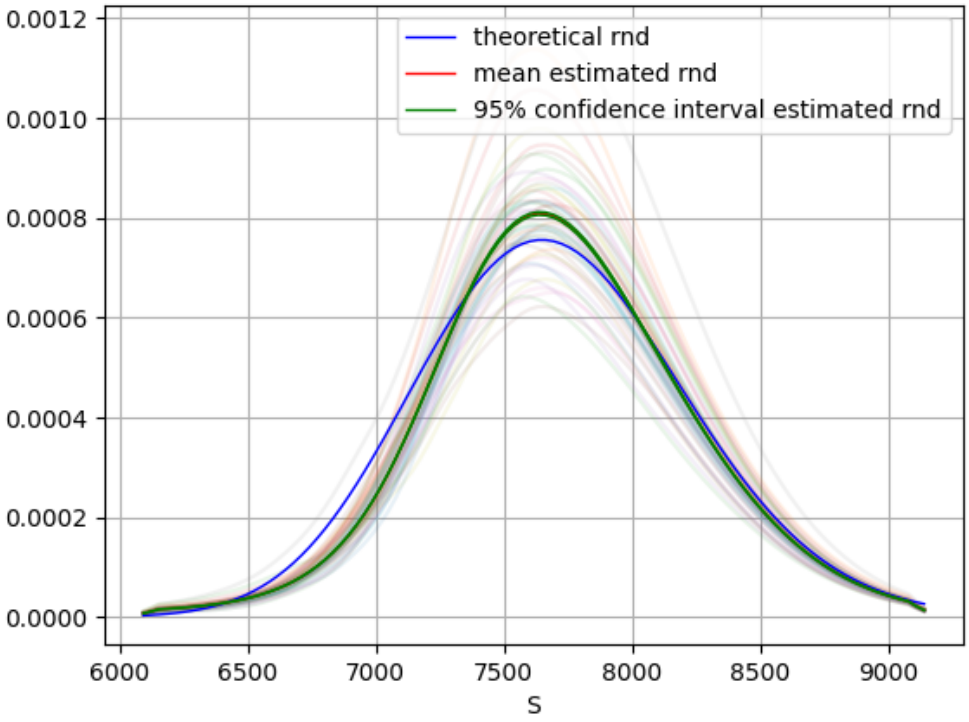
RAPPORT DE PROJET



Comparaison de méthodes d'estimation des densités risque-neutre à partir des prix options

Paul-Antoine LEVEILLEY, Ayman BOUHSS & Georgios KAZILIS,

Projet réalisé dans le cadre du cours [Python/Pytorch Project] dispensé par M. Julien CLAISSE pour le M2 intitulé [Mathématiques de l'Assurance de l'Economie et de la Finance] de l'Université Paris Dauphine - PSL

