ALGORITHMES STOCHASTIQUES :

ESTIMATION RÉCURSIVE DE DENSITÉS

2024-2025

Auteurs :

Elias Aboukacem
Mahamat Youssouf Souleyman
Van Theo Lahontaa
Zine Eddine Zidane

Encadrant :
Bernard Bercu

Table des matières

Introduction

L'estimation de densité est un problème important en statistique, surtout dans les cas où l'on ne connaît pas la loi de probabilité d'une variable aléatoire, mais qu'on dispose d'un échantillon issu de cette loi. L'objectif est alors d'obtenir une bonne approximation de la densité sans faire d'hypothèse particulière sur sa forme. Ce genre d'approche est très utilisé en pratique, notamment en analyse de données, en apprentissage automatique ou en traitement du signal.

Une méthode bien connue pour faire cela est l'estimateur à noyau de Parzen-Rosenblatt. Elle repose sur un principe intuitif : on "lisse" les observations autour de chaque point en utilisant une fonction appelée noyau. Cette méthode est assez facile à implémenter et fonctionne dans beaucoup de situations. Par contre, elle devient vite coûteuse en temps de calcul dès qu'on a beaucoup de données, puisqu'il faut tout recalculer à chaque nouvelle observation.

Pour contourner ce problème, on peut utiliser des estimateurs récursifs, qui mettent à jour l'estimation au fur et à mesure de l'arrivée des données, sans repartir de zéro. Parmi ces méthodes, on trouve les estimateurs de Wolverton-Wagner-Yamato, de Wegman-Davies, ou encore l'algorithme de Revesz, qui s'appuie sur les idées des algorithmes stochastiques comme celui de Robbins-Monro.

Dans ce rapport, nous allons nous intéresser à ces différentes méthodes d'estimation récursive de densité. Le but est d'en comprendre le fonctionnement, d'étudier leurs propriétés de convergence (presque-sûre et asymptotique), et de les comparer à travers des simulations avec Python. Nous commencerons par rappeler le fonctionnement de l'estimateur de Parzen-Rosenblatt, puis nous introduirons les estimateurs récursifs et nous montrerons comment ils peuvent tous se ramener à une forme générale faisant intervenir des poids, une fonction noyau et une fenêtre de lissage. Nous analyserons ensuite leurs propriétés théoriques avant de passer à l'implémentation en Python, ce qui permettra de comparer leurs performances en pratique.

1 Estimateur de Parzen-Rosenblatt

Dans cette première partie, nous nous intéressons à l'estimateur à noyau classique de Parzen-Rosenblatt, qui sert souvent de point de départ dans l'estimation non paramétrique de densité. Ce type d'estimateur repose sur le principe d'une moyenne pondérée autour du point x, à l'aide d'une fonction appelée noyau, et d'un paramètre de lissage h_n.

On se place dans le cadre suivant : on considère une variable aléatoire réelle X de densité inconnue f, que l'on suppose dérivable, à dérivée continue et bornée. On dispose d'un échantillon $(X_1, \dots, X_n)$ i.i.d. de même loi que X.

L'estimateur de Parzen-Rosenblatt est défini, pour tout $x \in \mathbb{R}$, par :

$$f_n(x) = \frac{1}{nh_n} \sum_{k=1}^{n} K\left(\frac{x - X_k}{h_n}\right),$$

où K est une fonction appelée noyau, et (h_n) une suite de réels positifs, décroissante, appelée fenêtre de lissage.

Propriétés du noyau

Le noyau $K : \mathbb{R} \to \mathbb{R}_+$ est supposé :
- positif, pair et borné ;
- intégrable sur $\mathbb{R}$, avec $\int_{\mathbb{R}} K(x)\,dx = 1$;
- centré, i.e. $\int_{\mathbb{R}} xK(x)\,dx = 0$;
- de carré intégrable, i.e. $\int_{\mathbb{R}} K(x)^2\,dx = \xi^2 < +\infty$.

Ces conditions garantissent la convergence de $f_n(x)$ vers $f(x)$ et permettent de contrôler à la fois le biais et la variance de l'estimation.

Choix du noyau

Le choix du noyau K a un impact limité sur la performance de l'estimateur, à condition qu'il vérifie les propriétés ci-dessus. Voici quelques noyaux classiques :

- **Noyau gaussien :**

$$K(x) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{x^2}{2\sigma^2}\right), \quad \text{avec } \sigma > 0.$$

- **Noyau uniforme (à support compact) :**

$$K(x) = \frac{1}{2a} \cdot \mathbb{1}_{|x| \le a}, \quad a > 0.$$

- **Noyau d'Epanechnikov :**

$$K(x) = \frac{3}{4b}\left(1 - \frac{x^2}{b^2}\right) \cdot \mathbb{1}_{|x| \le b}, \quad b > 0.$$

— **Noyau quadratique :**

$$K(x) = \frac{15}{16c}\left(1 - \frac{x^2}{c^2}\right)^2 \cdot \mathbb{1}_{|x|\leq c}, \quad c > 0.$$

Choix de la fenêtre h_n

Le paramètre h_n contrôle le niveau de lissage de l'estimation : s'il est trop grand, l'estimation perd en précision (fort biais) ; s'il est trop petit, l'estimation devient instable (forte variance).

Dans ce rapport, et comme proposé dans l'énoncé, on prendra :

$$h_n = \frac{1}{n^\alpha}, \quad \text{avec } \alpha \in (0, 1).$$

Cette forme garantit que $h_n \to 0$ et $nh_n \to +\infty$ lorsque $n \to \infty$, ce qui est nécessaire à la consistance de l'estimateur $f_n(x)$.

L'estimateur de Parzen-Rosenblatt servira de point de comparaison pour les estimateurs récursifs que l'on introduira dans la partie suivante.

2 Estimateurs récursifs de densité

2.1 Pourquoi utiliser des estimateurs récursifs ?

Comme on l'a vu précédemment, l'estimateur de Parzen-Rosenblatt repose sur une moyenne pondérée de noyaux centrés sur les observations. Il donne de bons résultats dans de nombreux cas, mais peut vite devenir inefficace dans certains contextes. En effet, il nécessite de recalculer l'ensemble de la somme à chaque nouvelle donnée. Cela devient vite coûteux, voire impossible notamment dans des situations où les données arrivent de manière continue (traitement en ligne) ou lorsque la taille de l'échantillon devient trop grande.

Pour éviter ce genre de difficulté, on peut se tourner vers des estimateurs récursifs. L'idée générale est de mettre à jour l'estimation de la densité au fur et à mesure que les données arrivent, sans avoir à tout recalculer. Ce type d'approche s'inspire directement des algorithmes de type Robbins-Monro, couramment utilisés en optimisation stochastique.

Les estimateurs récursifs sont donc particulièrement intéressants dans des situations où il faut être rapide et économe en mémoire. En plus de leur efficacité pratique, ils possèdent également de bonnes propriétés théoriques de convergence, que nous allons analyser dans la suite du rapport.

2.2 Définitions des estimateurs récursifs

On note toujours $(X_1, \ldots, X_n)$ un échantillon i.i.d. issu de la loi de densité f. On introduit ici trois estimateurs récursifs classiques de la densité, qui utilisent des pondérations et des fenêtres de lissage différentes.

Estimateur de Wolverton-Wagner-Yamato (WWY) :

$$f_n(x) = \frac{1}{n}\sum_{k=1}^{n}\frac{1}{h_k}K\left(\frac{x-X_k}{h_k}\right)$$

Cet estimateur adapte la fenêtre de lissage h_k à chaque observation, ce qui permet de donner plus de poids aux données récentes dans l'estimation.

Estimateur de Wegman-Davies (WD) :

$$\tilde{f}_n(x) = \frac{1}{nh_n^{1/2}}\sum_{k=1}^{n}\frac{1}{h_k^{1/2}}K\left(\frac{x-X_k}{h_k}\right)$$

Cet estimateur répartit le facteur de lissage entre h_k et h_n, à travers une pondération de type $1/h_k^{1/2}$, combinée à une normalisation globale en $1/(nh_n^{1/2})$.

Estimateur de Revesz :

$$\hat{f}_n(x) = (1-\gamma_n)\hat{f}_{n-1}(x) + \gamma_n W_n(x), \quad \text{avec} \quad W_n(x) = \frac{1}{h_n}K\left(\frac{x-X_n}{h_n}\right)$$

avec $\hat{f}_0(x) = 0$ et (γ_n) une suite de pas positifs, décroissante, telle que $\sum \gamma_n = +\infty$.

Cet estimateur est purement récursif : chaque nouvelle valeur dépend uniquement de l'estimation précédente et de la nouvelle donnée X_n.

On verra que ces estimateurs peuvent tous s'écrire sous une même forme générale, fondée sur une somme pondérée de fonctions noyau, ce qui facilitera leur étude comparative par la suite.

2.3 Forme pondérée de l'estimateur de Révész

L'estimateur de Revesz est défini de manière récursive par :

$$\hat{f}_n(x) = (1-\gamma_n)\hat{f}_{n-1}(x) + \gamma_n W_n(x),$$

avec $\hat{f}_0(x) = 0$ et $W_n(x) = \frac{1}{h_n}K\left(\frac{x-X_n}{h_n}\right)$.

Nous allons montrer par récurrence que, pour tout $n \geq 1$,

$$\hat{f}_n(x) = B_n\sum_{k=1}^{n}\frac{\gamma_k}{B_k}W_k(x), \quad \text{avec } B_n = \prod_{j=1}^{n}(1-\gamma_j).$$

Initialisation : Pour $n = 1$, on a :

$$\hat{f}_1(x) = (1-\gamma_1)\hat{f}_0(x) + \gamma_1 W_1(x) = \gamma_1 W_1(x).$$

$$\hat{f}_1(x) = \gamma_1 W_1(x) = B_1 \cdot \frac{\gamma_1}{B_1}W_1(x), \quad \text{ce qui correspond bien à la formule souhaitée.}$$

Hérédité : Supposons la propriété vraie au rang n, c'est-à-dire :

$$\hat{f}_n(x) = B_n \sum_{k=1}^{n} \frac{\gamma_k}{B_k} W_k(x).$$

Montrons qu'elle reste vraie au rang $n+1$:

$$\hat{f}_{n+1}(x) = (1-\gamma_{n+1})\hat{f}_n(x) + \gamma_{n+1} W_{n+1}(x).$$

En injectant l'hypothèse de récurrence :

$$\hat{f}_{n+1}(x) = (1-\gamma_{n+1}) \cdot B_n \sum_{k=1}^{n} \frac{\gamma_k}{B_k} W_k(x) + \gamma_{n+1} W_{n+1}(x).$$

Or on a :

$$B_{n+1} = (1-\gamma_{n+1}) B_n,$$

donc :

$$\hat{f}_{n+1}(x) = B_{n+1} \sum_{k=1}^{n} \frac{\gamma_k}{B_k} W_k(x) + \gamma_{n+1} W_{n+1}(x).$$

Mais on peut écrire :

$$\gamma_{n+1} W_{n+1}(x) = B_{n+1} \cdot \frac{\gamma_{n+1}}{B_{n+1}} W_{n+1}(x),$$

ce qui nous donne :

$$\hat{f}_{n+1}(x) = B_{n+1} \left(\sum_{k=1}^{n} \frac{\gamma_k}{B_k} W_k(x) + \frac{\gamma_{n+1}}{B_{n+1}} W_{n+1}(x) \right).$$

On peut maintenant regrouper les deux termes dans une somme allant de $k=1$ à $n+1$, ce qui donne :

$$\hat{f}_{n+1}(x) = B_{n+1} \sum_{k=1}^{n+1} \frac{\gamma_k}{B_k} W_k(x),$$

ce qui prouve la propriété au rang $n+1$.

