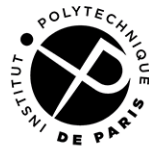

Projet 2 MACS205 - Weighted Schemes in AIS

Julien Béguinot, Mathis Fitoussi

6 juillet 2021



**INSTITUT
POLYTECHNIQUE
DE PARIS**

Table des matières

1 Introduction : méthodes de Monte Carlo et Adaptive Importance Sampling	3
1.1 Les méthodes de Monte-Carlo	3
1.2 Importance Sampling	3
1.3 L'adaptive Important Sampling	4
2 Notations	4
3 Poids optimaux	7
4 Propriétés de Martingales	7
5 Estimation des poids optimaux	9
6 Square root rule	10
7 Asymptotic Results for AIS and wAIS	11
7.1 Théorème Centrale Limite pour AIS	11
7.2 Théorème Centrale Limite pour le wAIS	13
7.3 Interpretation du théorème	16
8 Optimalité asymptotique pour l'AIS et le wAIS	17
8.1 Optimalité asymptotique de l'AIS	17
8.2 Optimalité asymptotique pour le wAIS	18
9 Résultats expérimentaux et conclusion	18
10 Annexe : Code	20

1 Introduction : méthodes de Monte Carlo et Adaptive Importance Sampling

1.1 Les méthodes de Monte-Carlo

On rappelle les résultats et le contexte générales des méthodes de **Monte-Carlo**. Ces méthodes sont utiles dans un large éventails de contextes scientifiques. En particulier c'est utile pour évaluer des intégrales de la forme $\int g f$ où f est une densité de probabilité.

Data: Nombre de particules n

1. On génère X_1, \dots, X_n de manière i.i.d selon f
2. On calcule

$$\hat{I}^{mc} = n^{-1} \sum_{i=1}^n g(X_i)$$

Result:

$$\hat{I}^{mc}$$

Algorithm 1: Algorithme de Monte Carlo

Ce genre d'estimateur est motivé par la loi des grands nombres et une variété de résultat en montre l'intérêt. En particulier pour des estimations d'intégrales dans des espaces de grandes dimensions.

1.2 Importance Sampling

Les méthodes de Monte-Carlo ont des limites. L'une d'elle est que l'on ne peut pas forcément générer des échantillons selon la loi f . On introduit alors une méthode appelée Importance Sampling. Si on considère une densité q pour laquelle on peut générer des échantillons dont le support contient celui de f alors on exploite la formule $\int f g = \int \frac{f g}{q} q$.

L'algorithme d'importance sampling est alors,

Data: Nombre de particules n . Un échantillonneur q .

1. On génère X_1, \dots, X_n de manière i.i.d selon q
2. On calcule

$$\hat{I}^{IS} = n^{-1} \sum_{i=1}^n \frac{g(X_i) f(X_i)}{q(X_i)}$$

Result:

$$\hat{I}^{IS}$$

Algorithm 2: Algorithme d'Importance Sampling

1.3 L'adaptive Important Sampling

Lorsqu'on utilise une méthode d'Import Sampling une question naturelle apparait. Comment bien choisir la densité q ? En effet, il apparait que plus q est proche de f plus l'estimateur sera efficace. On peut par exemple chercher à prendre q minimisant la divergence de Kullback-Leibler avec f . Mais cette minimisation induit également un calcul d'intégrale. On va donc améliorer notre choix de q au fur et à mesure de l'algorithme. C'est l'idée de l'**Adaptive Important Sampling**.

Cette remarque est notamment justifié par les lemmes techniques montré dans [3].

Lemme 1. *In the AIS context the optimal sampler $q \propto |g|$*

Lemme 2. *In the bayesian AIS context the optimal sampler $q \propto |g - \int g f| f$*

On part donc d'une densité q_0 puis au bout de n_0 itérations on choisit q_1 puis après n_1 itérations on détermine q_2 et ainsi de suite. la suite $(q_n)_{n \in \mathbb{N}}$. Cet algorithme est donc caractérisé par la suite $(n_t)_{t \in \mathbb{N}}$, la taille des **batch** et par la suite de densité $(q_n)_{n \in \mathbb{N}}$, la **politique**.

Pour un nombre de particules $N = \sum_{t=1}^T n_t$ fixé on doit donc considérer un compromis entre un grand nombre T de **mini-batch** ou un plus petit nombre de grand batch.

Une autre idée est la suivante. Pourquoi accorder la même importance à chaque batch alors que le choix de q est censé s'améliorer au cours des batchs? Ainsi, il faudrait mieux favoriser les derniers batchs et bien moins les premiers. C'est dans cette esprit qu'on introduit une méthode pondérés de l'Adaptive Importance Sampling, le **weigthed Adaptive Importance Sampling**.

Une nouvelle question se pose alors. Comment choisir au mieux les poids de cette méthode pondérée? C'est la question qui va nous intéresser dans la suite de ce rapport. Notamment à travers l'analyse des résultats de [3] (Portier et al.) et de [1] (Owen et al.). En particulier, on étends un résultats de [3] au cas pondérés.

2 Notations

On cherche à estimer via une méthode de Monte Carlo : $I = \int g f d\lambda$ où $g : \mathbb{R} \rightarrow \mathbb{R}$ et f est une densité.

A chaque étape $t, t \in \{1, \dots, T\}$ de l'algorithme d'échantillonnage préférentiel adaptatif, on utilise la politique q_{t-1} pour générer n_t variables.

On note les variables $X_{n_{t-1}+1}, \dots, X_{n_t}$, où $N = \sum_{i=1}^T n_i$.

L'estimation de l'étape t s'écrit donc :

$$\hat{I}_t = \frac{1}{t} \sum_{i=1}^t \frac{f g}{q_{t-1}}(X_i)$$

Dans le cadre bayésien qui nous intéresse, nous n'avons pas accès à la densité f . En effet nous obtenons f grace à la règle de Bayes donc à normalisation près. Ainsi, en général, nous avons accès à une version non normalisée de f , f_u tel que :

$$f = \frac{f_u}{\int f_u}$$

L'estimation de l'étape t s'écrit alors :

$$\hat{I}_t^{(norm)} = \sum_{i=1}^t \frac{f_u g}{q_{t-1}}(X_i) / \sum_{i=1}^t \frac{f_u}{q_{t-1}}(X_i)$$

En particulier,

$$\hat{I}_t^{(norm)} = \frac{\hat{I}_t(g f_u)}{\hat{I}_t(f_u)}$$

On cherche dans notre cas une pondération de ces estimations, qui s'écrit :

$$\hat{I}_\alpha^T = \sum_{t=1}^T \alpha_{T,t} \hat{I}_t$$

Si c'est un barycentre i.e. $\sum_{t=1}^T \alpha_{T,t} = 1$ alors on remarque notamment que c'est encore un estimateur non biaisé de I .