18 Avril 2025

Dauphine | PSL
UNIVERSITÉ PARIS

Table des matières

1 Introduction

Dans ce projet, nous essayons de reproduire les résultats de [7]. Le but de [7] est de comparer les performances de trois méthodes non paramétriques pour l'estimation de densité de probabilité risque-neutre à partir du prix d'options d'achat. Plus précisément, dans le cadre de l'évaluation risque-neutre des produits dérivés, le prix d'une option est donné par l'espérance sous la probabilité risque-neutre de son *payoff* actualisé. En dérivant deux fois par rapport au *strike*, on peut donc retrouver la densité de probabilité risque-neutre, à un facteur d'actualisation près. Une méthode pour estimer la densité de probabilité risque-neutre est donc d'observer les prix des options sur le marché, pour différents *strikes*, de tracer une courbe qui passe par ces points et de dériver deux fois par rapport au strike. Ceci pose plusieurs problèmes. Premièrement, les prix des *calls* sur les marchés sont disponibles seulement pour un nombre fini de *strikes*. Deuxièmement, les prix des options sont sujets à l'aléa du *trading*. Ces problèmes font qu'il n'est pas suffisant d'observer les prix des *calls* sur le marché pour estimer la densité de probabilité risque-neutre. Pour remédier à cela, il existe des méthodes d'interpolation. [7] étudie trois : la méthode des splines cubiques, la régression à noyau et les *radial basis neural network*. Le principe est le suivant. Les prix des *calls* sont convertis en volatilité implicite en inversant la formule de Black & Scholes, puis une des trois méthodes précédentes est utilisée pour interpoler la volatilité implicite (en fonction de S/K). Une fois la courbe interpolante calculée, elle est retransformée en prix de *call*, en appliquant la formule de Black & Scholes. Enfin, la courbe de prix de *calls* obtenue est dérivée deux fois pour obtenir la densité de probabilité risque-neutre. Le problème est qu'en pratique dans un marché, la "vraie" densité de probabilité risque-neutre n'est pas connue. Donc il n'y a aucun moyen de comparer la prédiction du modèle à la réalité. Dans [7], l'auteur propose de fixer un modèle, dont on connaît et on peut calculer la densité de probabilité risque-neutre, l'utiliser pour générer des prix de *calls* et d'appliquer les méthodes d'interpolation pour estimer la densité risque-neutre. Ainsi, on peut comparer le résultat donné par l'interpolation à la vraie densité de probabilité risque-neutre. [7] teste 3 modèles différents : le modèle de Black & Scholes [2], le modèle de Heston [5] et le modèle de Bakshi [1]. Dans ces trois modèles, il existe des formules

fermées pour le prix des *calls* et pour les densités de probabilité risque-neutre. Il est donc possible à la fois de simuler des prix et aussi de calculer des densités de probabilité risque-neutre.

2 Méthodologie

Dans cette sections nous expliquons en détail la méthodologie utilisée dans [7] pour générer les données, faire l'interpolation, calculer une estimation de la densité de probabilité risque-neutre et mesurer la qualité de l'estimation.

2.1 Génération des données

Pour des détails sur les modèles utilisés, nous renvoyons à [7],[2],[5] et [1]. La méthodologie de génération des données est la suivante. Premièrement, on fixe un horizon temporel T d'un an et un *spot* initial donné de 7616 (voir [7] paragraphe 4.1 pour la raison de ce choix). Ensuite, on utilise des simulations de Monte-Carlo pour générer 5000 réalisations du spot S_T (et de la volatilité stochastique pour les modèles de Heston et de Bakshi). On fait ceci pour mimer ce qu'il se passerait dans la vraie vie si un acteur souhaiterait estimer la densité probabilité risque-neutre. Ensuite pour chacune des valeurs générées du *spot*, on crée une grille de taille 50 de *strikes* qui va de 80% du spot à 120% du spot. On utilise la formule de *pricing* du modèle pour calculer les prix des *calls* sur cette grille de strikes, tous les autres paramètres étant fixés. Une fois les prix calculés, on leur ajoute un bruit gaussien de moyenne 0 et d'écart type 5% du prix du *call*. Enfin, on convertit ces prix de *calls* en volatilité implicite, en inversant la formule de Black & Scholes. On se retrouve alors avec 5000 jeux de 50 valeurs de volatilité implicite sur une grille de *strikes*.

TABLE 1 – Paramètres des processus de génération de données pour la simulation de Monte-Carlo

Paramètre	**BS**	**SV (Heston)**	**SVJ (Bakshi)**
S_0 (spot initial)	7616	7616	7616
r_f (taux sans risque)	0,045	0,045	0,045
δ (dividende)	0	0	0
σ (vol. constante)	0,13764	—	—
σ_v (vol. de la vol.)	—	0,20	0,20
κ (vitesse de retour)	—	9	9
θ_v (moyenne à long terme)	—	0,0189	0,0189
V_0 (variance initiale)	—	0,01	0,01
ρ (corrélation)	—	0,10	0,10
λ (taux de saut)	—	—	0,59
μ_j (taille moyenne des sauts)	—	—	$-0,05$
σ_j (écart type des sauts)	—	—	0,07

2.2 Méthodes d'interpolation

Une fois les 5000 jeux de 50 valeurs de volatilité implicite générés, on utilise une des trois techniques d'interpolation ci-dessous pour reconstituer la courbe de volatilité implicite en fonction de S_T/K.