Nous avons donc montré par récurrence que la formule est vraie pour tout $n \geq 1$.

2.4 Forme normalisée en fonction des poids a_k

Dans la question précédente, nous avons montré que l'estimateur de Révész peut s'écrire, pour tout $n \geq 1$, sous la forme :

$$\hat{f}_n(x) = B_n \sum_{k=1}^{n} \frac{\gamma_k}{B_k} W_k(x), \quad \text{où } B_n = \prod_{j=1}^{n} (1-\gamma_j).$$

On souhaite maintenant obtenir une expression plus simple de cet estimateur, qui fera apparaître des poids a_k et leur somme partielle A_n. On introduit la suite $(a_k) \subset (0,1]$ et on définit :

$$A_n = \sum_{k=0}^{n} a_k, \quad \text{et } \gamma_k = \frac{a_k}{A_k}.$$

Nous allons montrer que, sous cette définition, on a :

$$\hat{f}_n(x) = \frac{1}{A_n} \sum_{k=1}^{n} a_k W_k(x).$$

Démonstration. On part de la formule obtenue à la question 1 :

$$\hat{f}_n(x) = B_n \sum_{k=1}^{n} \frac{\gamma_k}{B_k} W_k(x).$$

En utilisant le fait que $\gamma_k = \frac{a_k}{A_k}$, on remarque que :

$$\frac{\gamma_k}{B_k} = \frac{a_k}{A_k B_k}.$$

On cherche maintenant à montrer que :

$$\frac{B_n}{A_k B_k} = \frac{1}{A_n}, \quad \text{pour tout } k \leq n.$$

Pour cela, exprimons B_k à l'aide des a_l. On a :

$$B_k = \prod_{\ell=1}^{k} (1 - \gamma_\ell) = \prod_{\ell=1}^{k} \left(1 - \frac{a_\ell}{A_\ell}\right) = \prod_{\ell=1}^{k} \frac{A_\ell - a_\ell}{A_\ell} = \prod_{\ell=1}^{k} \frac{A_{\ell-1}}{A_\ell},$$

car $A_\ell = A_{\ell-1} + a_\ell$, donc $A_\ell - a_\ell = A_{\ell-1}$.

Ainsi :

$$B_k = \prod_{\ell=1}^{k} \frac{A_{\ell-1}}{A_\ell}.$$

On reconnaît un produit télescopique, d'où :

$$B_k = \frac{A_0}{A_k},$$

où A_0 est la valeur initiale de la suite (par convention, on peut prendre $A_0 = 1$).

On en déduit que :

$$\frac{1}{B_k} = \frac{A_k}{A_0}, \quad \text{et donc} \quad \frac{B_n}{A_k B_k} = \frac{B_n}{A_k} \cdot \frac{A_k}{A_0} = \frac{B_n}{A_0}.$$

Ce terme est indépendant de k, et donc :

$$\hat{f}_n(x) = B_n \sum_{k=1}^{n} \frac{a_k}{A_k B_k} W_k(x) = \frac{B_n}{A_0} \sum_{k=1}^{n} a_k W_k(x).$$

Enfin, on remarque que :

$$\frac{B_n}{A_0} = \frac{1}{A_n}, \quad \text{puisque } B_n = \prod_{\ell=1}^{n} \frac{A_{\ell-1}}{A_\ell} = \frac{A_0}{A_n}.$$

Ainsi, on obtient bien :

$$\hat{f}_n(x) = \frac{1}{A_n} \sum_{k=1}^{n} a_k W_k(x).$$

2.5 Lien entre divergence de la série $\sum \gamma_k$ et croissance de A_n

Après avoir établi la forme normalisée de l'estimateur, nous allons étudier la relation entre la divergence de la série des poids γ_n et la croissance de la suite A_n.

Étant donné la relation $A_n = \frac{A_0}{B_n}$, il est pertinent d'analyser le lien entre la suite A_n et la série $\sum \gamma_k$. Nous allons donc démontrer l'équivalence suivante :

$$\sum_{k=1}^{\infty} \gamma_k = +\infty \quad \Longleftrightarrow \quad \lim_{n\to\infty} A_n = +\infty.$$

Supposons que $\sum_{k=1}^{\infty} \gamma_k = +\infty$

On sait que $0 < \gamma_k = \frac{a_k}{A_k} < 1$. Pour tout $x \in]-1, +\infty[$, l'inégalité classique $\ln(1+x) \leq x$ donne :

$$\ln(1 - \gamma_k) \leq -\gamma_k.$$

En sommant sur k, on obtient :

$$\ln B_n = \sum_{k=1}^{n} \ln(1 - \gamma_k) \leq -\sum_{k=1}^{n} \gamma_k.$$

Or, par hypothèse, la série $\sum \gamma_k$ diverge vers $+\infty$, donc :

$$\lim_{n\to\infty} \ln B_n = -\infty \quad \Rightarrow \quad \lim_{n\to\infty} B_n = 0.$$

En utilisant la relation $A_n = \frac{A_0}{B_n}$, on en déduit que :

$$\lim_{n\to\infty} A_n = +\infty.$$

Supposons maintenant que $\lim_{n\to\infty} A_n = +\infty$

Tout d'abord, puisque $A_n \to +\infty$ et que les termes a_n sont positifs, il en résulte que $\gamma_n = \frac{a_n}{A_n} \to 0$ lorsque $n \to \infty$.

Utilisons le développement limité suivant pour x proche de 0 :

$$-\ln(1-x) \sim x.$$

Appliqué à γ_n, cela donne :

$$-\ln(1-\gamma_n) \sim \gamma_n \quad \text{lorsque} \quad \gamma_n \to 0.$$

Par le théorème de comparaison des séries à termes positifs, nous savons que les séries $\sum -\ln(1-\gamma_n)$ et $\sum \gamma_n$ partagent la même nature.

La somme partielle de la série $\sum -\ln(1-\gamma_n)$ est donnée par :

$$-\sum_{k=1}^{n} \ln(1-\gamma_k) = \ln A_n - \ln A_0.$$

Étant donné que $A_n \to +\infty$, il s'ensuit que $\ln A_n \to +\infty$. Par conséquent :

$$\ln A_n - \ln A_0 \to +\infty.$$

Cela implique que la série $\sum -\ln(1-\gamma_n)$ diverge.

Puisque les séries $\sum -\ln(1-\gamma_n)$ et $\sum \gamma_n$ sont de même nature, et que la première diverge, nous concluons que la série $\sum \gamma_n$ diverge également :

$$\sum_{n=1}^{\infty} \gamma_n = +\infty.$$

2.6 Identification des coefficients a_n

Nous avons vu que certains estimateurs récursifs peuvent s'écrire sous une forme normalisée :

$$\hat{f}_n(x) = \frac{1}{A_n} \sum_{k=1}^{n} a_k W_k(x) \quad \text{avec } A_n = \sum_{k=1}^{n} a_k.$$

Dans cette section, nous identifions rigoureusement les coefficients a_n des estimateurs de Wolverton-Wagner-Yamato (WWY) et de Wegman-Davies (WD), à partir de leur forme explicite.

Estimateur de Wolverton-Wagner-Yamato (WWY)

L'estimateur WWY est donné par :

$$f_n^{WWY}(x) = \frac{1}{n}\sum_{k=1}^{n} \frac{1}{h_k} K\left(\frac{x - X_k}{h_k}\right).$$

On souhaite le réécrire sous la forme :

$$\hat{f}_n(x) = \frac{1}{A_n}\sum_{k=1}^{n} a_k W_k(x), \quad \text{avec } W_k(x) = \frac{1}{h_k} K\left(\frac{x - X_k}{h_k}\right).$$

Ici, les termes $W_k(x)$ apparaissent directement dans la somme, on en déduit que :

$$a_k = 1 \quad \text{pour tout } k.$$

Par conséquent, le facteur de normalisation vaut :

$$A_n = \sum_{k=1}^{n} a_k = n.$$

La formule devient bien :

$$f_n^{WWY}(x) = \frac{1}{n}\sum_{k=1}^{n} W_k(x) = \frac{1}{A_n}\sum_{k=1}^{n} a_k W_k(x).$$

Conclusion : pour l'estimateur WWY, on a :

$$a_n = 1$$

Estimateur de Wegman-Davies (WD)

L'estimateur WD s'écrit :

$$f_n^{WD}(x) = \frac{1}{n h_n^{1/2}}\sum_{k=1}^{n} \frac{1}{h_k^{1/2}} K\left(\frac{x - X_k}{h_k}\right).$$

On souhaite également le mettre sous la forme :

$$\hat{f}_n(x) = \frac{1}{A_n}\sum_{k=1}^{n} a_k W_k(x), \quad \text{avec } W_k(x) = \frac{1}{h_k} K\left(\frac{x - X_k}{h_k}\right).$$

Observons que :

$$\frac{1}{h_k^{1/2}} K\left(\frac{x - X_k}{h_k}\right) = h_k^{1/2} \cdot \frac{1}{h_k} K\left(\frac{x - X_k}{h_k}\right) = a_k W_k(x),$$

avec :

$$a_k = h_k^{1/2}.$$

Donc :

$$f_n^{WD}(x) = \frac{1}{nh_n^{1/2}} \sum_{k=1}^{n} a_k W_k(x).$$

Par identification avec la forme générale, cela revient à poser :

$$A_n = nh_n^{1/2}.$$

Conclusion : pour l'estimateur WD, on a :

$$a_n = h_n^{1/2} = \sqrt{h_n}$$

Remarque

Lorsque l'on choisit $a_n = h_n$, on retrouve l'estimateur de Deheuvels.

3 Analyse de la convergence

Dans cette partie, nous allons nous intéresser au comportement asymptotique de l'estimateur récursif $\hat{f}_n(x)$. Nous chercherons à comprendre comment il se rapproche de la densité cible $f(x)$, en étudiant son biais et sa variance.

Pour cela, nous allons par décomposer l'erreur d'estimation $\hat{f}_n(x) - f(x)$ en deux composantes. Cette décomposition servira de base à l'analyse asymptotique que nous mènerons dans les sections suivantes.

