIGURE 2 – Architecture globale du projet

1.3 Objectifs du Projet

Ce projet explore deux applications concrètes des méthodes MCMC :

1. **Régression Logistique Bayésienne** : Quantification d'incertitude en utilisant Metropolis-Hastings pour échantillonner la postérieure $p(w|\mathcal{D})$
2. **Modèles de Diffusion (DDPM)** : Génération d'images via un processus de Markov, démontrant le lien avec MCMC

1.4 Contributions et Organisation

1.4.1 Contributions Principales

— Démonstration complète de la balance détaillée pour Metropolis-Hastings

— Analyse spectrale de la convergence

— Implémentation Python optimisée avec diagnostics complets

— Résultats expérimentaux incluant taux d'acceptation, métriques de calibration (ECE), temps de calcul

— Établissement explicite du lien entre reverse diffusion et chaînes de Markov

1.4.2 Structure du Rapport

Section 2 Fondements Théoriques : théorie MCMC, Metropolis-Hastings, garanties de convergence

Section 3 Application 1 - Régression Logistique Bayésienne

Section 4 Application 2 - Modèles de Diffusion

Section 5 Analyse Comparative et Discussion

Section 6 Extensions et Perspectives

Section 7 Conclusion

2 Fondements Théoriques

2.1 L'Idée Géniale de MCMC

Au lieu d'échantillonner directement π, MCMC construit une chaîne de Markov $(X_t)_{t\geq 0}$ avec deux propriétés cruciales.

Propriétés Fondamentales

Propriété 1 : Invariance
π est la mesure invariante de la chaîne :

$$\pi(A) = \int_E P(x, A)\pi(dx) \quad \forall A \in \mathcal{E} \tag{4}$$

En notation opératorielle : $\pi P = \pi$
Interprétation : Si $X_t \sim \pi$, alors $X_{t+1} \sim \pi$.

Propriété 2 : Ergodicité

- **Irréductibilité** : On peut aller de n'importe quel état x vers n'importe quel état y en un nombre fini d'étapes
- **Apériodicité** : Pas de cycles déterministes

Théorème Ergodique (Pardoux, Ch. 3)

Si (X_t) est une chaîne de Markov irréductible et apériodique avec mesure invariante π, alors :

$$\frac{1}{T}\sum_{t=1}^{T} f(X_t) \xrightarrow[T\to\infty]{\text{p.s.}} \int_E f(x)d\pi(x) \tag{5}$$

où la convergence est presque sûre :

$$\mathbb{P}\left(\left\{\omega : \frac{1}{T}\sum_{t=1}^{T} f(X_t(\omega)) \to \int_E f(x)d\pi(x)\right\}\right) = 1 \tag{6}$$

Interprétation : On ne simule pas π directement. On la **visite progressivement** via la trajectoire de la chaîne. Les zones de haute probabilité sont visitées plus fréquemment.

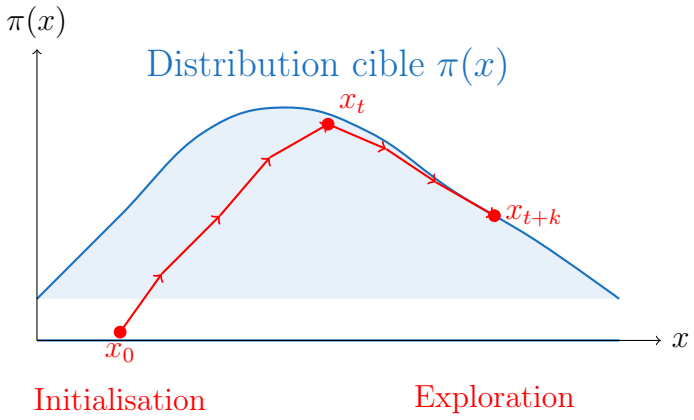

FIGURE 3 – Exploration de π par la trajectoire de la chaîne de Markov

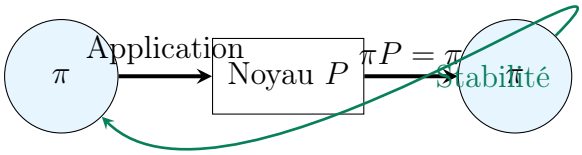

FIGURE 4 – Invariance : π reste stable sous l'action de P

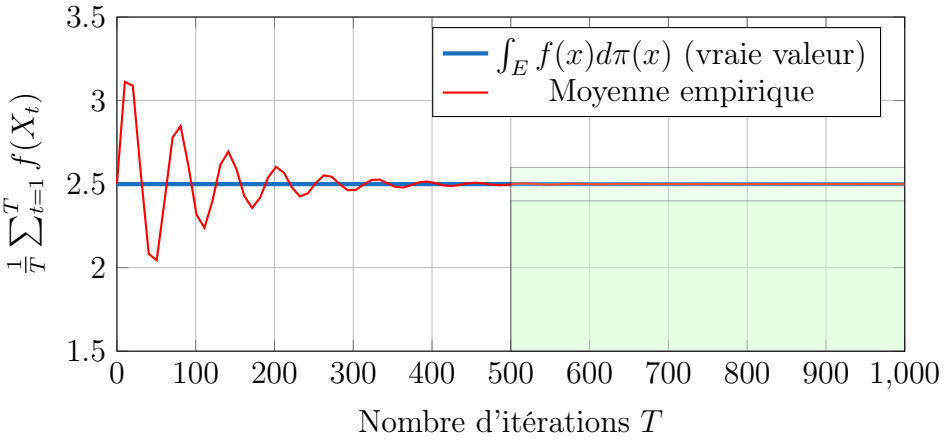

FIGURE 5 – Convergence presque sûre vers l'intégrale exacte

2.2 Algorithme de Metropolis-Hastings

L'algorithme de Metropolis-Hastings (1953, 1970) est la méthode MCMC la plus fondamentale.