Ou encore dans le cas bayésien,

$$\hat{I}_\alpha^{T,(norm)} = \sum_{t=1}^T \alpha_{T,t} \hat{I}_t^{(norm)}$$

Ajoutons les notations suivantes qui décrivent l'algorithme d'échantillonnage préférentiel adaptatif :

$$\Delta_t = \frac{f g}{q_{t-1}}(X_t) - I$$

$$M_n = \sum_{t=1}^n \Delta_j$$

Autres notations :

$$\text{Var}(\hat{I}_t) = \sigma_t^2$$

Data: Le nombre T d'étape de l'AIS, la politique $(n_t)_{t=1,\dots,T}$ et une distribution initiale q_0

$S_0 = 0$ $N_0 = 0$

for $t=1,\dots,T$ **do**

/* Phase d'exploration */

On tire $(X_{t,1}, \dots, X_{t,n_t})$ selon q_{t-1}

/* Phase d'exploitation */

/* Mise à jour de l'estimateur */

$$S_t = S_{t-1} + \sum_{i=1}^{n_t} \frac{f(X_{t,i})g(X_{t,i})}{q_{t-1}(X_{t,i})}$$

$$N_t = N_{t-1} + n_t$$

$$I_t = N_t^{-1} S_t$$

/* Mise à jour de l'échantillonneur q_t */

On calcul q_t comme l'élément minimisant la divergence de Kullback-Leibler avec f . Cette divergence de KL étant estimé grâce à q_{t-1}

end

Result:

Algorithm 3: AIS à l'échelle de la politique

Data: Le nombre T d'étape de l'AIS, la politique $(n_t)_{t=1,\dots,T}$ et une distribution initiale q_0
for $j=1,\dots,N$ **do**

/* Phase d'exploration */
 On tire x_j selon q_{j-1}
 /* Phase d'exploitation */
 /* Mise à jour de l'estimateur */

$$S_j = S_{j-1} + \frac{f(X_j)g(X_j)}{q_{j-1}(X_j)}$$

$$I_j = j^{-1}S_j$$

/* Mise à jour de l'échantillonneur q_j */
 Si $j \in \{N_t = \sum_{s=1}^t n_s | t \geq 1\}$ on met à jour l'échantillonneur.

end

Result:

Algorithm 5: AIS à l'échelle d'un tirage

3 Poids optimaux

Le problème peut être vu comme celui de la minimisation de la variance de \hat{I} sous contrainte.

$$\text{Var}(\hat{I}) = \sum_{t=1}^T \alpha_t^2 \sigma_t^2$$

On peut donc décrire le Lagrangien :

$$L((\alpha_i)_i, \beta) = \text{Var}(\hat{I}) - \beta \left(\sum_{t=1}^T \alpha_t - 1 \right) = \sum_{t=1}^T \alpha_t^2 \sigma_t^2 - \beta \left(\sum_{t=1}^T \alpha_t - 1 \right)$$

Soit $t \in \{1, \dots, T\}$, dérivons cette expression par rapport à α_t et annulons la dérivée :

$$\begin{aligned} \frac{\partial L}{\partial \alpha_t}((\alpha_i)_i, \beta) &= 0 = 2\alpha_t \sigma_t^2 - \beta \\ \implies \forall t \in \{1, \dots, T\}, \alpha_t &\propto \sigma_t^{-2} \end{aligned}$$

4 Propriétés de Martingales

Dans cette section on retrouve les résultats donnés dans [3].

Montrons que (M_n) est une martingale pour la filtration $F_n = \sigma(X_1, \dots, X_n)$:

$$\begin{aligned} \mathbb{E}(M_n | F_{n-1}) &= \sum_{j=1}^n \mathbb{E}(\Delta_j | F_{n-1}) \\ \mathbb{E}(M_n | F_{n-1}) &= \sum_{j=1}^{n-1} \mathbb{E}(\Delta_j | F_{n-1}) + \mathbb{E}(\Delta_n | F_{n-1}) \\ \mathbb{E}(M_n | F_{n-1}) &= M_{n-1} + \mathbb{E}\left(\frac{fg}{q_{n-1}}(X_n) - I | X_1, \dots, X_{n-1}\right) \end{aligned}$$

On sait que conditionnellement à X_1, \dots, X_{n-1} , X_n suit la loi q_{n-1} , et donc :

$$\mathbb{E}(M_n | F_{n-1}) = M_{n-1} + \underbrace{\int \frac{fg}{q_{n-1}}(x) \times q_{n-1}(x) dx - I}_{=I} = M_{n-1}$$

Calculons l'erreur quadratique moyenne commise à chaque étape. Remarquons d'abord l'identité suivante :

$$\hat{I}_t = \frac{1}{n_t} \sum_{i=1}^{n_t} \frac{fg}{q_{t-1}}(X_i) = \frac{1}{n_t} \sum_{i=1}^{n_t} (\Delta_i + I) \implies \hat{I}_t - I = \frac{M_n}{n}$$

Ainsi,

$$EQM(t) = \mathbb{E}\left[(\hat{I}_t - I)^2\right] = \frac{1}{n^2} \mathbb{E}(M_n^2)$$

Or on sait que $\mathbb{E}(M_n^2)$ se calcule avec la partie prédictible de la décomposition de Doob de M_n^2 . Procédons à son calcul :

$$\langle M \rangle_n = \sum_{j=1}^n \mathbb{E}(\Delta M_j^2 | F_{j-1})$$

$$\langle M \rangle_n = \sum_{j=1}^n \mathbb{E}(\Delta_j^2 | F_{j-1})$$

$$\langle M \rangle_n = \sum_{j=1}^n \mathbb{E} \left(\left(\frac{f^2 g^2}{q_{j-1}^2}(X_j) + I^2 - 2I \frac{fg}{q_{j-1}}(X_j) \right) | F_{j-1} \right)$$

On sait que conditionnellement à X_1, \dots, X_{n-1} , X_n suit la loi q_{n-1} , et donc $\mathbb{E} \left(\frac{fg}{q_{j-1}}(X_j) | F_{j-1} \right) = I$. Ainsi,

$$\langle M \rangle_n = -nI^2 + \sum_{j=1}^n \mathbb{E} \left(\left(\frac{f^2 g^2}{q_{j-1}^2}(X_j) \right) | F_{j-1} \right)$$

De même,

$$\langle M \rangle_n = -nI^2 + n \int \frac{f^2 g^2}{q_{j-1}}$$

Enfin :

$$EQM(t) = \frac{1}{n^2} \mathbb{E}(M_n^2) = \frac{1}{n^2} \mathbb{E}(\langle M \rangle_n) = \frac{1}{n} \left(\mathbb{E} \left(\int \frac{f^2 g^2}{q_{j-1}} \right) - I^2 \right) = \frac{1}{n} \left(\frac{1}{n} \sum_{j=1}^n \mathbb{E} \left(\frac{f^2 g^2}{q_{j-1}^2}(X_j) \right) - I^2 \right)$$

En posant :

$$V(q, \phi) = \int \frac{(\phi - q f \phi)^2}{q}$$

On remarque que l'on peut également écrire :

$$EQM(t) = \frac{1}{n^2} \sum_{j=1}^n \mathbb{E}(V(q_{j-1}, fg))$$

5 Estimation des poids optimaux

Reprenons les notations de l'algorithme 1.