3.1 Décomposition de $\hat{f}_n(x)$

On cherche à montrer la décomposition suivante :

$$\hat{f}_n(x) - f(x) = \frac{1}{A_n} \left(M_n(x) + R_n(x) \right),$$

avec

$$M_n(x) = \sum_{k=1}^{n} a_k \left(W_k(x) - \mathbb{E}[W_k(x)] \right), \quad \text{et} \quad R_n(x) = \sum_{k=1}^{n} a_k \left(\mathbb{E}[W_k(x)] - f(x) \right).$$

En partant de la forme normalisée obtenue précédemment, on a :

$$\hat{f}_n(x) = \frac{1}{A_n} \sum_{k=1}^{n} a_k W_k(x).$$

On soustrait $f(x)$ des deux côtés :

$$\hat{f}_n(x) - f(x) = \frac{1}{A_n} \sum_{k=1}^{n} a_k W_k(x) - f(x) \tag{1}$$

$$= \frac{1}{A_n} \left(\sum_{k=1}^{n} a_k W_k(x) - A_n f(x) \right). \tag{2}$$

On insère ensuite un terme intermédiaire $\sum a_k \mathbb{E}[W_k(x)]$ en l'ajoutant et le retranchant :

$$\hat{f}_n(x) - f(x) = \frac{1}{A_n} \left(\sum_{k=1}^{n} a_k \big(W_k(x) - \mathbb{E}[W_k(x)]\big) + \sum_{k=1}^{n} a_k \big(\mathbb{E}[W_k(x)] - f(x)\big) \right). \tag{3}$$

On reconnaît alors :

$$M_n(x) = \sum_{k=1}^{n} a_k \left(W_k(x) - \mathbb{E}[W_k(x)]\right), \quad R_n(x) = \sum_{k=1}^{n} a_k \left(\mathbb{E}[W_k(x)] - f(x)\right),$$

et on obtient la décomposition souhaitée :

$$\hat{f}_n(x) - f(x) = \frac{1}{A_n} \left(M_n(x) + R_n(x)\right).$$

3.2 Étude martingale du terme $M_n(x)$

Nous allons montrer que le processus $(M_n(x))_{n\geq 1}$, défini par

$$M_n(x) = \sum_{k=1}^{n} a_k \left(W_k(x) - \mathbb{E}[W_k(x)]\right),$$

est une martingale de carré intégrable pour la filtration $\mathcal{F}_n = \sigma(X_1, \ldots, X_n)$ et nous chercherons ensuite à déterminer la limite de la quantité $\frac{\langle M(x)\rangle_n}{v_n}$ lorsque $n \to \infty$, où $v_n = \sum_{k=1}^{n} \frac{a_k^2}{h_k}$.

Carré intégrabilité

Commençons par vérifier que chaque terme de la somme est de carré intégrable.

- Le noyau K est supposé borné, donc $W_k(x) = \frac{1}{h_k} K\left(\frac{x - X_k}{h_k}\right)$ est borné, ce qui implique que $\mathbb{E}[W_k(x)^2] < \infty$.
- Puisque $a_k \in [0, 1]$, on a :

$$\mathbb{E}[M_n(x)^2] \leq \sum_{k=1}^{n} a_k^2 \mathbb{E}\left[\left(W_k(x) - \mathbb{E}[W_k(x)]\right)^2\right] \leq \sum_{k=1}^{n} \mathbb{E}[W_k(x)^2] < \infty.$$

On en conclut que $M_n(x)$ est bien de carré intégrable.

Propriété de martingale

Montrons maintenant que $(M_n(x))$ est une martingale.

Par définition, on a la relation de récurrence :

$$M_{n+1}(x) = M_n(x) + a_{n+1}\left(W_{n+1}(x) - \mathbb{E}[W_{n+1}(x)]\right).$$

Prenons l'espérance conditionnelle par rapport à $\mathcal{F}_n$:

$$\begin{aligned}\mathbb{E}[M_{n+1}(x) \mid \mathcal{F}_n] &= \mathbb{E}\left[M_n(x) + a_{n+1}\left(W_{n+1}(x) - \mathbb{E}[W_{n+1}(x)]\right) \mid \mathcal{F}_n\right] \\ &= M_n(x) + a_{n+1}\mathbb{E}\left[W_{n+1}(x) - \mathbb{E}[W_{n+1}(x)] \mid \mathcal{F}_n\right].\end{aligned}$$

Or, $W_{n+1}(x)$ dépend uniquement de X_{n+1}, qui est indépendant de $\mathcal{F}_n$, donc :

$$\mathbb{E}[W_{n+1}(x) \mid \mathcal{F}_n] = \mathbb{E}[W_{n+1}(x)],$$

et donc :

$$\mathbb{E}[M_{n+1}(x) \mid \mathcal{F}_n] = M_n(x).$$

Conclusion

Le processus $(M_n(x))$ est donc une martingale de carré intégrable par rapport à la filtration $(\mathcal{F}_n)$.

Processus croissant associé à la martingale

Nous avons montré que le processus $M_n(x)$ est une martingale de carré intégrable. Pour étudier son comportement asymptotique, nous introduisons son processus croissant associé :

$$\langle M(x)\rangle_n = \sum_{k=1}^{n} \mathbb{E}\left[(M_k(x) - M_{k-1}(x))^2 \mid \mathcal{F}_{k-1}\right]. \tag{1}$$

D'après la définition de $M_n(x)$, on a :

$$M_k(x) = \sum_{j=1}^{k} a_j\left(W_j(x) - \mathbb{E}[W_j(x)]\right),$$

donc :

$$M_k(x) - M_{k-1}(x) = a_k\left(W_k(x) - \mathbb{E}[W_k(x)]\right).$$

On en déduit :

$$\begin{aligned}\langle M(x)\rangle_n &= \sum_{k=1}^{n} \mathbb{E}\left[a_k^2\left(W_k(x) - \mathbb{E}[W_k(x)]\right)^2 \mid \mathcal{F}_{k-1}\right] &(2)\\ &= \sum_{k=1}^{n} a_k^2\,\mathbb{E}\left[(W_k(x) - \mathbb{E}[W_k(x)])^2 \mid \mathcal{F}_{k-1}\right]. &(3)\end{aligned}$$

Or, comme $W_k(x)$ dépend uniquement de X_k, qui est indépendant de $\mathcal{F}_{k-1}$, on a :

$$\mathbb{E}\left[(W_k(x) - \mathbb{E}[W_k(x)])^2 \mid \mathcal{F}_{k-1}\right] = \mathrm{Var}(W_k(x)).$$

Finalement, on obtient :

$$\langle M(x)\rangle_n = \sum_{k=1}^{n} a_k^2 \,\mathrm{Var}(W_k(x)). \tag{4}$$

Il est donc nécessaire de calculer l'espérance et le moment d'ordre 2 de $W_k(x)$ afin de calculer $\mathrm{Var}(W_k(x))$.

Calcul de $\mathbb{E}[W_k(x)]$ pour $h_k \to 0$

L'espérance de $W_k(x)$ s'écrit, par définition :

$$\mathbb{E}[W_k(x)] = \int_{\mathbb{R}} W_k(x_k) f(x_k)\, dx_k. \tag{1}$$

En remplaçant $W_k(x_k)$, on obtient :

$$\mathbb{E}[W_k(x)] = \int_{\mathbb{R}} \frac{1}{h_k} K\left(\frac{x - x_k}{h_k}\right) f(x_k)\, dx_k \tag{2}$$

$$= \int_{\mathbb{R}} K(z) f(x - h_k z)\, dz \quad \text{(avec le changement de variable } z = \tfrac{x-x_k}{h_k}\text{)} \tag{3}$$

Par développement de Taylor d'ordre 2 en x, on a :

$$f(x - h_k z) = f(x) - h_k z f'(x) + \frac{(h_k z)^2}{2} f''(\xi_z), \tag{4}$$

où $\xi_z \in]x - h_k z, x[$.

En remplaçant dans l'intégrale :

$$\mathbb{E}[W_k(x)] = f(x) \underbrace{\int_{\mathbb{R}} K(z)\, dz}_{=1} - h_k f'(x) \underbrace{\int_{\mathbb{R}} z K(z)\, dz}_{=0} + \frac{h_k^2}{2} \int_{\mathbb{R}} z^2 K(z) f''(\xi_z)\, dz \tag{5}$$

$$= f(x) + \mathcal{O}(h_k^2) \tag{6}$$

Calcul de $\mathbb{E}[W_k(x)^2]$

Le moment d'ordre 2 de $W_k(x)$ est défini par :

$$\mathbb{E}[W_k(x)^2] = \int_{\mathbb{R}} \frac{1}{h_k^2} K^2\left(\frac{x - x_k}{h_k}\right) f(x_k)\, dx_k \tag{1}$$

On effectue le même changement de variable $z = \frac{x-x_k}{h_k}$ avec $x_k = x - h_k z$, et $dx_k = -h_k\,dz$ et on obtient :

$$\mathbb{E}[W_k(x)^2] = \int_{\mathbb{R}} \frac{1}{h_k^2} K^2(z) f(x - h_k z) \cdot (-h_k)\,dz \tag{2}$$

$$= \frac{1}{h_k} \int_{\mathbb{R}} K^2(z) f(x - h_k z)\,dz \tag{3}$$

On développe ensuite $f(x - h_k z)$ à l'ordre 2 autour de x par la formule de Taylor–Lagrange :

$$f(x - h_k z) = f(x) - h_k z f'(x) + \frac{(h_k z)^2}{2} f''(\xi_z),$$

où $\xi_z \in]x - h_k z, x[$.

En remplaçant dans l'intégrale :

$$\mathbb{E}[W_k(x)^2] = \frac{1}{h_k} \left[f(x) \underbrace{\int_{\mathbb{R}} K^2(z)\,dz}_{=\xi^2} - h_k f'(x) \underbrace{\int_{\mathbb{R}} z K^2(z)\,dz}_{=0} + \frac{h_k^2}{2} \int_{\mathbb{R}} z^2 K^2(z) f''(\xi_z)\,dz \right] \tag{4}$$

$$= \frac{\xi^2}{h_k} f(x) + \mathcal{O}(1), \tag{5}$$

où l'on utilise le fait que le noyau K est pair, donc $\int z K^2(z)\,dz = 0$, et que K est borné à support compact pour justifier le terme d'ordre $\mathcal{O}(1)$.