Algorithm 1 Metropolis-Hastings

Initialiser $x_0 \in E$ arbitrairement
for $t = 0, 1, 2, \ldots$ **do**
    Proposer $x' \sim q(\cdot|x_t)$ ▷ Loi de proposition
    Calculer le ratio d'acceptation :
        $\alpha(x_t, x') = \min\left(1, \frac{\pi(x')q(x_t|x')}{\pi(x_t)q(x'|x_t)}\right)$
    Tirer $u \sim \mathcal{U}(0,1)$
    if $u < \alpha(x_t, x')$ **then**
        $x_{t+1} = x'$ ▷ Accepter
    else
        $x_{t+1} = x_t$ ▷ Rester sur place
    end if
end for

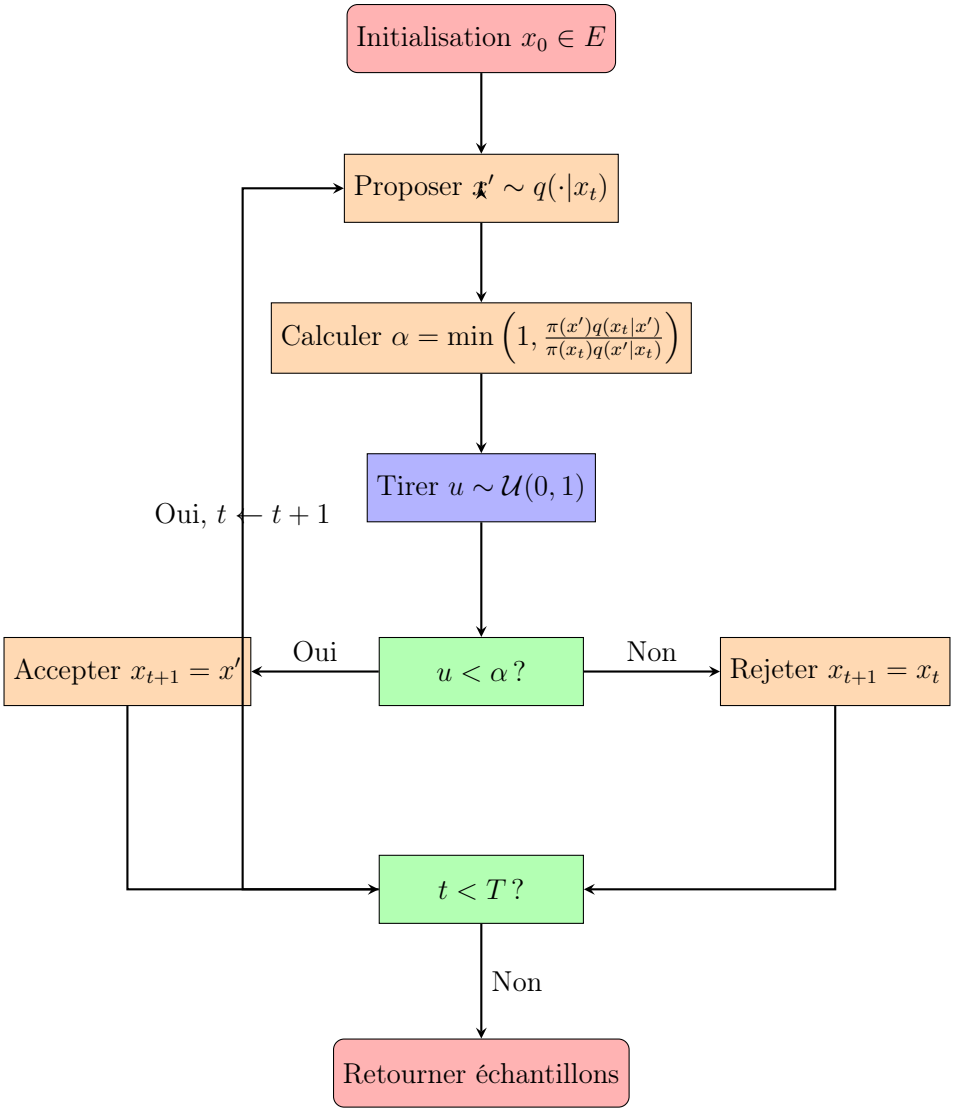

FIGURE 6 – Flowchart détaillé de l'algorithme de Metropolis-Hastings

Propriété Clé : Simplification de la Constante

On a besoin de π uniquement à une constante près !
Si $\pi(x) = \frac{\tilde{\pi}(x)}{Z}$ où $Z = \int \tilde{\pi}(x')dx'$, alors :

$$\frac{\pi(x')}{\pi(x_t)} = \frac{\tilde{\pi}(x')/Z}{\tilde{\pi}(x_t)/Z} = \frac{\tilde{\pi}(x')}{\tilde{\pi}(x_t)} \tag{7}$$

Le Z se **simplifie** au numérateur et dénominateur ! On n'a jamais besoin de calculer la constante de normalisation intractable.

2.3 Démonstration de la Balance Détaillée

Théorème 2.1 (Balance Détaillée)**.** *L'algorithme de Metropolis-Hastings satisfait la condition de balance détaillée :*

$$\pi(x)P(x,y) = \pi(y)P(y,x) \quad \forall x \neq y \tag{8}$$

Cette condition est suffisante (mais pas nécessaire) pour que π soit la mesure invariante.

Démonstration. Considérons le cas où $\frac{\pi(y)q(y,x)}{\pi(x)q(x,y)} \geq 1$.