On cherche à estimer les poids $\alpha_t \propto \sigma_t^{-2}$ et il faut donc parvenir à estimer les σ_t . On peut pour cela remarquer que $\sigma_t^2 = \text{Var}(\hat{I}_t)$.

$$\text{Var}(\hat{I}_t) \propto \sum_{i=1}^{n_t} \text{Var}\left(\frac{fg}{q_{i-1}}(X_{t,i})\right)$$

Or :

$$\begin{aligned} \text{Var}\left(\frac{fg}{q_{i-1}}(X_{t,i})\right) &= \mathbb{E}\left[\left(\frac{fg}{q_{i-1}}(X_{t,i}) - \mathbb{E}\left(\frac{fg}{q_{i-1}}(X_{t,i})\right)\right)^2\right] \\ \text{Var}\left(\frac{fg}{q_{i-1}}(X_{t,i})\right) &= \mathbb{E}\left[\left(\frac{fg}{q_{i-1}}(X_{t,i}) - \underbrace{\mathbb{E}\left(\frac{fg}{q_{i-1}}(X_{t,i}) \mid F_{i-1}\right)}_{=I}\right)^2\right] \\ \text{Var}\left(\frac{fg}{q_{i-1}}(X_{t,i})\right) &= \mathbb{E}\left[\left(\frac{fg}{q_{i-1}}(X_{t,i}) - I\right)^2\right] \end{aligned}$$

On obtient donc :

$$\text{Var}(\hat{I}_t) \propto \sum_{i=1}^{n_t} \mathbb{E}\left[\left(\frac{fg}{q_{i-1}}(X_{t,i}) - I\right)^2\right]$$

Et on aimerait proposer comme estimateur :

$$\hat{\sigma}_t^2 \propto \sum_{i=1}^{n_t} \left(\frac{fg}{q_{i-1}}(X_{t,i}) - I\right)^2$$

Remarquons tout d'abord que $\mathbb{E}(\hat{\sigma}_t^2) = \sigma_t^2$ donc l'estimateur est non-biaisé.

Cet estimateur a deux problèmes essentiels :

- On ne connaît évidemment pas la valeur exacte de I puisqu'on cherche à l'estimer.
- La méthode est dépendante de la fonction g

Néanmoins, on peut remarquer que si on veut être équitablement "efficace" sur tout l'espace d'intégration, une bonne méthode consiste à prendre g la fonction constante. Ainsi, $I = \int fg = g(0) \int f = g(0)$ et l'estimateur précédent se simplifie en :

$$\hat{\sigma}_t^2 \propto \sum_{i=1}^{n_t} \left(\frac{f}{q_{i-1}}(X_{t,i}) - 1\right)^2$$

En ajoutant la condition sur les poids $\sum_{i=1}^{n_t} \frac{1}{\hat{\sigma}_i^2} = 1$, on a un estimateur à chaque étape.

6 Square root rule

Supposons maintenant que l'algorithme de l'AIS progresse au rythme suivant : $\text{Var}(\hat{I}_k) = \sigma^2 k^{-y}$ pour un certain $y \in (0, 1)$.

A quoi correspond cette hypothèse ?

- On suppose ici que l'algorithme permet un gain constant, c'est à dire qu'il n'est jamais excessivement plus performant à une étape qu'à une autre.
- Le cas $y = 0$ est le cas le plus pessimiste possible : il correspond à une progression nulle de l'algorithme en termes de qualité du résultat.
- Le cas $y = 1$ (ou même $y > 1$) est trop optimiste car il indiquerait une convergence bien plus rapide que ce qui est connu actuellement.

Notre estimation \hat{I} dépend donc maintenant de la valeur que l'on choisit pour y :

$$\hat{I} = \hat{I}(y) = \frac{1}{\sum_{t=1}^T t^y} \sum_{t=1}^T t^y \hat{I}_t$$

Comme à l'accoutumée, on va donc essayer d'en minimiser la variance, cette fois en fonction de y , dans le but de montrer que prendre $y = \frac{1}{2}$ est un bon compromis.

On note :

$$\rho_T(y|x) = \frac{\text{Var}(\hat{I}(x))}{\text{Var}(\hat{I}(y))}$$

Cette valeur reflète l'inefficacité de prendre x par rapport à un certain y .

Les résultats principaux de [1] sont alors :

Proposition 1.

$$\sup_{1 \leq T < \infty} \sup_{0 \leq x \leq 1} \rho_T\left(\frac{1}{2}|x\right) \leq \frac{9}{8}$$

$$\forall K \geq 2, y \neq \frac{1}{2}, \sup_{0 \leq x \leq 1} \rho_T(y|x) > \sup_{0 \leq x \leq 1} \rho_T\left(\frac{1}{2}|x\right)$$

- La première inégalité signifie qu'en prenant $y = \frac{1}{2}$, la variance ne peut être augmentée que d'un facteur $\frac{9}{8}$.
- La seconde signifie que $y = \frac{1}{2}$ est l'unique meilleur pari si on ne connaît pas x .

7 Asymptotic Results for AIS and wAIS

On commence par rappeler les résultats de [3].

7.1 Théorème Centrale Limite pour AIS

Lemme 3. $\langle M \rangle_n = \sum_{j=1}^n V(q_{j-1}, fg)$ où $V(q, h) = \int \frac{\|h(x) - q(x)fh\|_2^2}{q(x)} dx$

Démonstration. We have $\langle M \rangle_n = \sum_{j=1}^n \mathbb{E}[\|\Delta_j\|_2^2 | \mathcal{F}_{j-1}] = \sum_{j=1}^n V(q_{j-1}, fg)$ par indépendance conditionnel. □

Theorème 1. (Théorème centrale limite pour l'AIS)

Supposons que :

—

$$V(q_n, fg) \rightarrow V_* \geq 0, p.s.$$

— $\exists \eta > 0$ tel que

$$\sup_{j \in \mathbb{N}} \int \frac{\|fg\|^{2+\eta}}{q_j^{1+\eta}} < \infty p.s.$$

Alors,

$$n^{\frac{1}{2}}(I_n - \int fg) \Rightarrow \mathcal{N}(0, V_*)$$

On reprend la preuve fournie dans le matériel complémentaire de [3].

Démonstration. On commence par se ramener au cas scalaire. En effet d'après le lemme de Cramer-Wald il suffit de montrer que :

$$\forall \gamma \in \mathbb{R}^p, \langle \gamma, n(I_n - fg) \rangle \Rightarrow \mathcal{N}(0, \gamma^T V_* \gamma).$$

On peut donc supposer dans la suite que fg est une fonction à valeur réel.

Or, on a que

$$\sqrt{n}(I_n - \int fg) = n^{-\frac{1}{2}} M_n$$

Par théorème (Corollaire 3.1 p. 58, [2]), en posant $X_{nj} = \frac{1}{\sqrt{n}} \Delta_j$ il faut montrer que :

1. $\sum_{j=1}^n \mathbb{E}[X_{nj}^2 | \mathcal{F}_{j-1}] \rightarrow V_*$ en probabilité
2. $\forall \epsilon > 0, \sum_{j=1}^n \mathbb{E}[X_{nj}^2 \mathbb{1}_{|X_{nj}| > \epsilon} | \mathcal{F}_{j-1}] \rightarrow 0$ en probabilité (condition de Lindeberg)

On commence par démontrer le premier point.

On a,

$$\begin{aligned} \sum_{j=1}^n \mathbb{E}[X_{nj}^2 | \mathcal{F}_{j-1}] &= n^{-1} \langle M \rangle_n \\ &= V_* + \frac{1}{n} \sum_{j=1}^n [V(q_{j-1}, fg) - V_*] \end{aligned}$$

Or on a supposé que $V(q_n, fg) \rightarrow V_* \geq 0, p.s.$ Donc d'après le lemme de Cesaro,

$$\sum_{j=1}^n \mathbb{E}[X_{nj}^2 | \mathcal{F}_{j-1}] \rightarrow 0$$

presque surement. Et donc en particulier en probabilité.