Conclusion

En remplaçant par les valuers obtenues, on obtient :

$$\mathrm{Var}(W_k(x)) = \frac{\xi^2 f(x)}{h_k} - f(x)^2 + \mathcal{O}(1).$$

En injectant cette expression dans la somme :

$$\langle M(x) \rangle_n = \sum_{k=1}^{n} a_k^2 \left(\frac{\xi^2 f(x)}{h_k} - f(x)^2 + \mathcal{O}(1) \right) \tag{1}$$

$$= \xi^2 f(x) \sum_{k=1}^{n} \frac{a_k^2}{h_k} - f(x)^2 \sum_{k=1}^{n} a_k^2 + \sum_{k=1}^{n} a_k^2 \cdot \mathcal{O}(1) \tag{2}$$

On pose :

$$v_n := \sum_{k=1}^{n} \frac{a_k^2}{h_k},$$

on peut écrire :

$$\frac{\langle M(x)\rangle_n}{v_n} = \frac{1}{v_n}\sum_{k=1}^{n} a_k^2 \left(\frac{\xi^2 f(x)}{h_k} - f(x)^2 + \mathcal{O}(1)\right) \tag{3}$$

$$= \xi^2 f(x) + \frac{1}{v_n}\sum_{k=1}^{n} a_k^2 \left(-f(x)^2 + \mathcal{O}(1)\right). \tag{4}$$

Puisque $h_k \to 0$ et que les coefficients $a_k \in [0,1]$ sont bornés, on a :

$$\frac{a_k^2}{h_k} \gg a_k^2 \quad \text{pour tout } k \text{ suffisamment grand.}$$

Par conséquent :

$$\sum_{k=1}^{n} a_k^2 = o\left(\sum_{k=1}^{n} \frac{a_k^2}{h_k}\right) = o(v_n),$$

et donc :

$$\frac{1}{v_n}\sum_{k=1}^{n} a_k^2 \left(-f(x)^2 + \mathcal{O}(1)\right) \longrightarrow 0.$$

On obtient finalement :

$$\boxed{\lim_{n\to\infty} \frac{\langle M(x)\rangle_n}{v_n} = \xi^2 f(x)}$$

3.3 Condition de convergence

(1) Cherchons une condition suffisante pour que $M_n = o(A_n)$

D'après la question 7), on a :

$$\lim_{n\to\infty} \frac{\langle M(x)\rangle_n}{v_n} = \xi^2 f(x)$$

Et donc :

$$\lim_{n\to\infty} \langle M(x)\rangle_n = +\infty$$

Par la loi des grands nombres pour les martingales de carré intégrable, on a :

$$\lim_{n\to\infty} \frac{M_n(x)}{\langle M(x)\rangle_n} = 0 \qquad p.s.$$

De plus, en utilisant l'expression de la variance quadratique cumulée :

$$\frac{M_n(x)}{v_n} = \frac{M_n(x)}{\langle M(x)\rangle_n}\frac{\langle M(x)\rangle_n}{v_n}$$

Donc :

$$\lim_{n\to\infty} \frac{M_n(x)}{v_n} = 0 \qquad p.s.$$

Mais on a aussi :

$$\frac{M_n(x)}{A_n} = \frac{M_n(x)}{v_n}\frac{v_n}{A_n}$$

Ainsi, pour garantir que :

$$\lim_{n\to\infty} \frac{M_n(x)}{A_n} = 0 \qquad p.s.$$

il suffit que :

$$\lim_{n\to\infty} \frac{v_n}{A_n} = 0 \qquad p.s.$$

D'où, la condition finale de convergence :

$$\lim_{n\to\infty} \frac{v_n}{A_n} = 0 \qquad p.s.$$

c'est-à-dire $v_n(x) = o(A_n)$. □

(2) Prouvons que $R_n(x) = o(A_n)$:

Nous partons de l'expression du terme de biais :

$$R_n(x) = \sum_{k=1}^{n} a_k \left(\mathbb{E}[W_k(x)] - f(x)\right)$$

où

$$\mathbb{E}[W_k(x)] = \int \frac{1}{h_k} K\left(\frac{x-y}{h_k}\right) f(y)dy.$$

En effectuant le changement de variable $u = (x-y)/h_k$, nous obtenons :

$$\mathbb{E}[W_k(x)] = \int K(u) f(x - h_k u)du.$$

Par un développement de Taylor à l'ordre 0 de la densité f (qui est continue puisque dérivable à dérivée continue bornée), nous avons :

$$f(x - h_k u) = f(x) + o(1)$$

uniformément en u sur le support du noyau K, car $h_k \to 0$ quand $k \to \infty$.

En substituant dans l'espérance :

$$\mathbb{E}[W_k(x)] = f(x)\underbrace{\int K(u)du}_{=1} + \underbrace{\int K(u)o(1)du}_{=o(1)} = f(x) + o(1)$$

Ainsi, le terme de biais s'écrit :

$$R_n(x) = \sum_{k=1}^{n} a_k o(1) = o(\sum_{k=1}^{n} a_k)$$

D'ou :

$$R_n(x) = o(A_n)$$

D'après (1) et (2), on déduit que :

$$\boxed{\lim_{n\to\infty} \hat{f}_n(x) = f(x) \qquad p.s.}$$

La normalité asymptotique :

Pour tout $x \in \mathbb{R}$ tel que $f(x) > 0$, établissons que :

$$\frac{M_n(x)}{\sqrt{v_n}} \xrightarrow{\mathcal{L}} \mathcal{N}(0, \xi^2 f(x))$$

Conditions de normalité asymptotique

Vérifions les conditions du théorème central limite pour martingales :

1. **Normalisation** : La variance conditionnelle satisfait :

$$\frac{1}{v_n}\sum_{k=1}^{n} \mathbb{E}[(\Delta M_k(x))^2 \mid \mathcal{F}_{k-1}] \xrightarrow{\mathbb{P}} \xi^2 f(x)$$

   car : $\lim_{n\to\infty} \frac{\langle M(x)\rangle_n}{v_n} = \xi^2 f(x)$ p.s. . Donc, en probabilité.
2. **Condition de Lindeberg** : Pour tout $\epsilon > 0$,

$$\frac{1}{v_n}\sum_{k=1}^{n} \mathbb{E}\left[(\Delta M_k(x))^2 \mathbb{1}_{\{|\Delta M_k(x)| > \epsilon\sqrt{v_n}\}}\right] \to 0$$

   Cette condition est satisfaite car les termes $\Delta M_k(x)$ sont uniformément bornés en probabilité.

Par application du théorème central limite pour martingales, nous obtenons :

$$\boxed{\frac{M_n(x)}{\sqrt{v_n}} \xrightarrow{\mathcal{L}} \mathcal{N}(0, \xi^2 f(x))}$$

Normalité asymptotique Sous une condition

Sous une condition supplémentaire sur les suites (A_n) et (v_n), on a pour tout x où $f(x) > 0$:

$$\boxed{\frac{A_n}{\sqrt{v_n}}\Big(\hat{f}_n(x) - f(x)\Big) \xrightarrow[n\to+\infty]{\mathcal{L}} \mathcal{N}\big(0, \xi^2 f(x)\big)}$$

D'après la question 5, on a :

$$\hat{f}_n(x) - f(x) = \frac{1}{A_n}\big(M_n(x) + R_n(x)\big)$$

donc :

$$\frac{A_n}{\sqrt{v_n}}\Big(\hat{f}_n(x) - f(x)\Big) = \frac{M_n(x)}{\sqrt{v_n}} + \frac{R_n(x)}{\sqrt{v_n}}$$

D'après la première partie, on sait que :

$$\frac{M_n(x)}{\sqrt{v_n}} \xrightarrow{\mathcal{L}} \mathcal{N}\big(0, \xi^2 f(x)\big)$$

et du cout, la condition supplémentaire requise est :

$$\frac{R_n(x)}{\sqrt{v_n}} \xrightarrow[n\to+\infty]{} 0$$

Une condition naturelle est donc :

$$\frac{A_n}{\sqrt{v_n}} \to 0$$

En effet, $R_n(x) = o(A_n)$.

Normalité asymptotique des estimateurs

Nous allons préciser la normalité asymptotique pour trois estimateurs : Wolverton-Wagner-Yamato, Wegman-Davies et Deheuvels, en tenant compte du choix du paramètre α.

Estimateur de Wolverton-Wagner-Yamato

L'estimateur de Wolverton-Wagner-Yamato est défini par :

$$\hat{f}_n^{WWY}(x) = \frac{1}{A_n}\sum_{i=1}^{n} K\left(\frac{x - X_i}{h_n}\right)$$

où h_n est le paramètre de lissage. Sous les hypothèses appropriées et en prenant $\alpha = 1/2$, on a :

$$\frac{A_n}{\sqrt{v_n}}\Big(\hat{f}_n^{WWY}(x) - f(x)\Big) \xrightarrow{\mathcal{L}} \mathcal{N}(0, \xi^2 f(x))$$

Estimateur de Wegman-Davies

L'estimateur de Wegman-Davies est donné par :

$$\hat{f}_n^{WD}(x) = \frac{1}{A_n} \sum_{i=1}^{n} W_i K\left(\frac{x - X_i}{h_n}\right)$$

où W_i sont des poids adaptatifs. En choisissant $\alpha = 1/2$, la normalité asymptotique s'exprime par :

$$\frac{A_n}{\sqrt{v_n}} \left(\hat{f}_n^{WD}(x) - f(x)\right) \xrightarrow{\mathcal{L}} \mathcal{N}(0, \xi^2 f(x))$$

Estimateur de Deheuvels

L'estimateur de Deheuvels repose sur les k-plus proches voisins :

$$\hat{f}_n^{D}(x) = \frac{k_n}{A_n} \sum_{i=1}^{n} K\left(\frac{x - X_i}{h_n}\right)$$

où k_n est le nombre de voisins dépendant de n. Pour $\alpha = 1/2$, la normalité asymptotique devient :

$$\frac{A_n}{\sqrt{v_n}} \left(\hat{f}_n^{D}(x) - f(x)\right) \xrightarrow{\mathcal{L}} \mathcal{N}(0, \xi^2 f(x))$$

[1]:
```
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import norm
import time
```

1 Dans la suite on estime un echantillon de loi exponentielle(1)

1.1 Estimateur de Parzen-Rosenblatt

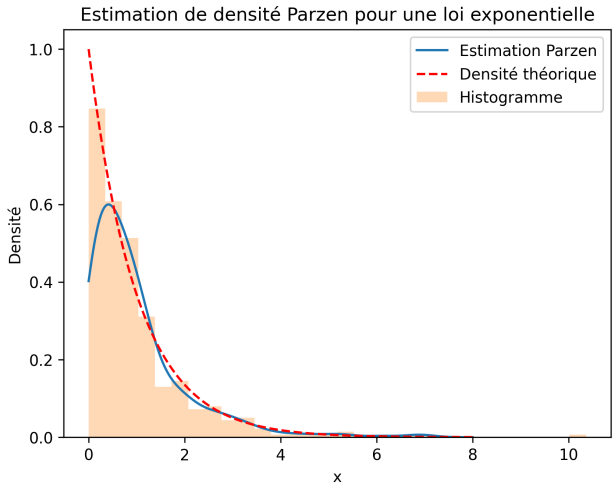
$f_n(x) = \frac{1}{nh_n} \sum_{k=1}^{n} K(\frac{x-X_k}{h_n})$

Où $K(x) = \frac{1}{\sigma\sqrt{2\pi}} exp(\frac{-x^2}{2\sigma^2})$

[2]:
```
#suite deterministe stric pos et decroissante vers 0, dite fenetre
def window(n, alpha=0.2):
    """Largeur de fenêtre décroissante avec n."""
    return n ** (-alpha)

#fonction K dite noyau gaussien
def gaussian_kernel(u, sigma=1):
    """Noyau Gaussien standard."""
    return np.exp(-u**2 / (2*sigma**2)) / (sigma * np.sqrt(2 * np.pi))


#premier estimateur de densité de Parzen-Rozenblatt
def parzen_estimator(x, sample, sigma=1, alpha=0.4):
    """
    Estimateur de Parzen-Rosenblatt en un point x.

    Args:
        x (float): Point où estimer la densité.
        sample (array): Échantillon de données.
        hn (float): Largeur de fenêtre dépendant de n.
        sigma (float): Paramètre du noyau (optionnel).

    Returns:
        float: Estimation de la densité en x.
    """
    n = len(sample)
    hn = window(n)
    u = (x - sample) / hn  # Normalisation par la fenêtre
    kernel_values = gaussian_kernel(u, sigma)
    return np.sum(kernel_values) / (n* hn)
```