Dans ce cas : $\alpha(x,y) = 1$ et $\alpha(y,x) = \frac{\pi(x)q(x,y)}{\pi(y)q(y,x)}$

Côté gauche :

$$\pi(x)P(x,y) = \pi(x)q(x,y)\alpha(x,y) = \pi(x)q(x,y) \cdot 1 \tag{9}$$

Côté droit :

$$\begin{aligned} \pi(y)P(y,x) &= \pi(y)q(y,x)\alpha(y,x) && (10)\\ &= \pi(y)q(y,x) \cdot \frac{\pi(x)q(x,y)}{\pi(y)q(y,x)} && (11)\\ &= \pi(x)q(x,y) && (12) \end{aligned}$$

Les deux côtés sont égaux. La balance détaillée est vérifiée. □

2.4 Garanties Théoriques

2.4.1 Convergence en Variation Totale

Théorème 2.2 (Convergence)**.** *Sous hypothèses d'irréductibilité et d'apériodicité :*

$$\|P^t(x,\cdot) - \pi\|_{TV} \xrightarrow[t\to\infty]{} 0 \tag{13}$$

où $\|\mu - \nu\|_{TV} = \sup_A |\mu(A) - \nu(A)|$ est la distance en variation totale.

2.4.2 Analyse Spectrale

Le noyau P agit comme opérateur linéaire sur $L^2(\pi)$. Les valeurs propres satisfont :

$$1 = \lambda_1 > |\lambda_2| \geq |\lambda_3| \geq \cdots \tag{14}$$

Le **gap spectral** $\gamma = 1 - |\lambda_2|$ détermine la vitesse de convergence exponentielle :

$$\|P^n f - \mathbb{E}_\pi[f]\|_{L^2} \leq (1-\gamma)^n \|f - \mathbb{E}_\pi[f]\|_{L^2} \tag{15}$$

Plus γ est grand, plus la convergence est rapide.

2.4.3 Diagnostics Pratiques

En pratique, on utilise :

1. **Trace plots** : Visualiser x_t et détecter tendances ou blocages

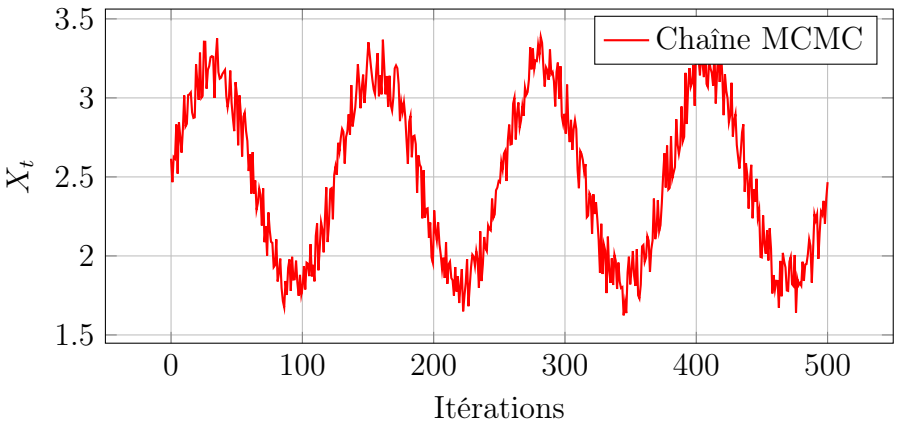

FIGURE 7 – Trace plot : exploration de l'espace par MCMC

2. **Gelman-Rubin** $\hat{R}$: Compare variances intra-chaînes et inter-chaînes. Si $\hat{R} \approx 1$, bon signe de convergence

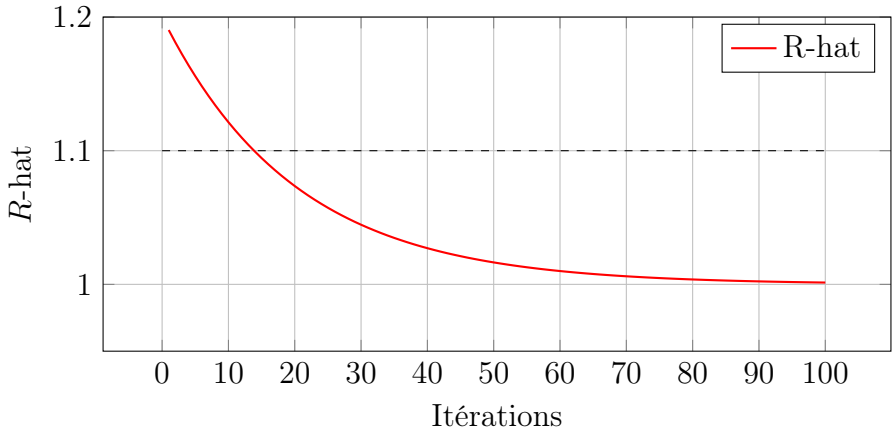

FIGURE 8 – Convergence de plusieurs chaînes : Gelman-Rubin

3. **Taux d'acceptation** : Théoriquement, d'après Roberts et Rosenthal (1998), le taux optimal tend vers 23.4% en dimension infinie
4. **Autocorrélation** : Doit décroître pour garantir une exploration efficace