Il reste a montrer le second point. Or, par définition de Δ_j

$$\sum_{j=1}^n \mathbb{E}[X_{nj}^2 \mathbb{1}_{|X_{nj}| > \epsilon} | \mathcal{F}_{j-1}] = \frac{1}{n} \sum_{j=1}^n \mathbb{E}[\Delta_j^2 \mathbb{1}_{|\Delta_j| > \sqrt{n}\epsilon} | \mathcal{F}_{j-1}]$$

Notons, $w_{j-1} = \frac{fg}{q_{j-1}}$

$$\sum_{j=1}^n \mathbb{E}[X_{nj}^2 \mathbb{1}_{|X_{nj}| > \epsilon} | \mathcal{F}_{j-1}] = \int (w_{j-1}(x) - I)^2 q_{j-1}(x) \mathbb{1}_{|w_{j-1}(x) - I| > \sqrt{n}\epsilon} dx$$

Par inégalité triangulaire, puis positivité,

$$\leq \int (w_{j-1}(x) - I)^2 q_{j-1}(x) \mathbb{1}_{|w_{j-1}(x)| > \sqrt{n}\epsilon - |I|} dx$$

Par inégalité arithmético-géométrique (ou convexité de $x \mapsto x^2$)

$$\begin{aligned} &\leq \int 2(w_{j-1}(x)^2 + I^2) q_{j-1}(x) \mathbb{1}_{|w_{j-1}(x)| > \sqrt{n}\epsilon - |I|} dx \\ &\leq 2 \int w_{j-1}(x)^2 q_{j-1}(x) \mathbb{1}_{|w_{j-1}(x)| > \sqrt{n}\epsilon - |I|} dx + 2I^2 \int q_{j-1}(x) \mathbb{1}_{|w_{j-1}(x)| > \sqrt{n}\epsilon - |I|} dx \end{aligned}$$

Soit $\eta > 0$ alors d'après l'inégalité de Markov,

$$\leq \frac{2}{(\epsilon\sqrt{n} - |I|)^\eta} \int \frac{|fg|^{2+\eta}}{q_{j-1}(x)^{1+\eta}} dx + \frac{2I^2}{\epsilon\sqrt{n} - |I|} \int \left| \frac{fg}{q_{j-1}} \right| q_{j-1} dx$$

Finalement,

$$\leq \frac{2}{(\epsilon\sqrt{n} - |I|)^\eta} \sup_j \int \frac{|fg|^{2+\eta}}{q_{j-1}(x)^{1+\eta}} dx + \frac{2I^2}{\epsilon\sqrt{n} - |I|} \int \left| \frac{fg}{q_{j-1}} \right| q_{j-1} dx$$

Ainsi, on a par hypothèse la converge presque sûr vers 0 et donc en particulier la convergence en probabilité. \square

Corollaire 4. (TCL pour l'AIS normalisé) Supposons que les hypothèses du théorème précédent soit vérifié pour $(f_u g, f_u)^\perp$ au lieu de fg alors :

$$\sqrt{n}(I_n^{(norm)} - \int fg) \Rightarrow \mathcal{N}(0, UV_* U^\perp)$$

avec $U = (\mathcal{I}, -\int f_u g)$

Démonstration. On applique le théorème précédent au vecteur. Le corollaire est alors vraie d'après le lemme de Slutsky. \square

7.2 Théorème Centrale Limite pour le wAIS

On va adapter l'argument de la martingale pour prouver notre théorème.

Posons,

$$M_n^\alpha = \sum_{t=1}^n \alpha_t \Delta_t$$

Avec

$$\Delta_j = \frac{f(X_j)g(X_j)}{q_{j-1}(X_j)} - I$$

Dans ce contexte, on a alors un estimateur pondéré par la suite α ,

$$I_n^\alpha = \frac{1}{\sum_{t=1}^n \alpha_t} \sum_{t=1}^n \alpha_t \frac{f(X_t)g(X_t)}{q_{t-1}(X_t)}$$

Lemme 5. M_n^α est une \mathcal{F}_n -martingale.

Démonstration. Soit $n \in \mathbb{N}^*$ alors,

Comme M_{n-1}^α est \mathcal{F}_{n-1} mesurable,

$$\mathbb{E}[M_n^\alpha | \mathcal{F}_{n-1}] = M_{n-1}^\alpha + \mathbb{E}[\alpha_n \Delta_n | \mathcal{F}_{n-1}]$$

Comme Δ_n est indépendant de \mathcal{F}_{n-1} ,

$$= M_{n-1}^\alpha + \alpha_n \mathbb{E}[\Delta_n]$$

$$= M_{n-1}^\alpha$$

□

Lemme 6. $\langle M^\alpha \rangle_n = \sum_{j=1}^n \alpha_j^2 V(q_{j-1}, fg)$ où $V(q, h) = \int \frac{\|h(x) - q(x)fh\|_2^2}{q(x)} dx$

Démonstration. $\langle M^\alpha \rangle_n = \sum_{j=1}^n \mathbb{E}[(\alpha_j \Delta_j)(\alpha_j \Delta_j)^\perp | \mathcal{F}_{j-1}] = \sum_{j=1}^n \alpha_j^2 \mathbb{E}[\|\Delta_j\|_2^2 | \mathcal{F}_{j-1}] = \sum_{j=1}^n \alpha_j^2 V(q_{j-1}, fg)$

□

Remarque 1. Notons que pour les α_j constant de valeur 1 on retrouve bien le lemme de la version non pondérés.

Theorem 2. (TCL pour le wAIS) Supposons que :

$$\sup_{n \in \mathbb{N}^*, 1 \leq j \leq n} n \alpha_j / \sum_{t=1}^n \alpha_t < \infty$$

$$V(q_n, fg) \rightarrow V_* \geq 0, p.s.$$

— $\exists \eta > 0$ tel que

$$\sup_{j \in \mathbb{N}} \int \frac{\|fg\|^{2+\eta}}{q_j^{1+\eta}} < \infty p.s.$$

Alors,

$$n^{\frac{1}{2}}(I_n^\alpha - \int fg) \Rightarrow \mathcal{N}(0, V_*)$$

Remarque 2. A noter que dans l'hypothèse, le α_j ne peut pas être retiré si on ne fait pas d'hypothèse supplémentaire sur la suite des poids. Cependant, si on suppose par exemple que les poids sont tous positifs, ce qui paraît raisonnable dans notre contexte alors elle se réécrit $\sup_{n \in \mathbb{N}^*} n \alpha_n / \sum_{t=1}^n \alpha_t < \infty$

Démonstration. On commence par remarquer que,

$$\sqrt{n}(I_n^\alpha - I) = \frac{\sqrt{n}}{\sum_{t=1}^n \alpha_t} M_n^\alpha$$

$$\text{Posons } X_{nj} = \frac{\sqrt{n}}{\sum_{t=1}^n \alpha_t} \alpha_j \Delta_j$$

On cherche à appliquer le corollaire 3 page 58 de [2]. Pour conclure il suffit alors de montrer que :

1. $\sum_{j=1}^n \mathbb{E}[X_{nj}^2 | \mathcal{F}_{j-1}] \rightarrow V_*$ en probabilité
2. $\forall \epsilon > 0, \sum_{j=1}^n \mathbb{E}[X_{nj}^2 \mathbb{1}_{|X_{nj}| > \epsilon} | \mathcal{F}_{j-1}] \rightarrow 0$ en probabilité (condition de Lindeberg)

On commence par démontrer le premier point.