2.2.1 Splines cubiques

La version des splines cubiques utilisée dans [7] est celle des splines cubiques lissantes [8]. Soit un jeu de données $(x_i, y_i)_{i=1\cdots,n}$, associés à des poids positifs ω_i et à un paramètre de lissage

λ, la spline cubique lissante est la solution du problème de minimisation :

$$\min_{f \in W_2^2([a,b])} \sum_{i=1}^{n} \omega_i (y_i - f(x_i))^2 + \lambda \int_a^b (f''(x))^2 dx$$

Les poids permettent de donner plus ou moins d'importance à certains points du jeu de données et le paramètre λ contrôle le compromis entre qualité de l'interpolation et lissité de la fonction interpolante. Pour notre cas d'usage, comme dans [7] et [3], nous utilisons le vega $\mathcal{V} = \frac{\partial C^{BS}}{\partial \sigma}$ de l'option pour pondérer les splines cubiques. Ceci a l'effet de donner plus de poids aux options à la monnaie. Ceci est cohérent d'un point de vue économique car les options à la monnaie sont les plus liquides donc leur prix est celui qui représente le mieux la dynamique risque-neutre du sous-jacent. Et d'un point de vue numérique, c'est aussi plus logique de donner plus d'importance aux options a la monnaie car la densité de probabilité risque-neutre est de toute façon très petite quand on s'éloigne de la monnaie.

2.2.2 Régression à noyau

Construction Supposons que nous disposions d'un ensemble d'observations $(X_i, Y_i)_{i \in [\![1,n]\!]}$ et que le lien entre X et Y soit donné par la relation $Y_i = f(X_i) + \epsilon_i \, , \forall i \in [\![1,\, n]\!]$, où les $(X_i)_{i \in [\![1,n]\!]}$ sont i.i.d. et de densité g, les $(\epsilon_i)_{i \in [\![1,n]\!]}$ sont centrés avec une variance σ_ϵ^2 et $(X_i) \perp\!\!\!\perp (\epsilon_i)$. Maintenant, si nous voulons estimer f, sur la base de ces hypothèses, nous avons :

$$f(x) = \mathbb{E}[Y|X=x] = \int y f_{Y|X}(x,y)\, dy = \frac{\int y f_{X,Y}(x,y)\, dy}{g(x)}, \quad \text{si } g(x) \neq 0,$$

où $f_{Y|X}$ est la fonction de distribution de probabilité conditionnelle de Y étant donné X et $f_{X,Y}$ est la fonction de distribution de probabilité conjointe de X et Y. Comme $f_{X,Y}$ et g sont des densités, nous pouvons réaliser notre objectif d'estimer f en estimant ces densités. L'une des méthodes possibles consiste à utiliser des estimateurs à noyau. Concrètement, l'estimation par noyau de g, par exemple, est donnée par :

$$\hat{g}(x) = \frac{1}{nh} \sum_{i=1}^{n} K\left(\frac{x - X_i}{h}\right) \quad \text{avec } K \text{ une fonction telle que } \int K(u)\, du = 1.$$

Ici, K et h sont deux paramètres nommés respectivement noyau et fenêtre. De même, l'estimateur à noyau pour 2 dimensions de $f_{X,Y}$, par exemple, est donné par :

$$\hat{f}_{X,Y}(x,y) = \frac{1}{nh'h''} \sum_{i=1}^{n} K'\left(\frac{x - X_i}{h'}\right) K''\left(\frac{y - Y_i}{h''}\right) \quad \text{avec } K', K'', h' \text{ et } h'' \text{ sont les paramètres respectifs.}$$

Ensuite, si K'' est symétrique, on a :

$$\frac{1}{h''} \int y K''\left(\frac{y - Y_i}{h''}\right) dy = \int (vh'' + Y_i) K''(v)\, dv = Y_i.$$