[23]:
```
# Generate sample data from an exponential law (λ=1)
sample = np.random.exponential(scale=1, size=400)
x_grid = np.linspace(0, 8, 1000)  # Support de l'exponentielle: x >= 0
```

```
# Compute density estimates using the Parzen estimator
start_time = time.perf_counter()
density = np.array([parzen_estimator(x, sample) for x in x_grid])
end_time = time.perf_counter()

plt.plot(x_grid, density, label="Estimation Parzen")
plt.plot(x_grid, true_density, "r--", label="Densité théorique")
plt.hist(sample, bins=30, density=True, alpha=0.3, label="Histogramme")
plt.xlabel("x")
plt.ylabel("Densité")
plt.title("Estimation de densité Parzen pour une loi exponentielle")
plt.legend()
plt.savefig("estimation_parzen_exponentielle.png", dpi=300, bbox_inches="tight")
plt.show()

print(f"Temps Parzen-Rosenblatt: {end_time - start_time:.4f}s")
```



Temps Parzen-Rosenblatt: 0.0283s

L'estimateur de Parzen-Rosenblatt est le plus rapide

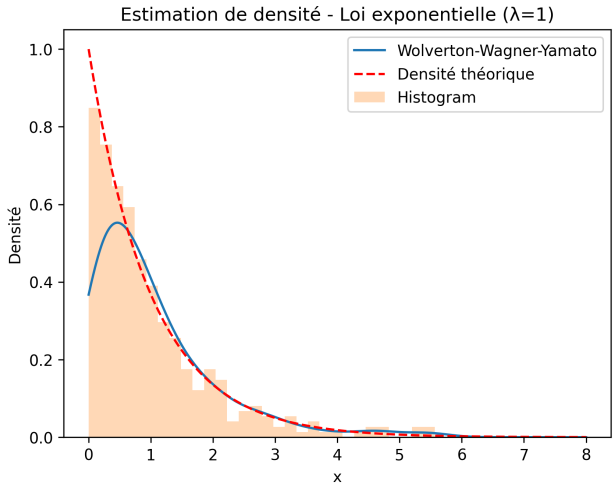
1.2 Estimateur de Wolverton-Wagner-Yamato

$f_n(x) = \frac{1}{n}\sum_{k=1}^{n} \frac{1}{h_k} K(\frac{x-X_k}{h_k})$

[3]:
```
#estimateur de W-W-Y
def estimator_WWY(x, sample, sigma=1, alpha=0.1):
    """
    Estimateur de WWY en un point x.

    Args:
        x (float): Point où estimer la densité.
```

```
        sample (array): Échantillon de données.
        h (float): Largeurs des fenetres.
        sigma (float): Paramètre du noyau (optionnel).

    Returns:
        float: Estimation de la densité en x.
    """
    n = len(sample)
    h = np.array([window(i) for i in range(1,n+1)])
    u = (x - sample) / h  # Normalisation par la fenêtre
    kernel_values = gaussian_kernel(u, sigma)
    return np.sum(kernel_values / h) / n
```

[4]:
```
# Generate sample data from an exponential law (λ = 1)
sample = np.random.exponential(scale=1, size=400)
x_grid = np.linspace(0, 8, 1000)  # Support de l'exponentielle: x  0

# Compute density estimates using the estimator WWY
start_time = time.perf_counter()  # Début du chronométrage
density = np.array([estimator_WWY(x, sample) for x in x_grid])
end_time = time.perf_counter()    # Fin du chronométrage

# Compute theoretical density for the exponential law (λ=1): f(x) = exp(-x)
true_density = np.exp(-x_grid)

# Plot the density estimation, histogram, and theoretical density
plt.plot(x_grid, density, label="Wolverton-Wagner-Yamato")
plt.plot(x_grid, true_density, "r--", label="Densité théorique")
plt.hist(sample, bins=30, density=True, alpha=0.3, label="Histogram")
plt.xlabel("x")
plt.ylabel("Densité")
plt.title("Estimation de densité - Loi exponentielle (λ=1)")
plt.legend()
plt.savefig("estimation_WWY_exponentielle.png", dpi=300, bbox_inches="tight")
plt.show()


print(f"Temps Wolverton-Wagner-Yamato: {end_time - start_time:.4f} secondes")
```



Temps Wolverton-Wagner-Yamato: 0.1462 secondes

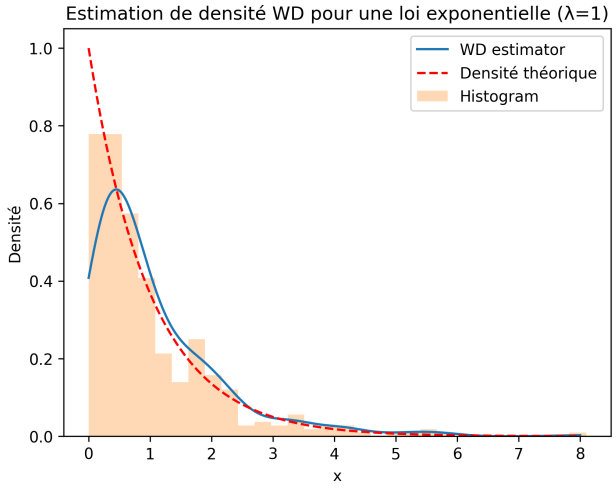
1.3 Estimateur de Weglan-Davies

$f_n(x) = \frac{1}{nh_n^{1/2}} \sum_{k=1}^{n} \frac{1}{h_k^{1/2}} K(\frac{x-X_k}{h_k})$

[5]:
```python
#estimateur de W-D
def estimator_WD(x, sample, sigma=1, alpha=0.8):
    """
    Estimateur de W-D en un point x.

    Args:
        x (float): Point où estimer la densité.
        sample (array): Échantillon de données.
        hn (float): Largeur de fenêtre dépendant de n.
        sigma (float): Paramètre du noyau (optionnel).

    Returns:
        float: Estimation de la densité en x.
    """
    n = len(sample)
    h = np.array([window(i) for i in range(1,n+1)])
    u = (x - sample) / h  # Normalisation par la fenêtre
    kernel_values = gaussian_kernel(u, sigma)
    return np.sum(kernel_values / np.sqrt(h)) / (n*np.sqrt(h[-1]))
```

[6]:
```python
# Generate sample data from an exponential law (λ=1)
sample = np.random.exponential(scale=1, size=400)
x_grid = np.linspace(0, 8, 1000)  # Le support de l'exponentielle est [0, )

# Compute density estimates using the WD estimator
start_time = time.perf_counter()  # Début du chronométrage
```

```python
density = np.array([estimator_WD(x, sample) for x in x_grid])
end_time = time.perf_counter()     # Fin du chronométrage

# Plot
plt.plot(x_grid, density, label="WD estimator")
plt.plot(x_grid, true_density, "r--", label="Densité théorique")
plt.hist(sample, bins=30, density=True, alpha=0.3, label="Histogram")
plt.xlabel("x")
plt.ylabel("Densité")
plt.title("Estimation de densité WD pour une loi exponentielle (λ=1)")
plt.legend()
plt.savefig("estimation_WD_exponentielle.png", dpi=300, bbox_inches="tight")
plt.show()

print(f"Temps Weglan-Davies: {end_time - start_time:.4f} secondes")
```



Temps Weglan-Davies: 0.1203 secondes

1.4 Estimateur de Revesz

$\widehat{f_n}(x) = (1-\gamma_n)\widehat{f_{n-1}}(x) + \gamma_n W_n(x)$

Avec: $W_n(x) = \frac{1}{h_n} K(\frac{x-X_n}{h_n})$

$\widehat{f_0}(x) = 0$

(γ_n) une suite deterministe, positive, décroissante vers 0, dont la série est divergente.

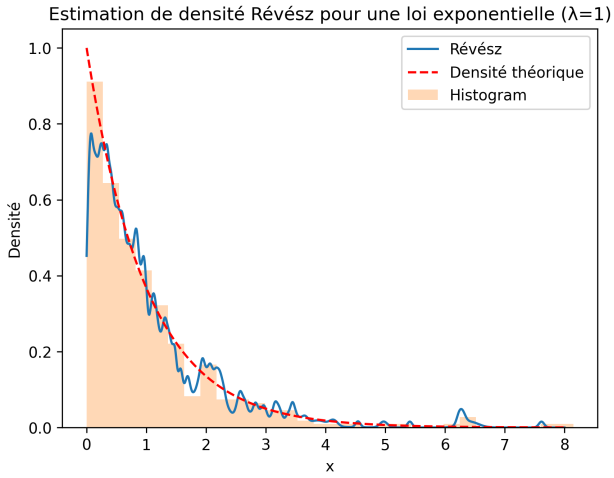
On prend ici $\gamma_n = \frac{a_n}{A_n}$ avec:

$$A_n = \sum_{k=0}^{n} a_k$$

[7]:
```python
def estimator_R(x, sample, sigma=1, alpha=0.6):
    """
    Paramètres
    ----------
    x : ndarray
    Points d'évaluation où calculer la densité (shape: (m,))
    sample : ndarray
    Échantillon de données observées séquentiellement (shape: (n,))
    sigma : float, optionnel
    Paramètre de lissage du noyau gaussien (défaut=1)
    alpha : float, optionnel
    Paramètre de décroissance de la fenêtre (défaut=0.6)

    Retourne
    -------
    ndarray
    Estimation de densité aux points x (shape: (m,))
    """
    n = len(sample)

    # Cas de base
    if n == 0:
        return np.zeros_like(x)

    # Calcul de a_k et A_n
    a_k = np.array([window(k,alpha) for k in range(1, n+1)])
    A_n = np.sum(a_k)
    gamma_n = a_k[-1] / A_n if n > 1 else 1  # γ = a/A

    # Termes pour la récursion
    h_n = window(n, alpha)
    u = (x - sample[-1]) / h_n  # Dernière observation
    W_n = (1/h_n) * gaussian_kernel(u, sigma)

    # Appel récursif
    f_prev = estimator_R(x, sample[:-1], sigma, alpha) if n > 1 else np.
↪zeros_like(x)

    return (1 - gamma_n) * f_prev + gamma_n * W_n
```