On a,

$$\sum_{j=1}^n \mathbb{E}[X_{nj}^2 | \mathcal{F}_{j-1}] = \frac{n}{(\sum_{t=1}^n \alpha_t)^2} \langle M^\alpha \rangle_n$$

D'après le lemme ci-dessus.

$$\begin{aligned} &= V_* + \frac{n}{(\sum_{t=1}^n \alpha_t)^2} \sum_{j=1}^n \alpha_j^2 [V(q_{j-1}, fg) - V_*] \\ &= V_* + \frac{1}{n} \sum_{j=1}^n \left(\frac{n \alpha_j}{\sum_{t=1}^n \alpha_t} \right)^2 [V(q_{j-1}, fg) - V_*] \end{aligned}$$

Or on a supposé que $V(q_n, fg) \rightarrow V_* \geq 0, p.s.$ et que $\frac{n\alpha_j}{(\sum_{t=1}^n \alpha_t)}$ était bornée disons majoré par une constante B. Donc d'après le lemme de Cesaro,

$$\sum_{j=1}^n \mathbb{E}[X_{nj}^2 | \mathcal{F}_{j-1}] \rightarrow 0$$

presque surement. Et donc en particulier en probabilité.

Il reste a montrer le second point. Or, par définition de Δ_j on a,

$$\begin{aligned} \sum_{j=1}^n \mathbb{E}[X_{nj}^2 \mathbb{1}_{|X_{nj}| > \epsilon} | \mathcal{F}_{j-1}] &= \sum_{j=1}^n \mathbb{E}\left[\frac{n\alpha_j^2}{(\sum \alpha_t)^2} \Delta_j^2 \mathbb{1}_{|\Delta_j| > (\frac{\sum \alpha_t}{n\alpha_j})\sqrt{n\epsilon}} | \mathcal{F}_{j-1}\right] \\ &\leq B^2 \sum_{j=1}^n \mathbb{E}[\Delta_j^2 \mathbb{1}_{|\Delta_j| > \sqrt{n}B\epsilon} | \mathcal{F}_{j-1}] \end{aligned}$$

Or dans la démonstration en l'absence de ponderation on a montré sous les même hypothèses que :

$$\forall \epsilon > 0, \sum_{j=1}^n \mathbb{E}[\Delta_j^2 \mathbb{1}_{|\Delta_j| > \sqrt{n}\epsilon} | \mathcal{F}_{j-1}] \rightarrow 0 p.s.$$

On l'applique dans notre cas avec ϵ remplacé par $B\epsilon$ et on obtient également le résultat voulu. □

Dans le contexte bayésien, de la même façon que pour l'AIS classique on considère un estimateur normalisé.

$$\hat{f}_n^{\alpha, (norm)} = \frac{\sum_{t=1}^n \alpha_t \frac{g(X_t) f_u(X_t)}{q_{t-1}(X_t)}}{\sum_{t=1}^n \alpha_t \frac{f_u(X_t)}{q_{t-1}(X_t)}} = \frac{I_n^{\alpha, (norm)}(g f_u)}{I_n^{\alpha, (norm)}(f_u)}$$

Corollaire 7. (TCL pour le wAIS normalisé) Supposons que les hypothèses du théorème précédent soit vérifiées pour $(f_u g, f_u)^\perp$ au lieu de $f g$ alors :

$$\sqrt{n} \left(I_n^{(norm), \alpha} - \int f g \right) \Rightarrow \mathcal{N}(0, UV_* U^\perp)$$

avec $U = (\mathcal{I}, - \int f_u g)$

Démonstration. On applique le théorème précédent au vecteur. Le corollaire est alors vraie d'après le lemme de Slutsky. \square

7.3 Interpretation du théorème

Commençons par regarder les hypothèses de notre théorème. Les deux dernières hypothèses sont les mêmes que pour le théorème de convergence de l'AIS de [3]. La première est une hypothèse sur la suite qui est raisonnable. En effet, cette hypothèse est par exemple vérifiée pour des suites bornées $0 < \alpha_- < \alpha_t < \alpha_+$, des suites en progressions arithmétique et géométriques. En particulier elle est vérifiée pour la "square root rule" de [1]. Par contre, l'hypothèse n'est pas vérifiée pour des suites du type $\alpha_t = \frac{1}{t}$, c'est compréhensible car alors dans ce cas on donne du poids au premiers batchs qui sont les moins bons et on ne bénéficie pas des derniers.

En ce qui concerne le résultat du théorème, il nous donne la normalité asymptotique de notre estimateur avec le même V_* que dans le théorème de l'AIS. Ce résultat est donc plutôt pessimiste, en effet, il montre qu'asymptotiquement les méthodes pondérées du type wAIS ne sont pas meilleurs que la méthode AIS. Heuristiquement on comprend bien ce résultat, en effet, une fois que l'échantillonneur a convergé, il ne bouge plus et donc tous les batchs deviennent équivalents. En particulier pour le cas de [3] les variances n'ont pas de raisons de beaucoup varier d'une étape à une autre. En conclusion, les méthodes pondérées ont un intérêt pour un nombre fini de particules mais pas dans le régime asymptotique.

8 Optimalité asymptotique pour l'AIS et le wAIS

8.1 Optimalité asymptotique de l'AIS

On se place dans le contexte où la politique q_n provient d'un modèle paramétrique $(q_\theta)_{\theta \in \Theta}$ avec Θ un compact de \mathbb{R}^n . On se place dans le cadre de la M-estimation.

Alors on rappelle les théorèmes de [3]

Théorème 3. (Consistance de la politique) Soit $M(x) = \sup_{\theta \in \Theta} m_\theta(x)$. On suppose que Θ est un compact de \mathbb{R}^n et que :

1.

$$\int M < \infty$$

2.

$$\sup_{\theta \in \Theta} \int \frac{M^2}{q_\theta} < \infty$$

3.

$$\forall \theta \neq \theta^*, \int m_\theta > \int m_{\theta^*}$$

4. L'application $\theta \mapsto m_\theta(x)$ est continue.

Alors, $\theta_n \rightarrow \theta^*$ p.s.

Démonstration. Dans le matériel supplémentaire de [3] □

Théorème 4. (Optimalité asymptotique de l'AIS)

Supposons que :

— Les hypothèses du théorème de consistance de la politique sont vérifiées.

— $\exists \eta > 0$ tel que

$$\sup_{\theta \in \Theta} \int \frac{\|fg\|^{2+\eta}}{q_j^{1+\eta}} < \infty \text{ p.s.}$$

Alors,

$$n^{\frac{1}{2}}(I_n - \int fg) \Rightarrow \mathcal{N}(0, V(q_{\theta^*}, fg))$$

Corollaire 8. (Optimalité asymptotique de l'AIS normalisé)

Supposons que les hypothèses du théorème précédent soit vérifiées pour $(f_u g, f_u)^\perp$ au lieu de fg alors :

$$\sqrt{n}(I_n^{(norm), \alpha} - \int fg) \Rightarrow \mathcal{N}(0, UV(q_{\theta^*}, fg)U^\perp)$$

avec $U = (\mathcal{I}, - \int f_u g)$

8.2 Optimalité asymptotique pour le wAIS

Theorème 5. (Optimalité asymptotique de le wAIS)

Supposons que :

$$\sup_{n \in \mathbb{N}^*, 1 \leq j \leq n} n \alpha_j / \sum_{t=1}^n \alpha_t < \infty$$

— Les hypothèse du théorème de consistance de la politique sont vérifiées.

— $\exists \eta > 0$ tel que

$$\sup_{\theta \in \Theta} \int \frac{\|fg\|^{2+\eta}}{q_j^{1+\eta}} < \infty p.s.$$

Alors,

$$n^{\frac{1}{2}} (I_n - \int fg) \Rightarrow \mathcal{N}(0, V(q_{\theta_*}, fg))$$

9 Résultats expérimentaux et conclusion

Nos expériences montrent l'intérêt des méthodes pondérées dans l'accélération de la convergence des méthodes d'importance sampling.