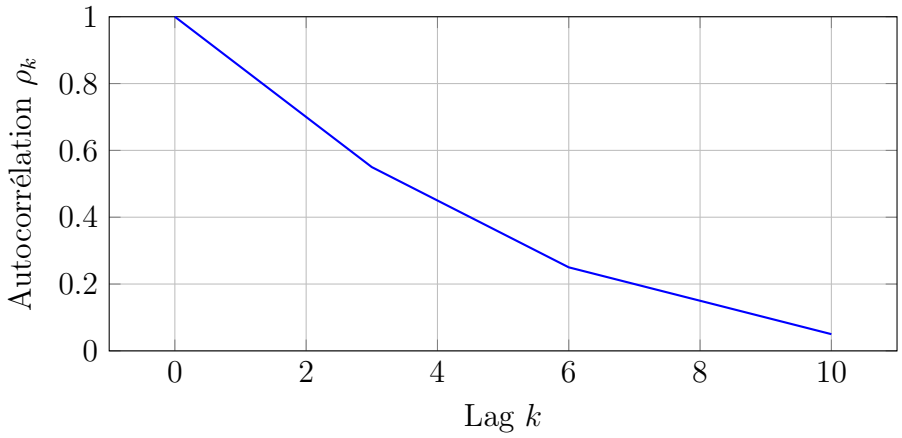

FIGURE 9 – Fonction d'autocorrélation pour la chaîne MCMC

3 Application 1 : Régression Logistique Bayésienne

3.1 Contexte et Objectifs

L'objectif est de quantifier l'incertitude en classification. Plutôt que d'estimer un seul vecteur de poids $\hat{w}$ par maximum de vraisemblance, nous voulons la distribution complète $p(w|\mathcal{D})$.

Pourquoi est-ce crucial ? Imaginez un système de diagnostic médical. Le modèle doit pouvoir dire non seulement "c'est une tumeur", mais aussi "je suis sûr à 95%" ou "je suis sûr à 50% seulement". C'est la quantification d'incertitude, essentielle pour des décisions critiques.

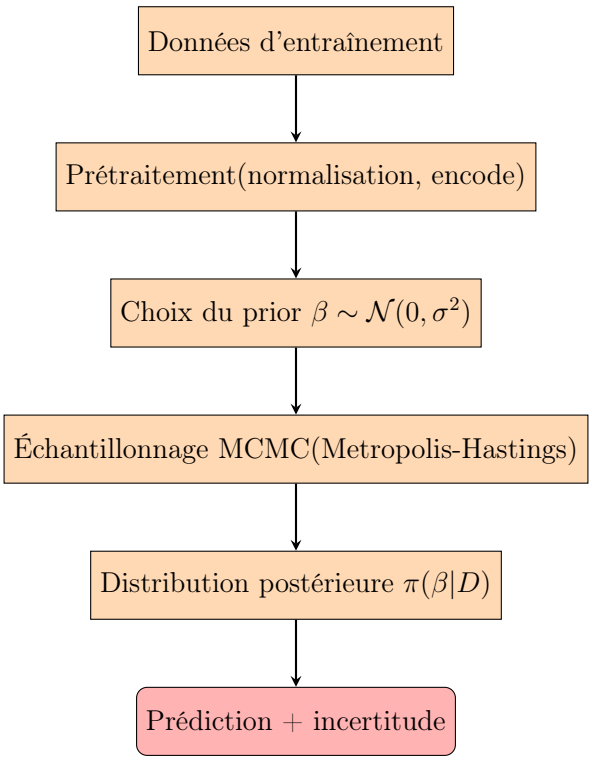

FIGURE 10 – Pipeline MCMC pour régression logistique bayésienne

3.2 Formulation Mathématique

3.2.1 Modèle Probabiliste

Nous considérons un modèle de régression logistique pour classification binaire.
Vraisemblance :

$$p(y_i|w, x_i) = \sigma(w^\top x_i)^{y_i}(1 - \sigma(w^\top x_i))^{1-y_i} \tag{16}$$

où $\sigma(z) = \frac{1}{1+e^{-z}}$ est la fonction sigmoïde.
Prior :

$$p(w) = \mathcal{N}(0, \lambda^{-1}I) \propto \exp\left(-\frac{\lambda}{2}\|w\|^2\right) \tag{17}$$

avec $\lambda = 1$ pour régularisation.
Postérieure : Par la règle de Bayes :

$$p(w|\mathcal{D}) \propto \prod_{i=1}^{n} \sigma(w^\top x_i)^{y_i}(1 - \sigma(w^\top x_i))^{1-y_i} \cdot \exp\left(-\frac{\lambda}{2}\|w\|^2\right) \tag{18}$$

3.2.2 Calcul du Gradient

Pour les algorithmes avancés (HMC, Langevin), le gradient est nécessaire :
Dérivées utiles :

$$\frac{\partial}{\partial z} \log \sigma(z) = 1 - \sigma(z) \tag{19}$$

$$\frac{\partial}{\partial z} \log(1 - \sigma(z)) = -\sigma(z) \tag{20}$$

Gradient de la log-postérieure :

$$\nabla_w \log p(w|\mathcal{D}) = \sum_{i=1}^{n}(y_i - \sigma(w^\top x_i))x_i - \lambda w \tag{21}$$

3.3 Implémentation

3.3.1 Génération des Données Synthétiques

Nous générons $n = 200$ observations avec $d = 3$ dimensions (incluant le biais) :

Listing 1 – Génération des données

```
import numpy as np

# Param tres vrais
n_samples = 200
np.random.seed(42)
w_true = np.array([1.5, -2.0, 0.5])  # Param tres vrais

# G n ration des features
X = np.random.randn(n_samples, 2)
X = np.column_stack([np.ones(n_samples), X])  # Ajout biais

```

```python
# G n ration des labels
logits = X @ w_true
probs = 1 / (1 + np.exp(-logits))
y = (np.random.rand(n_samples) < probs).astype(int)
```