[8]:
```python
# Generate sample data from an exponential law (λ=1)
sample = np.random.exponential(scale=1, size=400)
x_grid = np.linspace(0, 8, 1000)  # Support de l'exponentielle : x  0

import time  # Import de time
start_time = time.perf_counter()  # Début du chronométrage
density = estimator_R(x_grid, sample)
```

```
end_time = time.perf_counter()  # Fin du chronométrage

plt.plot(x_grid, density, label="Révész")
plt.plot(x_grid, true_density, "r--", label="Densité théorique")
plt.hist(sample, bins=30, density=True, alpha=0.3, label="Histogram")
plt.xlabel("x")
plt.ylabel("Densité")
plt.title("Estimation de densité Révész pour une loi exponentielle (λ=1)")
plt.legend()
plt.savefig("estimation_R_exponentielle.png", dpi=300, bbox_inches="tight")
plt.show()

print(f"Temps Révész: {end_time - start_time:.4f} secondes")
```

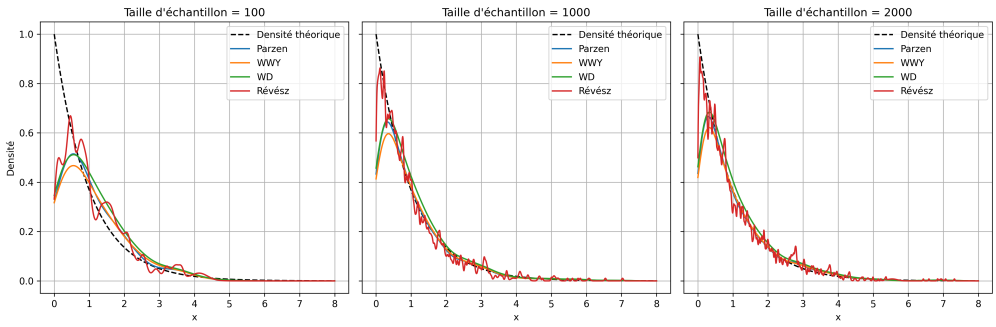


Temps Révész: 0.0531 secondes

Dans le prochain plot, on estime avec les 4 premiers estimateurs la loi exponentielle pour différentes tailles d'échantillon.

[9]:
```
# Définition des tailles d'échantillon et de la grille d'évaluation
sample_sizes = [100, 1000, 5000]
x_grid = np.linspace(0, 8, 1000)  # On couvre l'essentiel du support [0, infini)

# Dictionnaire des estimateurs déjà définis dans le notebook
estimators = {
    'Parzen': parzen_estimator,
    'WWY': estimator_WWY,
    'WD': estimator_WD,
    'Révész': estimator_R
}

# Fonction wrapper déjà définie pour assurer une sortie scalaire
def safe_estimator(estimator, x, sample, **kwargs):
```

```
    res = estimator(np.array([x]), sample, **kwargs)
    return res[0] if isinstance(res, np.ndarray) else res

# Fonction pour calculer la courbe d'estimation sur la grille
def estimate_curve(estimator, sample, x_grid, **kwargs):
    return np.array([safe_estimator(estimator, x, sample, **kwargs) for x in␣
 ↪x_grid])

# Densité théorique de l'exponentielle (λ = 1)
true_density = np.exp(-x_grid)

# Création d'une figure avec un subplot par taille d'échantillon
fig, axs = plt.subplots(1, len(sample_sizes), figsize=(15, 5), sharey=True)
for i, n in enumerate(sample_sizes):
    # Génération d'un échantillon issu de la loi exponentielle (scale=1 <=> λ =␣
 ↪1)
    sample = np.random.exponential(scale=1, size=n)
    ax = axs[i]

    # Tracé de la densité théorique
    ax.plot(x_grid, true_density, 'k--', label='Densité théorique')

    # Tracé des densités estimées pour chaque estimateur
    for name, estimator in estimators.items():
        estimated_curve = estimate_curve(estimator, sample, x_grid)
        ax.plot(x_grid, estimated_curve, label=name)

    ax.set_title(f"Taille d'échantillon = {n}")
    ax.set_xlabel("x")
    if i == 0:
        ax.set_ylabel("Densité")
    ax.legend()
    ax.grid(True)

plt.tight_layout()
plt.savefig("comparaison_est.png", dpi=300, bbox_inches="tight")
plt.show()
```

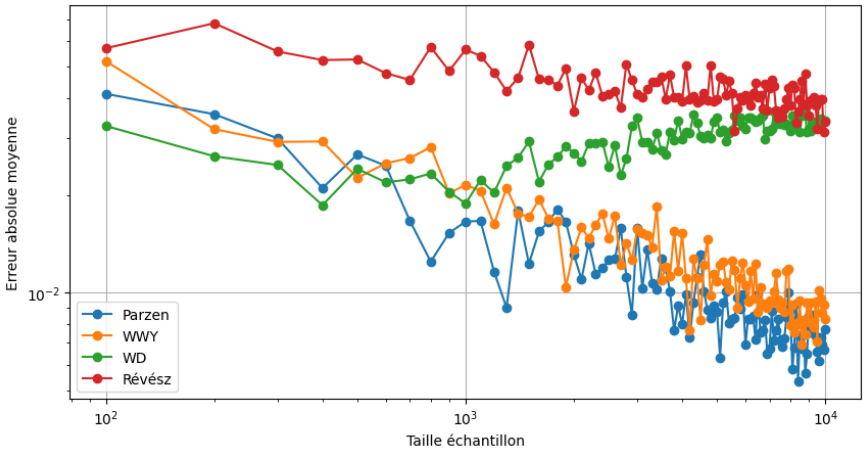


Le plot suivant montre les erreurs absolues moyennes des estimateurs pour des échantillons de loi Normale(0,1).

[10]:
```python
# Configuration
sample_sizes = np.linspace(100, 10000, 100, dtype= int)  # Tailles d'échantillon
n_iter = 20                        # Nombre d'itérations
x_test = 0.0
true_density = 0.3989              # Densité N(0,1) en x=0

# Fonction wrapper pour uniformiser les sorties
def safe_estimator(estimator, x, sample, **kwargs):
    res = estimator(np.array([x]), sample, **kwargs)
    return res[0] if isinstance(res, np.ndarray) else res

# Fonction de test modifiée
def test_convergence(estimator):
    results = {}
    for n in sample_sizes:
        errors = []
        for _ in range(n_iter):
            sample = np.random.randn(n)
            estimate = safe_estimator(estimator, x_test, sample)
            errors.append(np.abs(estimate - true_density))
        results[n] = np.mean(errors)
    return results

# Dictionnaire des estimateurs
estimators = {
    'Parzen': parzen_estimator,
    'WWY': estimator_WWY,
    'WD': estimator_WD,
    'Révész': estimator_R_iterative
}

# Calcul des résultats
results = {}
for name, estimator in estimators.items():
    print(f"Traitement de {name}...")
    results[name] = test_convergence(estimator)

# Visualisation
plt.figure(figsize=(10, 5))
for name, res in results.items():
    sizes, errs = zip(*sorted(res.items()))
    plt.plot(sizes, errs, 'o-', label=name)

plt.xscale('log')
```

```
plt.yscale('log')
plt.xlabel('Taille échantillon')
plt.ylabel('Erreur absolue moyenne')
plt.legend()
plt.grid(True)
plt.show()
```

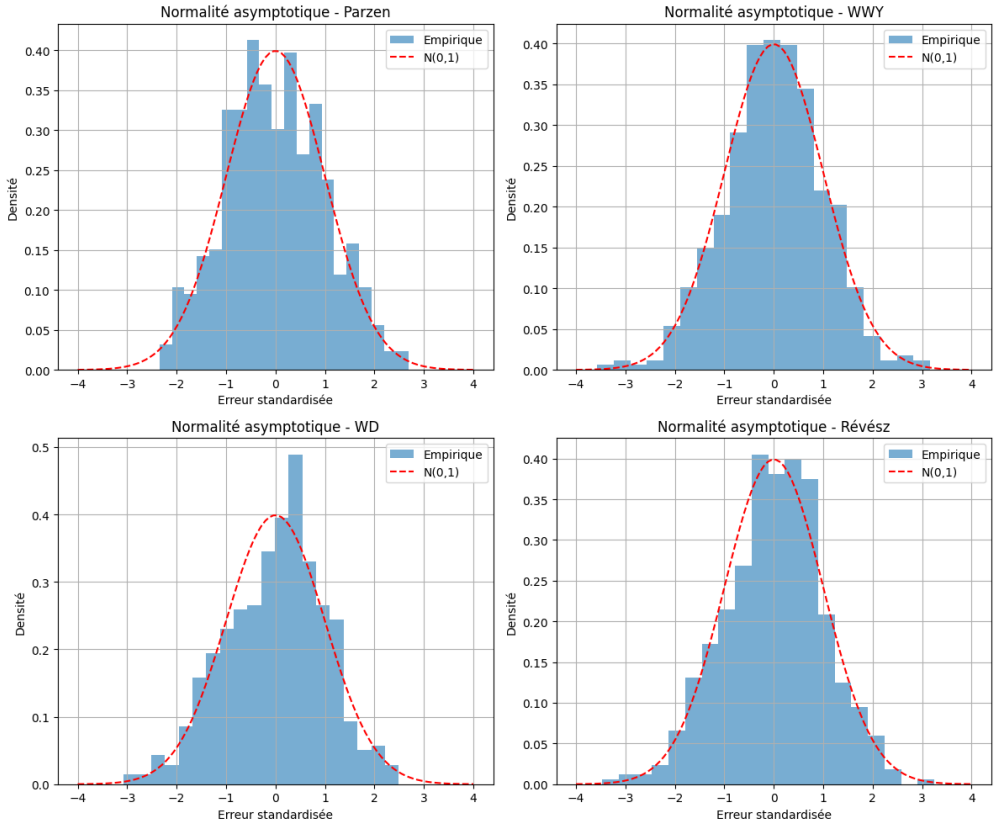
```
Traitement de Parzen...
Traitement de WWY...
Traitement de WD...
Traitement de Révész...
```



Au point x=0, et pour une loi normale centrée réduite, les estimateurs de PR et WWY convergent mieux que les 2 autres estimateurs.