Pour les figures suivantes, on se fixe un budget en terme de nombre total de variables tirées pour estimer une intégrale. On fixe $\sum_{t=1}^T n_t = 2^{10}$.

On considère de plus une politique de la forme $n_t = n, \forall t$, et donc $nT = 2^{10}$.

Dans les figures suivantes, on a en abscisse n .

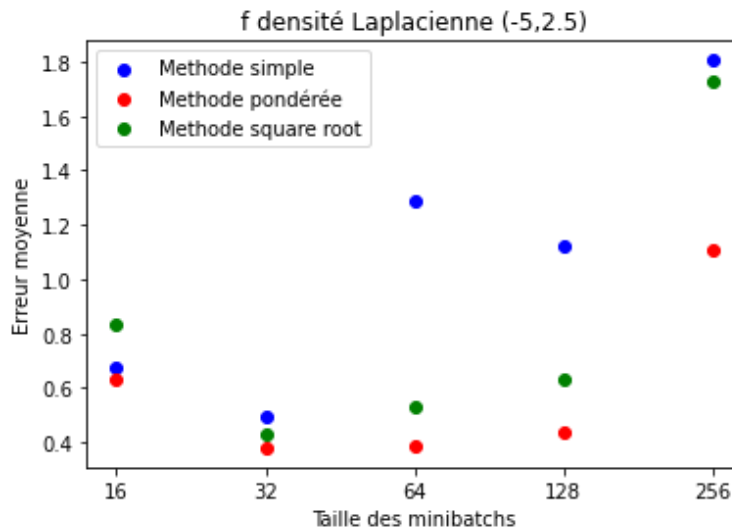


FIGURE 1 – Erreur pour $g : x \mapsto x$ et f une densité Laplacienne de paramètres $(-5, 2.5)$

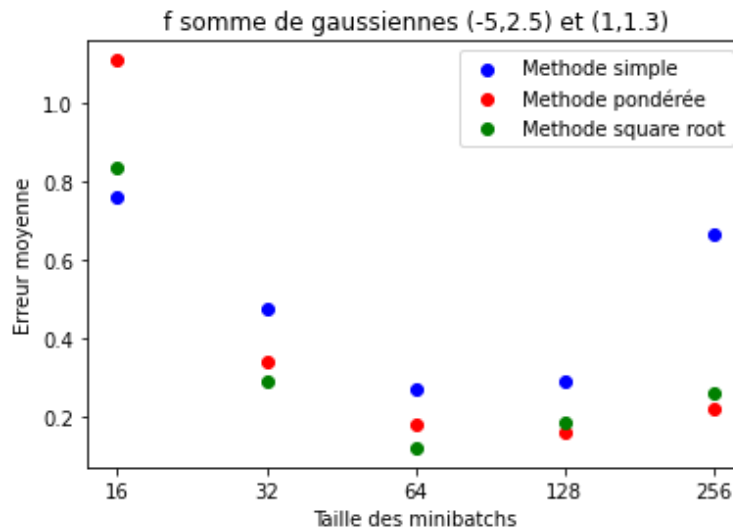


FIGURE 2 – Erreur pour $g : x \mapsto x$ et f une somme de gaussiennes de paramètres $(-5,2.5)$ et $(1,1.3)$

Dans le cas des **mini-batches** la méthode de "square root rule" [1] est la meilleur, notamment car elle est plus robuste dans son estimation de la variance. Pour des batchs suffisamment grands, l'estimateur de la variance de [3] devient suffisamment robuste. Dans ce cas, c'est la méthodes de [3] qui est donc la plus performante.

Références

- [1] Yi Zhou ART B. OWEN. "The square root rule for adaptive importance sampling". In : (2019). URL : <https://arxiv.org/abs/1901.02976>.
- [2] P. HALL et C. C. HEYDE. "Martingale limit theory and its application. Academic Press". In : (1980).
- [3] François PORTIER et Bernard DELYON. "Asymptotic optimality of adaptive importance sampling". In : *Advances in Neural Information Processing Systems*. Sous la dir. de S. BENGIO et al. T. 31. Curran Associates, Inc., 2018. URL : <https://proceedings.neurips.cc/paper/2018/file/1bc0249a6412ef49b07fe6f62e6dc8de-Paper.pdf>.

10 Annexe : Code

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3 import scipy.stats
4 from math import exp
5 import scipy.special
6 import warnings
7 warnings.filterwarnings("ignore")
8 #Quelques fonctions      int grer :
9
10 def f(x):
11     mu = -5
12     b = 2.5
13     return(np.exp(-abs(x-mu)/b)/(2*b))
14
15 def gamma(x):
16     if x<=0:
17         return 0
18
19     theta = 0.5
20     k = 9
21
22     return( (x**(k-1)) * np.exp(-x/theta)/(scipy.special.gamma(k)*(theta**k)) )
23
24 def gaussian1(x):
25     mu = -5
26     sigma = 3
27     return scipy.stats.norm.pdf(x,mu,sigma)
28
29 def gaussian2(x):
30     mu = 1
31     sigma = 1.3
32     return scipy.stats.norm.pdf(x,mu,sigma)
33
34 def gaussian_mix(x):
35     return 0.5*(gaussian1(x) + gaussian2(x))
36
37 def g(x):
38     return x
39
40 #La fonction principale faisant appel aux autres
41
42 def run(f,g,n,m0,sigma0,methods):
43     '''output[i] est le r sultat de la methode i dans la liste methods'''
44     output = []
45     for method in methods:
46         if method == 'simple':
47             output.append(gaussian_AIS(f,g,n,m0,sigma0))
48         if method == 'weighted':
49             output.append(gaussian_wAIS(f,g,n,m0,sigma0))
50         if method == 'square_root':
51             output.append(gaussian_srAIS(f,g,n,m0,sigma0))
```

```

52     if method == 'bayesian':
53         output.append(bayesian_gaussian_wAIS(f,g,n,m0,sigma0))
54     return output
55
56 #Les fonctions permettant de d finir et de minimiser la KL pour une q gaussienne
57
58 def KL(f,mu,sigma,mui,sigmai,Xt):
59     return np.mean(np.log10(f(Xt)/scipy.stats.norm.pdf(Xt,mu,sigma)) * (f(Xt)) / (scipy.stats.norm.
60
61 def KL_grad(f,mu,sigma,mui,sigmai,Xt):
62     alpha = (Xt - mu)**2
63
64     gradmu = np.mean((f(Xt)/(scipy.stats.norm.pdf(Xt,mui,sigmai) * np.log(10)))*(mu - Xt)/(sigma**2
65     gradsig = np.mean((f(Xt)/(scipy.stats.norm.pdf(Xt,mui,sigmai) * np.log(10)))*(sigma**2 - alpha)
66
67     return gradmu,gradsig
68
69 #print(KL_grad(f,0,1,-2,2,np.array([-1,2,-4,1,0])))
70 #print((KL(f,10**-3,1,-2,2,np.array([-1,2,-4,1,0]))-KL(f,0,1,-2,2,np.array([-1,2,-4,1,0])))*(10**3)
71 #print((KL(f,0,1+10**-3,-2,2,np.array([-1,2,-4,1,0]))-KL(f,0,1,-2,2,np.array([-1,2,-4,1,0])))*(10**
72
73 def minimisation_KL(f,mui,sigmai,Xt):
74
75     def kl(theta):
76         return KL(f,theta[0],theta[1],mui,sigmai,Xt)
77
78     def kl_grad(theta):
79         return KL_grad(f,theta[0],theta[1],mui,sigmai,Xt)
80
81     try:
82         m,s=scipy.optimize.minimize(kl,x0=(mui,sigmai),jac=kl_grad).x
83     except:
84         #print("Erreur survenue durant l'optimisation, on garde les deux param tres pr c dents d
85         return mui,sigmai
86     if s<0:
87         #print("L'optimisation donne une variance n gative, on garde les param tres pr c dents"
88         return mui,sigmai
89     return m,s
90
91 #Les fonctions de calcul d'int grale
92
93 def gaussian_AIS(f,g,n,m0,sigma0):
94     '''
95     Approche par une densit  gaussienne classique
96     f est une densit
97     g est une fonction      int grer
98     n est la politique d' chantillonnage
99     m0 et sigma0 sont la moyenne et l' cart  type de la distribution initiale
100     '''
101
102     s = 0
103     N = 0
104     number_of_steps = len(n)
105     m = [m0]

```