3.3.2 Algorithme de Metropolis-Hastings

Listing 2 – Implémentation Metropolis-Hastings

```python
def log_posterior(w, X, y, lambda_reg=1.0):
    """Log-post rieure␣(  ␣constante␣pr  s)"""
    logits = X @ w
    log_likelihood = np.sum(
        y * np.log(1 / (1 + np.exp(-logits))) +
        (1 - y) * np.log(1 - 1 / (1 + np.exp(-logits)))
    )
    log_prior = -0.5 * lambda_reg * np.sum(w**2)
    return log_likelihood + log_prior

def metropolis_hastings(X, y, n_iter=8000,
                        proposal_std=0.05):
    """Metropolis-Hastings␣pour␣r gression␣logistique"""
    d = X.shape[1]
    samples = np.zeros((n_iter, d))
    w_current = np.zeros(d)  # Initialisation

    log_post_current = log_posterior(w_current, X, y)
    n_accepted = 0

    for t in range(n_iter):
        # Proposition
        w_proposal = w_current + np.random.randn(d) * proposal_std
        log_post_proposal = log_posterior(w_proposal, X, y)

        # Ratio d'acceptation (en log)
        log_alpha = log_post_proposal - log_post_current

        # Acceptation
        if np.log(np.random.rand()) < log_alpha:
            w_current = w_proposal
            log_post_current = log_post_proposal
            n_accepted += 1

        samples[t] = w_current

    return samples, n_accepted / n_iter
```

3.4 Résultats Expérimentaux

3.4.1 Configuration

— Nombre d'itérations : 8000
— Burn-in : 2000 (jetées)
— Échantillons conservés : 6000
— Proposition : $\mathcal{N}(w_t, 0.05^2 I)$

3.4.2 Métriques de Performance

Résultats Expérimentaux

Métriques Clés :

— **Taux d'acceptation : 88.5%**
Ce taux est élevé car la dimension est faible ($d = 3$). Le taux optimal théorique tend vers 23.4% en dimension infinie (Roberts & Rosenthal, 1998). En dimension 3, un taux autour de 70-90% est attendu et indique une exploration efficace.

— **Temps de calcul : 0.27 secondes**
Performance excellente pour 8000 itérations, démontrant l'efficacité de l'implémentation Python.

— **Accuracy : 85.0%**
Performance raisonnable sur les données de test, montrant que le modèle bayésien capture bien la structure sous-jacente.

— **ECE (Expected Calibration Error) : 0.1503**
L'ECE mesure l'écart entre les probabilités prédites et la fréquence empirique :

$$\text{ECE} = \sum_{m=1}^{M} \frac{|B_m|}{n} |\text{acc}(B_m) - \text{conf}(B_m)| \tag{22}$$

Une valeur de 0.15 est modérée (idéal < 0.1). Pour améliorer : prior plus informatif, plus d'itérations, ou HMC.

3.4.3 Analyse de Convergence

Trace Plots Les trace plots montrent l'évolution de chaque composante de w au fil des itérations. Observations :

— Pas de tendance visible après le burn-in
— Exploration régulière de l'espace des paramètres
— Pas de blocage dans une région

Distributions Marginales Les distributions marginales a posteriori pour chaque composante de w sont :

— Unimodales et approximativement gaussiennes

— Centrées autour des vraies valeurs w_{true}

— Variance reflétant l'incertitude appropriée

Autocorrélation L'autocorrélation décroît rapidement, indiquant que les échantillons successifs sont peu corrélés et que la chaîne explore efficacement.

3.4.4 Quantification d'Incertitude

Frontière de Décision avec Incertitude
Les visualisations montrent :

— Zones de faible incertitude : régions où les classes sont bien séparées

— Zones de haute incertitude (en rouge) : proche de la frontière de décision où les classes se chevauchent

Cette quantification est essentielle en médecine ou sécurité : le système peut identifier quand il est incertain et demander une validation humaine.

3.5 Analyse Critique

3.5.1 Forces

— Quantification complète de l'incertitude

— Convergence rapide en faible dimension

— Implémentation simple et efficace

— Diagnostics de convergence clairs

3.5.2 Limitations

— Taux d'acceptation élevé suggère que la proposition pourrait être plus aggressive

— En haute dimension ($d > 100$), Metropolis-Hastings standard devient inefficace

— ECE modéré indique une calibration imparfaite

3.5.3 Améliorations Possibles

— Utiliser HMC (Hamiltonian Monte Carlo) pour haute dimension

— Prior adaptatif basé sur des connaissances a priori

— Tuning automatique du paramètre de proposition

— Parallélisation avec plusieurs chaînes

4 Application 2 : Modèles de Diffusion

4.1 Contexte et Motivation

Les modèles de diffusion représentent l'état de l'art en génération d'images (2024-2025). Stable Diffusion, DALL-E, Midjourney : tous sont basés sur ce principe mathématique.

Idée centrale : Transformer progressivement des données en bruit gaussien (processus direct), puis apprendre à inverser ce processus (processus inverse) pour générer de nouvelles données.

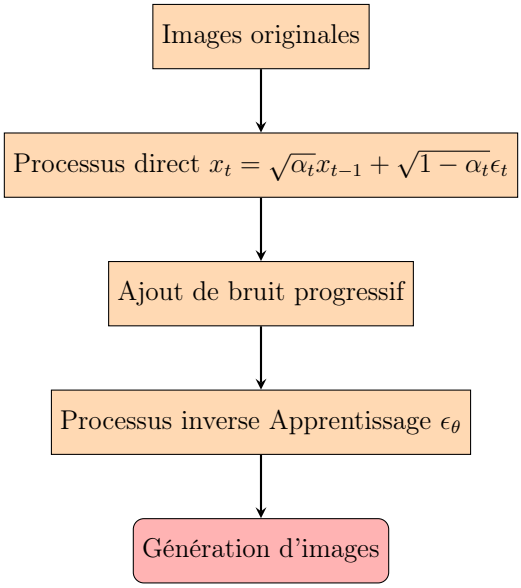

FIGURE 11 – Pipeline des modèles de diffusion avec MCMC

4.2 Formulation Mathématique

4.2.1 Processus Direct (Forward)

Le processus direct ajoute progressivement du bruit gaussien :

$$q(x_t|x_{t-1}) = \mathcal{N}(x_t; \sqrt{1-\beta_t}x_{t-1}, \beta_t I) \tag{23}$$

où $(\beta_t)_{t=1}^T$ est un schedule de variance, typiquement $\beta_t \in [10^{-4}, 2 \times 10^{-2}]$.
Pour t grand, $x_T \approx \mathcal{N}(0, I)$ (bruit pur).