Ainsi, en remplaçant les estimateurs du noyau de densité ci-dessus par l'expression de la fonction f, nous obtenons l'estimation suivante, appelée estimateur du noyau de Nadaraya-Watson :

$$\hat{f}_{h,h'}(x) = \frac{\frac{1}{nh'} \sum_{i=1}^{n} K'\left(\frac{x-X_i}{h'}\right) Y_i}{\frac{1}{nh} \sum_{i=1}^{n} K\left(\frac{x-X_i}{h}\right)} = \sum_{i=1}^{n} \underbrace{\frac{\frac{1}{h'} K'\left(\frac{x-X_i}{h'}\right)}{\frac{1}{h} \sum_{j=1}^{n} K\left(\frac{x-X_j}{h}\right)}}_{=: W_{h,h',i}(x)} Y_i = \sum_{i=1}^{n} W_{h,h',i}(x) Y_i.$$

La question est maintenant de savoir comment choisir les paramètres K, K', h et h'. Pour des raisons de simplicité, nous choisissons la même fonction noyau $K = K'$ (mais, bien sûr, pour certaines applications, il peut être utile d'utiliser des fonctions noyau différentes pour la régression non paramétrique). Maintenant, en gardant à l'esprit que dans notre application, nous allons dériver deux fois une transformation non linéaire de la régression résultante, nous avons choisi d'utiliser le noyau gaussien $K(u) = \frac{1}{\sqrt{2\pi}} e^{-\frac{u^2}{2}}$. Nous l'avons fait parce qu'il s'agit d'une fonction à support infini et qu'il a été observé que les noyaux à support fini (par exemple le noyau d'Epanechnikov) produisaient des dérivées très irrégulières. Ensuite, nous devons rechercher un choix optimal des paramètres des fenêtres. Là encore, pour des raisons de simplicité, nous avons décidé d'effectuer une optimisation 1D (c'est-à-dire pour $h = h'$), alors qu'il existe de nombreux articles sur la manière d'effectuer une optimisation 2D (c'est-à-dire pour $h \neq h'$). Dans la partie suivante, nous discuterons d'une telle procédure.

Validation croisée Comme expliqué dans [4, p. 158], l'une des méthodes possibles pour trouver la valeur optimale de h est la validation croisée, et plus précisément la *leave-one-out-cross-validation* des moindres carrés. Concrètement, dans notre cas, comme $h = h'$, notons l'estimateur de Nadaraya-Watson de f par $\hat{f}_h(x) = \sum_{i=1}^n W_{h,i}(x) Y_i$, où $W_{h,i}(x) = \frac{K\left(\frac{x-X_i}{h}\right)}{\sum_{j=1}^n K\left(\frac{x-X_j}{h}\right)}$. Ensuite, en dénotant $\hat{f}_{h,-i}(x) = \sum_{\substack{j=1 \\ j\neq i}}^n W_{h,j,-i}(x) Y_j$, où $W_{h,j,-i}(x) = \frac{K\left(\frac{x-X_j}{h}\right)}{\sum_{\substack{k=1 \\ k\neq i}}^n K\left(\frac{x-X_k}{h}\right)}$, la validation croisée *leave-one-out* consiste à minimiser le critère $CV(h) := \frac{1}{n} \sum_{i=1}^n \left(Y_i - \hat{f}_{h,-i}(X_i)\right)^2$ sur un ensemble fini de valeurs possibles de h, **h**. Mathématiquement, la valeur optimale de h selon ce critère est $\hat{h}_{CV} := argmin_{h \in \mathbf{h}} CV(h)$.

Supposons que nous disposions d'un ensemble de volatilités implicites simulées et que nous souhaitions appliquer l'estimateur Nadaraya-Watson à cet ensemble de données. En calculant, donc, la valeur de la fonction objective de ce critère pour une grille de fenêtres prédéfinie, on choisira la valeur minimisante comme le montre la figure suivante :