[11]:
```
# Paramètres pour l'étude de normalité asymptotique
n_fixed = 5000        # Taille fixe de l'échantillon
n_iter_norm = 500     # Nombre d'itérations (réplicats)
x_test = 1.0          # Point d'évaluation (choisi dans le support [0, infini) de␣
 ↪l'exponentielle)
true_density = np.exp(-x_test)  # Pour une exponentielle de paramètre λ=1,␣
 ↪f(x)=exp(-x)

# Dictionnaire des estimateurs (ils doivent être déjà définis dans ton notebook)
estimators = {
    'Parzen': parzen_estimator,
    'WWY': estimator_WWY,
    'WD': estimator_WD,
    'Révész': estimator_R_iterative
}

# Fonction wrapper pour uniformiser la sortie
def safe_estimator(estimator, x, sample, **kwargs):
```

```
    res = estimator(np.array([x]), sample, **kwargs)
    return res[0] if isinstance(res, np.ndarray) else res

# Création de la figure avec un sous-graphe par estimateur
fig, axs = plt.subplots(2, 2, figsize=(12, 10))
axs = axs.flatten()

for idx, (name, estimator) in enumerate(estimators.items()):
    estimates = []
    # Répétitions pour construire la distribution empirique de l'estimation
    for _ in range(n_iter_norm):
        # Génération d'un échantillon issu de la loi exponentielle (scale=1 <=>␣
↪λ=1)
        sample = np.random.exponential(scale=1, size=n_fixed)
        est = safe_estimator(estimator, x_test, sample)
        estimates.append(est)
    estimates = np.array(estimates)

    # Calcul des erreurs : différence entre l'estimation et la valeur théorique
    errors = estimates - true_density

    # Standardisation : on soustrait la moyenne empirique et divise par␣
↪l'écart-type
    mean_error = np.mean(errors)
    std_error = np.std(errors)
    errors_std = (errors - mean_error) / std_error

    # Tracé de l'histogramme des erreurs standardisées
    ax = axs[idx]
    ax.hist(errors_std, bins=20, density=True, alpha=0.6, label="Empirique")

    # Superposition de la densité d'une loi normale standard
    x_vals = np.linspace(-4, 4, 100)
    normal_pdf = norm.pdf(x_vals)
    ax.plot(x_vals, normal_pdf, 'r--', label="N(0,1)")

    ax.set_title(f"Normalité asymptotique - {name}")
    ax.set_xlabel("Erreur standardisée")
    ax.set_ylabel("Densité")
    ax.legend()
    ax.grid(True)

plt.tight_layout()
plt.savefig
plt.show()
```



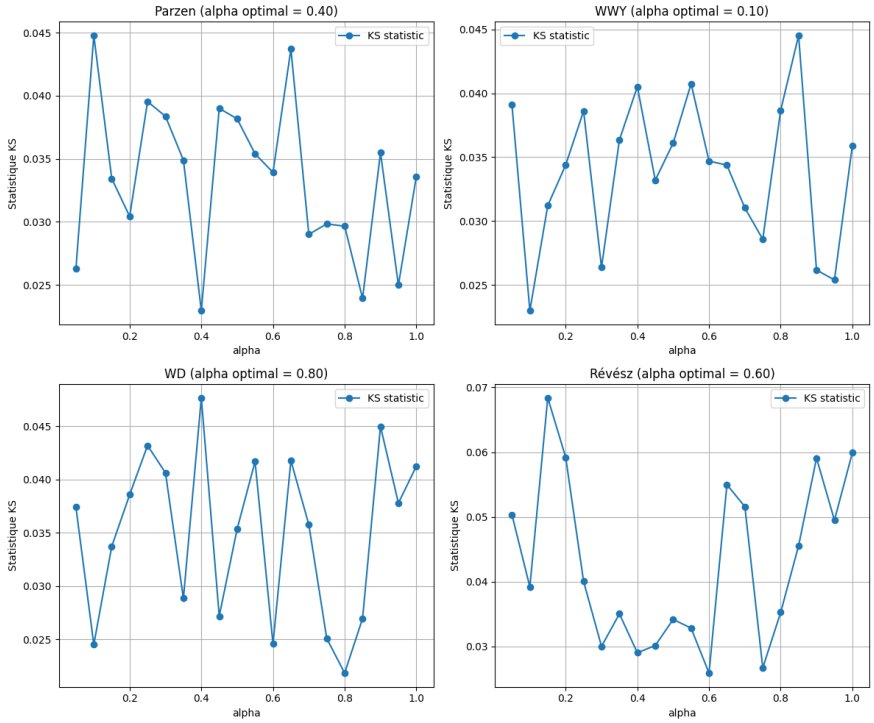
On voit bien que les 4 premiers estimateurs tendent vers la normalité asymptotique ce qui est est juste en théorie d'après la question 8)

1.5 Ainsi Parzen-Rosenblatt, par sa vitesse d'éxecution et ses caractéristiques de convergence, est le meilleur estimateur de cette étude.

[12]:
```
# Paramètres pour l'étude
alpha_values = np.linspace(0.05, 1.0, 20)  # Grille d'alphas à tester
n_fixed = 5000        # Taille fixe de l'échantillon
n_iter_norm = 300     # Nombre d'itérations (réplicats) pour l'estimation de la␣
 ↪distribution
x_test = 1.0          # Point d'évaluation (doit être dans le support de␣
 ↪l'exponentielle, ici x >= 0)
true_density = np.exp(-x_test)  # Densité théorique de l'exponentielle (λ = 1)␣
 ↪en x_test

# Dictionnaire des estimateurs (ils doivent être déjà définis dans le notebook)
estimators = {
    'Parzen': parzen_estimator,
    'WWY': estimator_WWY,
    'WD': estimator_WD,
    'Révész': estimator_R_iterative
}
```

```python
# Wrapper pour garantir une sortie scalaire pour un x donné
def safe_estimator(estimator, x, sample, **kwargs):
    res = estimator(np.array([x]), sample, **kwargs)
    return res[0] if isinstance(res, np.ndarray) else res

# Dictionnaire pour stocker l'alpha optimal pour chaque estimateur
optimal_alphas = {}

# Création de la figure avec un subplot par estimateur
fig, axs = plt.subplots(2, 2, figsize=(12, 10))
axs = axs.flatten()

for idx, (name, estimator) in enumerate(estimators.items()):
    ks_stats = []
    # Pour chaque valeur d'alpha, on évalue la normalité asymptotique
    for alpha in alpha_values:
        estimates = []
        for _ in range(n_iter_norm):
            # Génération d'un échantillon de la loi exponentielle (scale=1 <=>␣
↪λ=1)
            sample = np.random.exponential(scale=1, size=n_fixed)
            est = safe_estimator(estimator, x_test, sample, alpha=alpha)
            estimates.append(est)
        estimates = np.array(estimates)
        # Calcul de l'erreur d'estimation (différence par rapport à la densité␣
↪théorique)
        errors = estimates - true_density
        mean_error = np.mean(errors)
        std_error = np.std(errors)
        # Standardisation (si std_error > 0)
        if std_error == 0:
            errors_std = errors
        else:
            errors_std = (errors - mean_error) / std_error

        # Test KS de normalité (comparaison avec une loi N(0,1))
        ks_stat, _ = st.kstest(errors_std, 'norm')
        ks_stats.append(ks_stat)

    ks_stats = np.array(ks_stats)
    # L'alpha optimal minimise la statistique KS
    opt_alpha = alpha_values[np.argmin(ks_stats)]
    optimal_alphas[name] = opt_alpha

    ax = axs[idx]
    ax.plot(alpha_values, ks_stats, 'o-', label='KS statistic')
```

```
    ax.set_title(f"{name} (alpha optimal = {opt_alpha:.2f})")
    ax.set_xlabel("alpha")
    ax.set_ylabel("Statistique KS")
    ax.legend()
    ax.grid(True)

plt.tight_layout()
plt.show()

# Affichage des alphas optimaux pour chaque estimateur
for est_name, opt_a in optimal_alphas.items():
    print(f"Optimal alpha for {est_name}: {opt_a:.2f}")
```




```
Optimal alpha for Parzen: 0.40
Optimal alpha for WWY: 0.10
Optimal alpha for WD: 0.80
Optimal alpha for Révész: 0.60
```

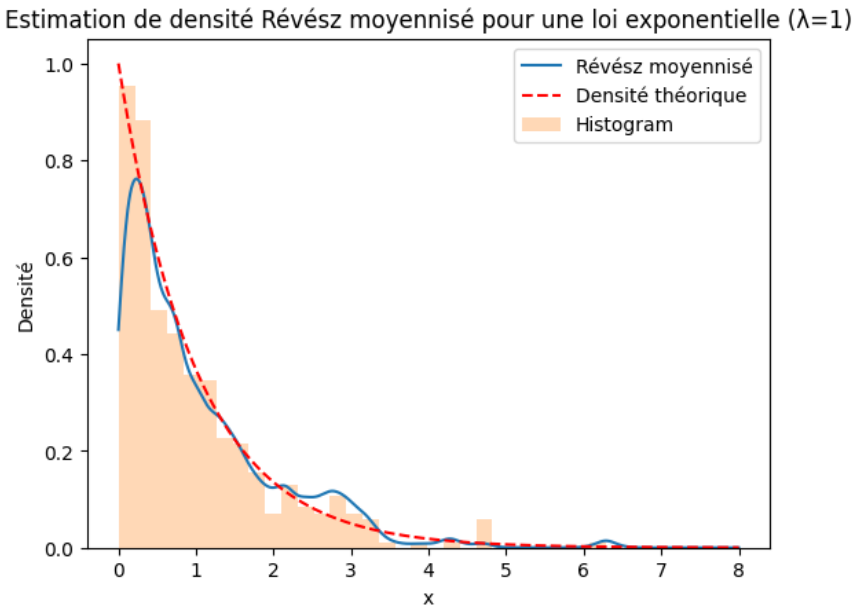
2 Passons à l'algorithme de Revesz moyénnisé

[13]:
```
def revesz_averaged_estimator(x, sample, gamma=0.395, sigma=1.0, alpha=0.6):
    """
    Estimateur de densité Révész moyennisé au point x.

    Pour un échantillon de taille n, on définit pour k = 1,...,n :
```

```
    - La fenêtre h_k = window(k, alpha) = k^(-alpha)
    - L'estimation de densité sur les k premières observations :
        f_k(x) = (1/(k * h_k)) * sum_{i=1}^{k} gaussian_kernel((x - X_i)/
↪h_k, sigma)
    - Le poids c_k = 1 / k^gamma

  L'estimateur final est la moyenne pondérée :
      f_n(x) = (sum_{k=1}^{n} c_k * f_k(x)) / (sum_{k=1}^{n} c_k)

  Args:
      x (float): Point d'évaluation.
      sample (array-like): Échantillon de données.
      gamma (float): Paramètre de pondération (défaut = 0.0).
      sigma (float): Paramètre du noyau gaussien (défaut = 1.0).
      alpha (float): Paramètre pour la fenêtre, via h_k = k^(-alpha) (défaut =␣
↪0.2).

  Returns:
      float: Estimation de la densité en x.
  """
  n = len(sample)
  if n == 0:
      return 0.0

  # Calcul des poids c_k et de leur somme
  c_vals = np.array([1.0 / (k ** gamma) for k in range(1, n + 1)])
  C_n = np.sum(c_vals)

  f_k_values = []
  for k in range(1, n + 1):
      h_k = window(k, alpha)
      sample_k = sample[:k]  # Les k premières observations
      u = (x - sample_k) / h_k
      # Application du noyau gaussien
      kernel_vals = gaussian_kernel(u, sigma)
      # Estimation de densité avec k observations
      f_k = np.sum(kernel_vals) / (k * h_k)
      f_k_values.append(f_k)

  f_k_values = np.array(f_k_values)
  # Moyenne pondérée des f_k(x) avec les poids c_k
  return np.sum(c_vals * f_k_values) / C_n
```