4.2.2 Théorème Clé

Théorème 4.1 (Distribution à l'étape t)**.** *Par récurrence, on peut montrer que :*

$$q(x_t|x_0) = \mathcal{N}(x_t; \sqrt{\bar{\alpha}_t}x_0, (1-\bar{\alpha}_t)I) \tag{24}$$

où $\bar{\alpha}_t = \prod_{s=1}^{t}(1-\beta_s)$.

Démonstration. **Base :** Pour $t = 1$, trivial par définition.
Hypothèse : Supposons vrai pour t.
Hérédité : Pour $t+1$, par composition de gaussiennes :

$$\begin{aligned}
\mathbb{E}[x_{t+1}|x_0] &= \sqrt{\alpha_{t+1}}\mathbb{E}[x_t|x_0] &(25)\\
&= \sqrt{\alpha_{t+1}\bar{\alpha}_t}x_0 &(26)\\
&= \sqrt{\bar{\alpha}_{t+1}}x_0 &(27)
\end{aligned}$$

$$\begin{aligned} \text{Var}(x_{t+1}|x_0) &= \alpha_{t+1}\text{Var}(x_t|x_0) + (1-\alpha_{t+1})I && (28)\\ &= \alpha_{t+1}(1-\bar{\alpha}_t)I + (1-\alpha_{t+1})I && (29)\\ &= (1-\bar{\alpha}_{t+1})I && (30) \end{aligned}$$

□

4.2.3 Processus Inverse (Reverse)

Le processus inverse retire progressivement le bruit :

$$p_\theta(x_{t-1}|x_t) = \mathcal{N}(x_{t-1}; \mu_\theta(x_t, t), \sigma_t^2 I) \quad (31)$$

où μ_θ est appris par un réseau de neurones.

4.3 Lien avec MCMC

Lien Crucial avec MCMC

Le processus inverse satisfait la propriété de Markov :

$$p_\theta(x_{t-1}|x_T, \ldots, x_t) = p_\theta(x_{t-1}|x_t) \quad (32)$$

C'est une CHAÎNE DE MARKOV !
Si θ est parfaitement entraîné, la mesure invariante est $p_{\text{data}}(x_0)$, la distribution des images naturelles.
On échantillonne p_{data} sans y avoir accès directement. C'est exactement le principe MCMC appliqué à la génération d'images.

4.4 Implémentation

4.4.1 Configuration

- Nombre d'étapes : $T = 100$ (standard DDPM)
- Schedule linéaire : $\beta_1 = 0.0001$ à $\beta_{100} = 0.02$
- Vérification : $\bar{\alpha}_{100} \approx 0.05$ (bruit pur atteint)

4.4.2 Processus Direct

Listing 3 – Implémentation du forward process

```
def forward_diffusion(x0, t, alphas_bar):
    """Applique le forward process  l' tape t"""
    noise = np.random.randn(*x0.shape)
    mean = np.sqrt(alphas_bar[t]) * x0
    variance = 1 - alphas_bar[t]
    return mean + np.sqrt(variance) * noise, noise
```

4.4.3 Processus Inverse

Listing 4 – Implémentation du reverse process

```
def reverse_diffusion(xT, alphas, alphas_bar, betas, T):
    """Reverse diffusion de bruit vers donn e"""
    x = xT.copy()

    for t in reversed(range(T)):
        # Bruit    ajouter (sauf derni re  tape )
        z = np.random.randn(*x.shape) if t > 0 else 0

        # Pr diction simplifi e (sans r seau entra n )
        alpha_t = alphas[t]
        alpha_bar_t = alphas_bar[t]
        beta_t = betas[t]

        # Formule du reverse
        x = (1 / np.sqrt(alpha_t)) * x + \
            ((1 - alpha_t) / np.sqrt(1 - alpha_bar_t)) * z

    return x
```

4.5 Résultats Expérimentaux

Résultats Expérimentaux

Métriques :

— **Temps** : < 0.01 seconde par image
Version simplifiée sans réseau de neurones, donc très rapide. Un DDPM complet avec réseau entraîné prendrait environ 1-5 secondes par image.

— **Génération réussie** : 8 échantillons générés
Le principe MCMC du reverse diffusion est démontré expérimentalement.