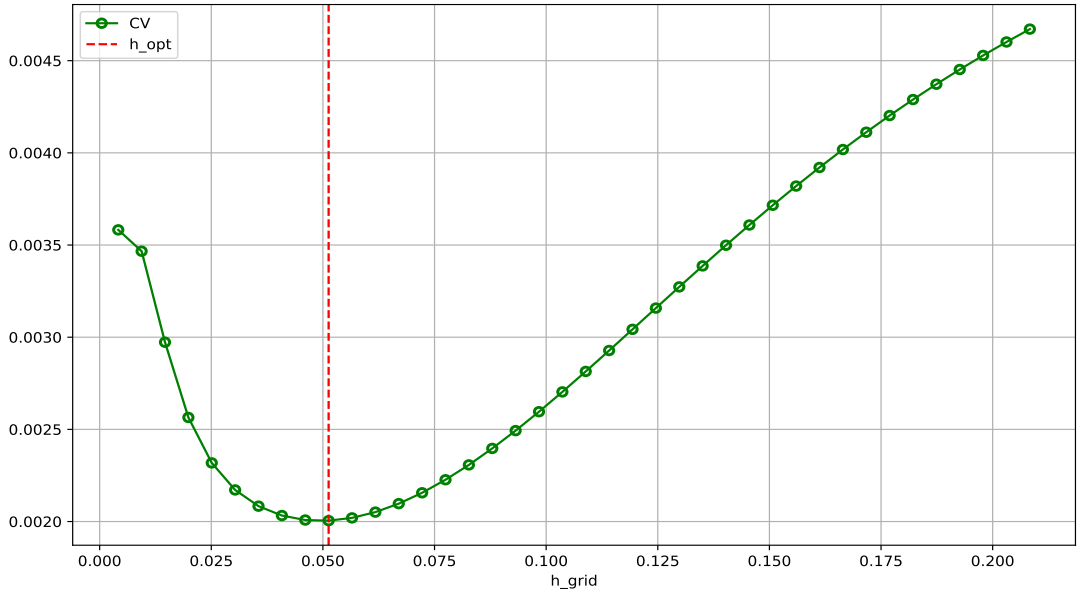


FIGURE 1 – Valeurs de la fonction objective par rapport à la grille de fenêtres

En appliquant l'estimateur Nadaraya-Watson avec cette valeur optimale de la fenêtre et avec 2 autres valeurs, nous obtenons les régressions suivantes :