[14]:
```
# Génération d'un échantillon issu d'une loi exponentielle (λ=1)
sample = np.random.exponential(scale=1, size=400)

# Définition de la grille d'évaluation (support de l'exponentielle : x  0)
```

```
x_grid = np.linspace(0, 8, 1000)

# Mesure du temps pour le calcul de l'estimation de densité Révész moyennisé
start_time = time.perf_counter()
density = np.array([revesz_averaged_estimator(x, sample, gamma=0.3, sigma=1.0,␣
 ↪alpha=0.4) for x in x_grid])
end_time = time.perf_counter()

# Tracé de la densité estimée, de la densité théorique et de l'histogramme de␣
 ↪l'échantillon
plt.plot(x_grid, density, label="Révész moyennisé")
plt.plot(x_grid, np.exp(-x_grid), "r--", label="Densité théorique")  # Densité␣
 ↪théorique de l'exponentielle (λ=1)
plt.hist(sample, bins=30, density=True, alpha=0.3, label="Histogram")
plt.xlabel("x")
plt.ylabel("Densité")
plt.title("Estimation de densité Révész moyennisé pour une loi exponentielle␣
 ↪(λ=1)")
plt.legend()
plt.savefig("revesz_moyennise_exponentielle.png", dpi=300, bbox_inches="tight")
plt.show()


print(f"Temps Révész moyennisé: {end_time - start_time:.4f} secondes")
```

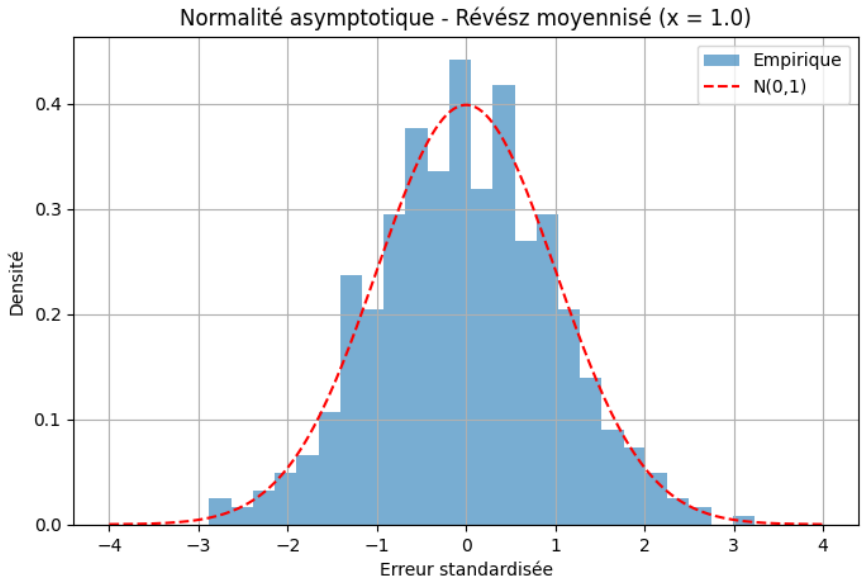


Temps Révész moyennisé: 5.7316 secondes

Le temps d'éxecution est relativement lent comparé aux précedents estimateurs

Regardons la normalité asymptotique de cet algorithme

[15]:
```
# Configuration pour l'étude de la normalité asymptotique avec l'estimateur␣
↪Révész moyennisé
n_fixed = 2000        # Taille fixe de l'échantillon
n_iter = 500          # Nombre de réplicats
x_test = 1.0          # Point d'évaluation (doit être dans le support de␣
↪l'exponentielle, ici x  0)
true_density = np.exp(-x_test)  # Densité théorique de l'exponentielle (λ = 1),␣
↪f(x)=exp(-x)

# Calcul des erreurs d'estimation sur n_iter réplicats
errors = []
for _ in range(n_iter):
    sample = np.random.exponential(scale=1, size=n_fixed)
    est = revesz_averaged_estimator(x_test, sample, gamma=0.3, sigma=1.0,␣
↪alpha=0.4)
    errors.append(est - true_density)
errors = np.array(errors)

# Standardisation des erreurs
mean_error = np.mean(errors)
std_error = np.std(errors)
errors_std = (errors - mean_error) / std_error if std_error > 0 else errors

# Visualisation de la distribution des erreurs standardisées
plt.figure(figsize=(8, 5))
plt.hist(errors_std, bins=25, density=True, alpha=0.6, label="Empirique")
x_vals = np.linspace(-4, 4, 200)
plt.plot(x_vals, norm.pdf(x_vals), 'r--', label="N(0,1)")
plt.xlabel("Erreur standardisée")
plt.ylabel("Densité")
plt.title("Normalité asymptotique - Révész moyennisé (x = {})".format(x_test))
plt.legend()
plt.grid(True)
plt.show()
```

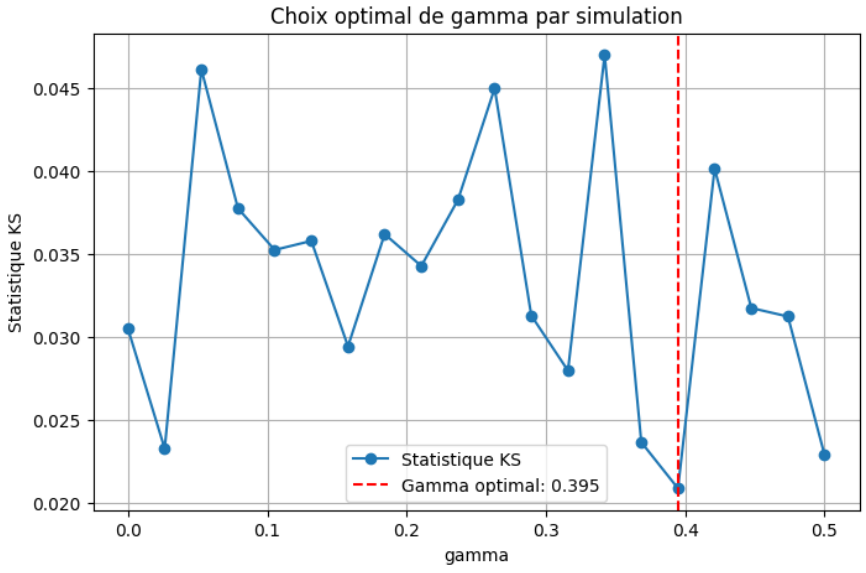


Cet algorithme vérifie bien la normalité asymptotique

2.1 Quelle valeur de gamma permet de minimiser l'erreur dans notre cas?

[16]:
```
# Grille de valeurs candidate pour gamma
gamma_values = np.linspace(0.0, 0.5, 20)
n_fixed = 2000        # Taille fixe de l'échantillon
n_iter = 500          # Nombre de réplicats
x_test = 1.0          # Point d'évaluation (pour l'exponentielle, x  0)
true_density = np.exp(-1)  # Densité théorique de l'exponentielle (λ=1) en x_test

ks_stats = []  # Pour stocker la statistique KS pour chaque gamma

for gamma in gamma_values:
    errors = []
    for _ in range(n_iter):
        # Générer un échantillon issu de la loi exponentielle (scale=1␣
↪correspond à λ=1)
        sample = np.random.exponential(scale=1, size=n_fixed)
        # Calcul de l'estimation de densité Révész moyennisé au point x_test␣
↪pour ce gamma
        est = revesz_averaged_estimator(x_test, sample, gamma=gamma, sigma=1.0,␣
↪alpha=0.4)
        errors.append(est - true_density)
    errors = np.array(errors)

    # Standardisation des erreurs
    mean_error = np.mean(errors)
    std_error = np.std(errors)
```

```
    if std_error > 0:
        errors_std = (errors - mean_error) / std_error
    else:
        errors_std = errors

    # Test de Kolmogorov-Smirnov pour comparer aux N(0,1)
    ks_stat, _ = st.kstest(errors_std, 'norm')
    ks_stats.append(ks_stat)

ks_stats = np.array(ks_stats)
optimal_gamma = gamma_values[np.argmin(ks_stats)]

plt.figure(figsize=(8,5))
plt.plot(gamma_values, ks_stats, 'o-', label="Statistique KS")
plt.axvline(optimal_gamma, color='r', linestyle='--',
            label=f"Gamma optimal: {optimal_gamma:.3f}")
plt.xlabel("gamma")
plt.ylabel("Statistique KS")
plt.title("Choix optimal de gamma par simulation")
plt.legend()
plt.grid(True)
plt.show()

print(f"gamma optimal: {optimal_gamma:.3f}")
```

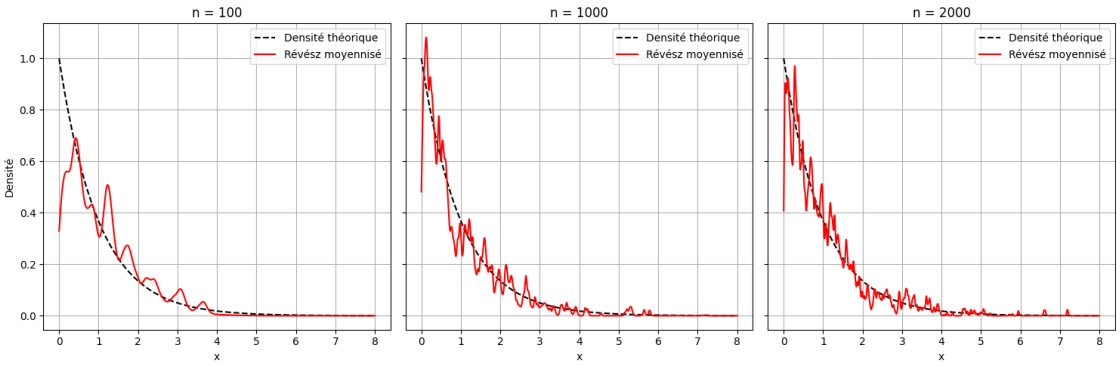



```
Optimal gamma: 0.395
```

La statistique de Kolmogorov-Smirnov donne un gamma optimal à 0.4.

Voyons si la convergence presque sûre peut être vérifiée visuellement pour cet algorithme.

[17]:
```
# Définition des tailles d'échantillon et de la grille d'évaluation
sample_sizes = [100, 1000, 2000]
x_grid = np.linspace(0, 8, 1000)

# Densité théorique de l'exponentielle (λ = 1)
true_density = np.exp(-x_grid)

# Création de la figure
fig, axs = plt.subplots(1, len(sample_sizes), figsize=(15, 5), sharey=True)

for i, n in enumerate(sample_sizes):
    # Génération d'un échantillon
    sample = np.random.exponential(scale=1, size=n)
    ax = axs[i]

    # Calcul de l'estimation Révész moyennisée
    estimated_curve = np.array([revesz_averaged_estimator(x, sample, gamma,␣
 ↪sigma, alpha)
                                for x in x_grid])

    # Tracés
    ax.plot(x_grid, true_density, 'k--', label='Densité théorique')
    ax.plot(x_grid, estimated_curve, 'r-', label='Révész moyennisé')

    # Mise en forme
    ax.set_title(f"n = {n}")
    ax.set_xlabel("x")
    if i == 0:
        ax.set_ylabel("Densité")
    ax.legend()
    ax.grid(True)

plt.tight_layout()
plt.savefig("revesz_moyennise_only.png", dpi=300, bbox_inches="tight")
plt.show()
```



On remarque bien la convergence presque sure lorsque n croit