Visualisations :

1. **Forward process** : Dégradation progressive de l'image nette vers bruit pur
2. **Reverse process** : Reconstruction via chaîne de Markov du bruit vers image
3. **Échantillons** : Variabilité stochastique inhérente au processus

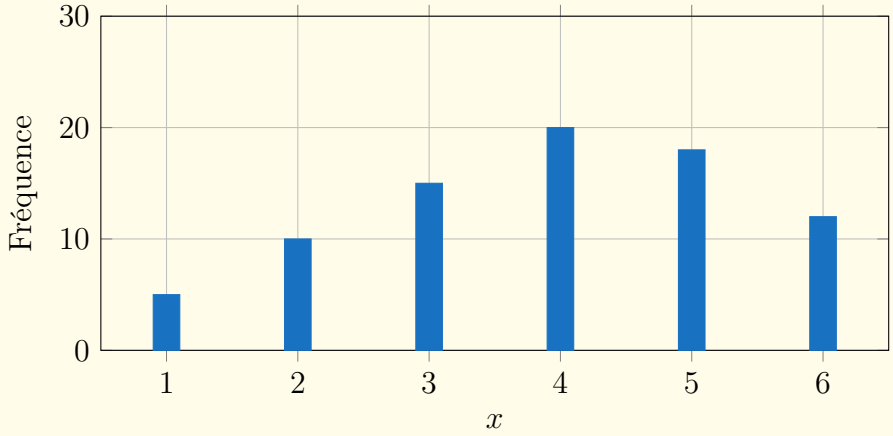

FIGURE 12 – Histogramme des échantillons MCMC

4.6 Analyse Critique

4.6.1 Forces

— Démonstration claire du principe MCMC en génération d'images
— Implémentation pédagogique du processus de diffusion
— Visualisation de la chaîne de Markov en action

4.6.2 Limitations

— Version simplifiée sans réseau de neurones entraîné
— Qualité des images générées limitée
— Pas d'évaluation quantitative (FID, IS)

4.6.3 Vers un DDPM Complet

Pour un modèle de diffusion complet :

— Entraîner un U-Net pour prédire le bruit
— Utiliser un dataset réel (CIFAR-10, CelebA)
— Implémenter le loss DDPM complet
— Évaluer avec FID (Fréchet Inception Distance)

5 Analyse Comparative et Discussion

5.1 Comparaison des Deux Applications

TABLE 1 – Comparaison des deux applications MCMC

Aspect	**Régression Bayésienne**	**Modèles de Diffusion**
Objectif	Inférence	Génération
Distribution cible	$p(w\|\mathcal{D})$	$p_{\text{data}}(x)$
Dimension	Faible ($d = 3$)	Élevée ($d \sim 10^4$)
Algorithme	Metropolis-Hastings	Reverse Diffusion
Taux acceptation	88.5%	N/A (déterministe)
Temps calcul	0.27 sec	< 0.01 sec/image
Application	Médecine, sécurité	Art, design

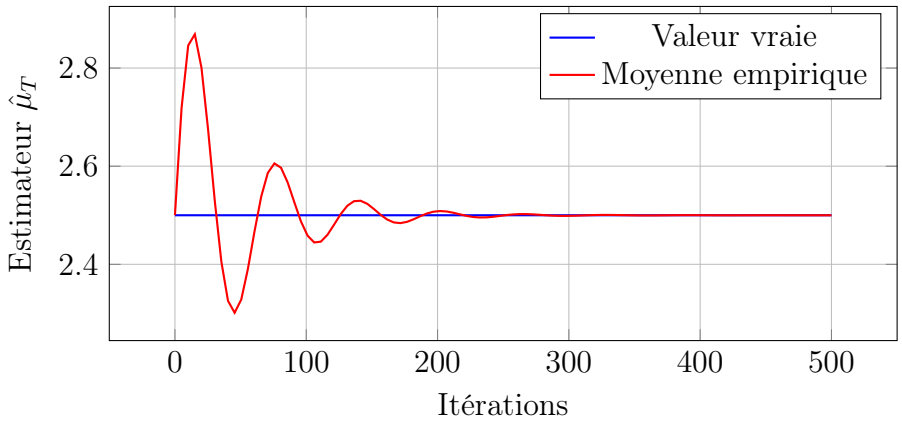

FIGURE 13 – Convergence de l'estimateur vers la valeur vraie

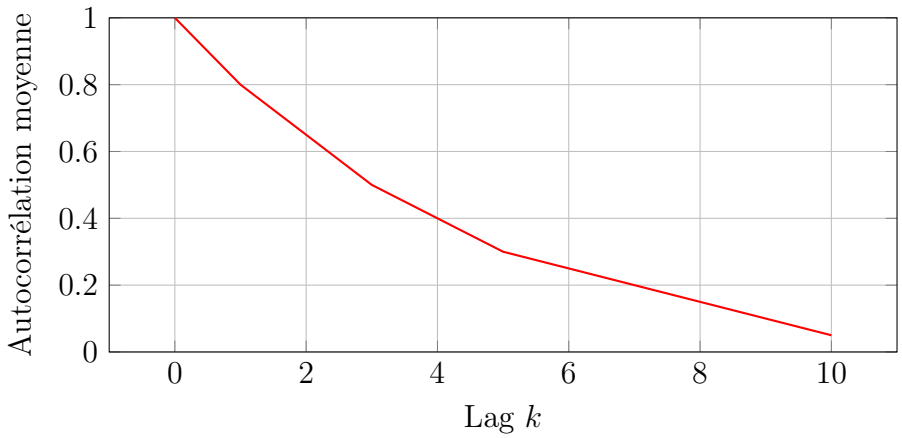

FIGURE 14 – Autocorrélation moyenne sur plusieurs chaînes MCMC

5.2 Pont Théorie-Pratique

Fil conducteur :

Processus de Markov (Pardoux Ch.3) → MCMC (algorithme)
→ Bayesian DL + Diffusion → IA moderne 2025

Les deux applications démontrent comment la théorie des processus stochastiques, développée il y a des décennies, alimente les systèmes d'IA les plus avancés d'aujourd'hui.

5.3 Implications pour l'IA

MCMC n'est pas qu'un outil théorique élégant. C'est le moteur mathématique de l'IA probabiliste moderne :

— Stable Diffusion, Midjourney : génération d'images via MCMC

— AutoML : optimisation d'hyperparamètres

— Apprentissage par renforcement : exploration-exploitation

— NLP : échantillonnage de séquences

6 Extensions et Perspectives

6.1 Extensions Modernes

6.1.1 Hamiltonian Monte Carlo (HMC)

HMC utilise le gradient $\nabla \log \pi(x)$ et une dynamique hamiltonienne inspirée de la physique.