```

106     sigma = [sigma0]
107     I = []
108
109     for t in range(number_of_steps):
110         #Tirage des X_(t,i), phase d'exploration
111         Xt = np.random.normal(loc = m[t],scale = sigma[t], size = n[t])
112
113         #Ajout des nouvelles valeurs la somme globale (m j de l'estimateur)
114         partial_sum = np.sum(f(Xt)*g(Xt)/scipy.stats.norm.pdf(Xt,m[t],sigma[t]))
115         s+=partial_sum
116         N += n[t]
117         I.append(s/N)
118
119         #Mise jour de l'chantillonneur
120         muOpti,sigmaOpti = minimisation_KL(f,m[t],sigma[t],Xt)
121         m.append(muOpti)
122         sigma.append(sigmaOpti)
123
124     return I[-1]
125
126 def gaussian_wAIS(f,g,n,m0,sigma0):
127     '''
128     Approche par une densit gaussienne, schema pond r
129     f est une densit
130     g est une fonction int grer
131     n est la politique d'chantillonnage
132     m0 et sigma0 sont la moyenne et l'cart type de la distribution initiale
133     '''
134
135     number_of_steps = len(n)
136     m = [m0]
137     sigma = [sigma0]
138     I = []
139     estimated_variance=[]
140
141     for t in range(number_of_steps):
142         #Tirage des X_(t,i), phase d'exploration
143         Xt = np.random.normal(loc = m[t],scale = sigma[t], size = n[t])
144
145         #Ajout des nouvelles valeurs la somme globale (m j de l'estimateur)
146         partial_sum = np.sum(f(Xt)*g(Xt)/scipy.stats.norm.pdf(Xt,m[t],sigma[t]))
147         I.append(partial_sum/n[t])
148
149         #Estimation de la variance pour cette tape
150         estimated_variance.append(np.sum(((f(Xt)/scipy.stats.norm.pdf(Xt,m[t],sigma[t])) - 1)**2))
151
152         #Mise jour de l'chantillonneur
153         muOpti,sigmaOpti = minimisation_KL(f,m[t],sigma[t],Xt)
154         m.append(muOpti)
155         sigma.append(sigmaOpti)
156
157     #Calcul des poids normalis s:
158     alpha = 1/np.array(estimated_variance)
159     alpha_normalized = alpha/np.sum(alpha)

```

```

160
161     try:
162         assert(abs(np.sum(alpha_normalized)-1)<10**-3)
163     except:
164         print("Erreur d'assertion W. L' cart      0 est : " + str(abs(np.sum(alpha_normalized)-1)))
165         print("Les poids sont " + str(alpha_normalized))
166     return np.sum(alpha_normalized*np.array(I))
167
168
169 def gaussian_srAIS(f,g,n,m0,sigma0):
170     '''
171     Approche par une densit  gaussienne, schema pond r  par des racines carr es
172     f est une densit
173     g est une fonction      int grer
174     n est la politique d'  chantillonnage
175     m0 et sigma0 sont la moyenne et l' cart  type de la distribution initiale
176     '''
177
178     number_of_steps = len(n)
179     m = [m0]
180     sigma = [sigma0]
181     I = []
182     estimated_variance=[]
183
184     for t in range(number_of_steps):
185         #Tirage des X_(t,i), phase d'exploration
186         Xt = np.random.normal(loc = m[t],scale = sigma[t], size = n[t])
187
188         #Ajout des nouvelles valeurs      la somme lobale (m j de l'estimateur)
189         partial_sum = np.sum(f(Xt)*g(Xt)/scipy.stats.norm.pdf(Xt,m[t],sigma[t]))
190         I.append(partial_sum/n[t])
191
192         #Mise      jour de l'  chantillonneur
193         muOpti,sigmaOpti = minimisation_KL(f,m[t],sigma[t],Xt)
194         m.append(muOpti)
195         sigma.append(sigmaOpti)
196
197     #Calcul des poids normalis s:
198     alpha = [(k+1)**(1/2) for k in range(number_of_steps)]
199     alpha_normalized = alpha/np.sum(alpha)
200     try:
201         assert(abs(np.sum(alpha_normalized)-1)<10**-3)
202     except:
203         print("Erreur d'assertion SR. L' cart      0 est : " + str(abs(np.sum(alpha_normalized)-1)))
204         print("Les poids sont " + str(alpha_normalized))
205
206     return np.sum(alpha_normalized*np.array(I))
207
208 def gaussian_wAIS_livePlot(f,g,n,m0,sigma0):
209     '''
210     Approche par une densit  gaussienne, schema pond r
211     f est une densit
212     g est une fonction      int grer
213     n est la politique d'  chantillonnage

```

```

214     m0 et sigma0 sont la moyenne et l' cart type de la distribution initiale
215     ',,'
216     # plt.close()
217     # plt.figure()
218     plt.ion()
219     plt.show()
220     fig,ax=plt.subplots()
221     number_of_steps = len(n)
222     m = [m0]
223     sigma = [sigma0]
224     I = []
225     estimated_variance=[]
226
227     for t in range(number_of_steps):
228         #Tirage des X_(t,i), phase d'exploration
229         Xt = np.random.normal(loc = m[t],scale = sigma[t], size = n[t])
230
231         #Ajout des nouvelles valeurs la somme globale (m j de l'estimateur)
232         partial_sum = np.sum(f(Xt)*g(Xt)/scipy.stats.norm.pdf(Xt,m[t],sigma[t]))
233         I.append(partial_sum/n[t])
234
235         #Estimation de la variance pour cette tape
236         estimated_variance.append(np.sum(((f(Xt)/scipy.stats.norm.pdf(Xt,m[t],sigma[t])) - 1)**2))
237
238         #Mise jour de l' chantillonneur
239         muOpti,sigmaOpti = minimisation_KL(f,m[t],sigma[t],Xt)
240         m.append(muOpti)
241         sigma.append(sigmaOpti)
242
243         #affichage live
244         Z = []
245         Lq = []
246         Lf = []
247         a = -13
248         b = 3
249         steps = 150
250         for z in range(steps):
251             abs = a + z*(b-a)/steps
252             Z.append(abs)
253             Lq.append(scipy.stats.norm.pdf(abs,muOpti,sigmaOpti))
254             Lf.append(f(abs))
255
256         ax.clear()
257         plt.plot(Z,Lq,color="blue",label = "q actuel")
258         plt.plot(Z,Lf,color="red",label = "f cible")
259         plt.legend()
260         ax.set_ylim(0,0.38)
261         ax.set_xlim(a,b)
262         plt.gcf().canvas.draw()
263         plt.pause(0.01)
264
265         m.append(muOpti)
266         sigma.append(sigmaOpti)
267

```