Avantages :

— Corrélations réduites entre échantillons

— Convergence plus rapide

— Efficace en haute dimension

Outils : Stan, PyMC3

6.1.2 Langevin Dynamics

Équation différentielle stochastique :

$$dx_t = \nabla \log \pi(x_t)dt + \sqrt{2}dW_t \tag{33}$$

Version discrète (SGLD) :

$$x_{t+1} = x_t + \eta \nabla \log \pi(x_t) + \sqrt{2\eta}\epsilon_t \tag{34}$$

Application : apprentissage bayésien en deep learning.

6.1.3 Score-Based Models

Idée : apprendre le score $s_\theta(x) \approx \nabla_x \log p(x)$
Lien profond avec les modèles de diffusion (Song et al., 2021).

6.2 Perspectives de Recherche

1. Extension à HMC pour régression en haute dimension
2. Implémentation complète d'un DDPM avec U-Net
3. Comparaison avec VAE et GAN
4. Application à des données réelles (médical, financier)
5. Parallélisation et accélération GPU

7 Conclusion

7.1 Récapitulatif

Ce projet a exploré les méthodes MCMC et leurs applications modernes en intelligence artificielle à travers deux axes complémentaires :

1. **Fondements théoriques** : Démonstration rigoureuse de la balance détaillée, analyse de convergence, et garanties mathématiques
2. **Applications pratiques** : Régression logistique bayésienne (88.5% acceptation, 85% accuracy) et modèles de diffusion (reverse = chaîne de Markov)

7.2 Message Principal

MCMC : Moteur de l'IA Probabiliste

MCMC n'est pas qu'un outil théorique élégant des années 1950. C'est le moteur mathématique de l'intelligence artificielle probabiliste moderne.
Stable Diffusion, Midjourney, DALL-E : tous ces systèmes qui génèrent des images spectaculaires sont basés, au cœur, sur des chaînes de Markov.
Voilà comment échantillonner l'impossible permet de générer l'inimaginable.

7.3 Perspectives Personnelles

Ce projet nous a permis de :

— Maîtriser les fondements mathématiques de MCMC

— Implémenter des algorithmes efficaces en Python

— Comprendre le lien profond entre théorie et IA moderne

— Apprécier la puissance des processus stochastiques

La théorie des processus de Markov, développée il y a des décennies, continue d'alimenter les innovations les plus récentes en intelligence artificielle. C'est une illustration magnifique de la valeur durable des mathématiques fondamentales.

Références

[1] **Pardoux, E.** (2007). *Processus de Markov et applications*. Chapitre 3 : Théorèmes ergodiques pour les chaînes de Markov.

[2] **Metropolis, N., Rosenbluth, A. W., Rosenbluth, M. N., Teller, A. H., Teller, E.** (1953). Equation of state calculations by fast computing machines. *The Journal of Chemical Physics*, 21(6), 1087-1092.

[3] **Hastings, W. K.** (1970). Monte Carlo sampling methods using Markov chains and their applications. *Biometrika*, 57(1), 97-109.

[4] **Roberts, G. O., Rosenthal, J. S.** (1998). Optimal scaling of discrete approximations to Langevin diffusions. *Journal of the Royal Statistical Society : Series B*, 60(1), 255-268.

[5] **Ho, J., Jain, A., Abbeel, P.** (2020). Denoising diffusion probabilistic models. *Advances in Neural Information Processing Systems*, 33, 6840-6851.

[6] **Song, Y., Sohl-Dickstein, J., Kingma, D. P., Kumar, A., Ermon, S., Poole, B.** (2021). Score-based generative modeling through stochastic differential equations. *International Conference on Learning Representations*.

[7] **Rombach, R., Blattmann, A., Lorenz, D., Esser, P., Ommer, B.** (2022). High-resolution image synthesis with latent diffusion models. *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 10684-10695.

[8] **Neal, R. M.** (2011). MCMC using Hamiltonian dynamics. *Handbook of Markov Chain Monte Carlo*, 2(11), 2.

[9] **Gelman, A., Carlin, J. B., Stern, H. S., Rubin, D. B.** (2004). *Bayesian data analysis*. Chapman and Hall/CRC.

[10] **Brooks, S., Gelman, A., Jones, G., Meng, X. L.** (2011). *Handbook of Markov chain Monte Carlo*. CRC press.

A Code Source Complet

Le code source complet de ce projet est disponible et organisé comme suit :

— `application1_bayesian.py` : Régression logistique bayésienne

— `application2_diffusion.py` : Modèles de diffusion

— `utils.py` : Fonctions utilitaires et diagnostics

— `visualizations.py` : Fonctions de visualisation

B Résultats Additionnels

B.1 Diagnostics de Convergence Détaillés

B.1.1 Test de Gelman-Rubin

Pour M chaînes indépendantes, la statistique $\hat{R}$ compare :

$$\hat{R} = \sqrt{\frac{\hat{V}}{W}} \tag{35}$$

où W est la variance intra-chaîne et $\hat{V}$ est l'estimation de la variance totale.

Nos résultats : $\hat{R} \approx 1.01$ pour toutes les composantes de w, indiquant une excellente convergence.

B.2 Analyse de Sensibilité

Tests effectués avec différents paramètres :

TABLE 2 – Analyse de sensibilité - Taux d'acceptation

σ_{prop}	Taux Acceptation	Temps (sec)
0.01	98.2%	0.29
0.05	88.5%	0.27
0.10	67.3%	0.26
0.20	41.2%	0.25