```

268 #Calcul des poids normalis s :
269 alpha = [(k+1)**(1/2) for k in range(number_of_steps)]
270 alpha_normalized = alpha/np.sum(alpha)
271 try:
272     assert(abs(np.sum(alpha_normalized)-1)<10**-3)
273 except:
274     print("Erreur d'assertion SR. L' cart      0 est : " + str(abs(np.sum(alpha_normalized)-1)))
275     print("Les poids sont " + str(alpha_normalized))
276
277 return np.sum(alpha_normalized*np.array(I))
278
279
280 def bayesian_gaussian_wAIS(f,g,n,m0,sigma0):
281     '''
282     Approche par une densit  gaussienne, schema pond r
283     f est une fonction L1
284     g est une fonction      int grer
285     n est la politique d'  chantillonnage
286     m0 et sigma0 sont la moyenne et l' cart  type de la distribution initiale
287     '''
288
289     number_of_steps = len(n)
290     m = [m0]
291     sigma = [sigma0]
292     I = []
293     estimated_variance=[]
294
295     for t in range(number_of_steps):
296         #Tirage des X_(t,i), phase d'exploration
297         Xt = np.random.normal(loc = m[t],scale = sigma[t], size = n[t])
298
299         #Ajout des nouvelles valeurs      la somme globale (m j de l'estimateur)
300         partial_sum = np.sum(f(Xt)*g(Xt)/scipy.stats.norm.pdf(Xt,m[t],sigma[t]))
301         partial_sum2 = np.sum(f(Xt)/scipy.stats.norm.pdf(Xt,m[t],sigma[t]))
302         I.append(partial_sum/partial_sum2)
303
304         #Estimation de la variance pour cette tape
305         estimated_variance.append(np.sum(((f(Xt)/scipy.stats.norm.pdf(Xt,m[t],sigma[t])) - 1)**2))
306
307         #Mise      jour de l'  chantillonneur
308         muOpti,sigmaOpti = minimisation_KL(f,m[t],sigma[t],Xt)
309         m.append(muOpti)
310         sigma.append(sigmaOpti)
311
312     #Calcul des poids normalis s :
313     alpha = 1/np.array(estimated_variance)
314     alpha_normalized = alpha/np.sum(alpha)
315
316     try:
317         assert(abs(np.sum(alpha_normalized)-1)<10**-3)
318     except:
319         print("Erreur d'assertion W. L' cart      0 est : " + str(abs(np.sum(alpha_normalized)-1)))
320         print("Les poids sont " + str(alpha_normalized))
321     return np.sum(alpha_normalized*np.array(I))

```

```
322
323 n=[20 for i in range(10)]
324 methods = ['simple','weighted','square_root']
325 results = run(gaussian1,g,n,0,1,methods)
326
327 for i in range(len(methods)):
328     print('Le resultat pour la methode '+ methods[i]+' est : ' + str(results[i]) + ', soit une er
329
330 #Cellule pour le calcul, longue
331
332 results = []
333
334 for i in range(50):
335     results.append(run(f,g,n,0,1,methods))
336
337 #Cellule pour l'affichage des resultats, execution rapide
338
339 error_simple=[]
340 error_w=[]
341 error_sr=[]
342
343 for result in results:
344     error_simple.append(abs((result[0])+5))
345     error_w.append(abs((result[1])+5))
346     error_sr.append(abs((result[2])+5))
347
348 print('En moyenne sur 50 tests : \n')
349 print("L'erreur pour la methode " + methods[0]+' est : ' + str(np.mean(error_simple)))
350 print("L'erreur pour la methode " + methods[1]+' est : ' + str(np.mean(error_w)))
351 print("L'erreur pour la methode " + methods[2]+' est : ' + str(np.mean(error_sr)))
352
353 errors_s = []
354 errors_w = []
355 errors_sr = []
356 from time import time
357 import datetime
358 t = time()
359 methods = ['simple','weighted','square_root']
360
361 results=[]
362
363 starting_value=4
364 end_value = 7
365 max_samples = 10
366
367 number_of_samples = 40
368 print("Le nombre de variables tirer pour cette batterie de tests est : " + str(2**max_samples *
369 for i in range(starting_value,end_value + 1):
370     print("On en est i = "+ str(i))
371     results=[]
372     n=[2**i for h in range(2**(max_samples-i))]
373     for i in range(number_of_samples):
374         results.append(run(f,g,n,0,1,methods))
375     error_simple=[]
```

```
376     error_w=[]
377     error_sr=[]
378     for result in results:
379         error_simple.append(abs((result[0])+5))
380         error_w.append(abs((result[1])+5))
381         error_sr.append(abs((result[2])+5))
382     errors_s.append(np.mean(error_simple))
383     errors_w.append(np.mean(error_w))
384     errors_sr.append(np.mean(error_sr))
385
386 X = [str(i+starting_value) for i in range(len(errors_s))]
387 plt.scatter(X,errors_s,color="blue",label="Methode simple")
388 plt.scatter(X,errors_w,color="red",label="Methode pond r e")
389 plt.scatter(X,errors_sr,color="green",label="Methode square root")
390 plt.xlabel("i")
391 plt.title("f densit Laplacienne (-5,2.5)")
392 plt.legend()
393 plt.show()
394
395 print("Temps coul : "+ str(time()-t) + "s.")
396
397 errors_s =[]
398 errors_w=[]
399 errors_sr=[]
400 methods = ['simple','weighted','square_root']
401
402 t = time()
403
404 results=[]
405
406 starting_value=4
407 end_value = 8
408 max_samples = 10
409
410 number_of_samples = 50
411 print("Le nombre de variables tirer pour cette batterie de tests est : " + str(2**max_samples *
412 for i in range(starting_value,end_value + 1):
413     print("On en est i = "+ str(i))
414     results=[]
415     n=[2**i for h in range(2**(max_samples-i))]
416     for i in range(number_of_samples):
417         results.append(run(gaussian_mix,g,n,-4,1,methods))
418     error_simple=[]
419     error_w=[]
420     error_sr=[]
421     for result in results:
422         error_simple.append(abs((result[0])+2))
423         error_w.append(abs((result[1])+2))
424         error_sr.append(abs((result[2])+2))
425     errors_s.append(np.mean(error_simple))
426     errors_w.append(np.mean(error_w))
427     errors_sr.append(np.mean(error_sr))
428
429 X = [str(i+starting_value) for i in range(len(errors_s))]
```

```
430 plt.scatter(X,errors_s,color="blue",label="Methode simple")
431 plt.scatter(X,errors_w,color="red",label="Methode pond r e")
432 plt.scatter(X,errors_sr,color="green",label="Methode square root")
433 plt.xlabel("i")
434 plt.title("f somme de gaussiennes (-5,2.5) et (1,1.3)")
435 plt.legend()
436 plt.show()
437
438 print("Temps coul : "+ str(time()-t) + "s.")
439
440 #Cellule pour le calcul, longue
441 methods = ['bayesian']
442 results = []
443 n=[2**12 for i in range(2**4)]
444
445 def not_f(x):
446     return f(x)
447
448 for i in range(20):
449     results.append(run(not_f,g,n,0,1,methods))
450
451 #Cellule pour l'affichage des r sultats , ex cution rapide
452
453 error_w=[]
454
455
456 for result in results:
457     error_w.append(abs((result[0])+5))
458
459 print('En moyenne sur 50 tests : \n')
460 print("L'erreur pour la m thode " + methods[0]+' est : ' + str(np.mean(error_w)))
461
462 print(len(results))
```