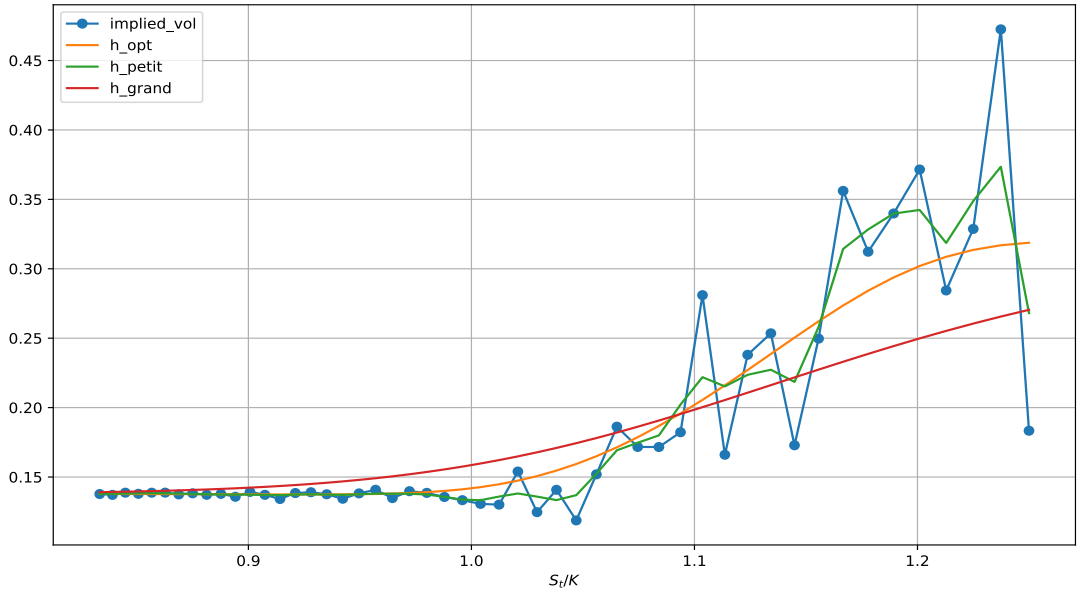


FIGURE 2 – Influence de la fenêtre sur la régression du noyau

Nous observons que plus la valeur de la fenêtre est élevée, plus la courbe résultante est lisse et plus la valeur est petite, plus la régression colle aux données d'entrée. Ainsi, nous comprenons qu'il existe à nouveau un compromis biais-variance et que le choix de la fenêtre est notre contrôle. Nous devons également observer que le problème d'optimisation heuristique de la validation croisée peut être très coûteux en termes de calcul, car il faut effectuer n régressions pour l'évaluation de la fonction objective à une seule fenêtre. Ceci est très problématique pour nous si nous gardons à l'esprit notre objectif principal, qui est d'effectuer 5000 régressions non paramétriques pour chaque modèle et pour chaque maturité. Nous pouvons résoudre ce problème en utilisant une proposition [4, Prop. 5.1, p. 158] qui prouve que nous pouvons évaluer la fonction objective à une seule fenêtre en effectuant une seule régression.

Proposition : Les poids de l'estimateur *leave-one-out* $\hat{f}_{h,-i}(x) = \sum_{\substack{j=1 \\ j\neq i}}^{n} W_{h,j,-i}(x)Y_j$ peuvent être obtenus à partir de $\hat{f}_h(x) = \sum_{i=1}^{n} W_{h,i}(x)Y_i$ de la manière suivante :

$$W_{h,j,-i}(x) = \frac{W_{h,j}(x)}{\sum_{\substack{k=1 \\ k\neq i}}^{n} W_{h,k}(x)} = \frac{W_{h,j}(x)}{1 - W_{h,i}(x)}$$

Cela implique que

$$CV(h) := \frac{1}{n}\sum_{i=1}^{n}\left(\frac{Y_i - \hat{f}_h(X_i)}{1 - W_{h,i}(X_i)}\right)^2.$$

Preuve : On a

$$W_{h,j,-i}(x) = \frac{K\left(\frac{x-X_j}{h}\right)}{\sum_{\substack{k=1 \\ k\neq i}}^{n} K\left(\frac{x-X_k}{h}\right)} = \frac{\overbrace{\frac{K\left(\frac{x-X_j}{h}\right)}{\sum_{l=1}^{n} K\left(\frac{x-X_l}{h}\right)}}^{=W_{h,j}(x)}}{\sum_{\substack{k=1 \\ k\neq i}}^{n} \underbrace{\frac{K\left(\frac{x-X_k}{h}\right)}{\sum_{l=1}^{n} K\left(\frac{x-X_l}{h}\right)}}_{=W_{h,k}(x)}} = \frac{W_{h,j}(x)}{\sum_{\substack{k=1 \\ k\neq i}}^{n} W_{h,k}(x)}.$$

Or,

$$1 - W_{h,i}(x) = 1 - \frac{K\left(\frac{x-X_i}{h}\right)}{\sum_{k=1}^{n} K\left(\frac{x-X_k}{h}\right)} = \frac{\sum_{\substack{k=1 \\ k\neq i}}^{n} K\left(\frac{x-X_k}{h}\right)}{\sum_{k=1}^{n} K\left(\frac{x-X_k}{h}\right)} = \sum_{\substack{k=1 \\ k\neq i}}^{n} \frac{K\left(\frac{x-X_k}{h}\right)}{\sum_{l=1}^{n} K\left(\frac{x-X_l}{h}\right)} = \sum_{\substack{k=1 \\ k\neq i}}^{n} W_{h,k}(x).$$

Du coup, on trouve, en effet,

$$W_{h,j,-i}(x) = \frac{W_{h,j}(x)}{1 - W_{h,i}(x)}.$$

Ainsi,

$$CV(h) = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{f}_{h,-i}(X_i) = \frac{1}{n}\sum_{i=1}^{n}\left(Y_i - \sum_{\substack{j=1\\j\neq i}}^{n} W_{h,j,-i}(X_i)Y_j\right)^2 = \frac{1}{n}\sum_{i=1}^{n}\left(Y_i - \sum_{\substack{j=1\\j\neq i}}^{n} \frac{W_{h,j}(X_i)}{1 - W_{h,i}(X_i)}Y_j\right)^2$$

$$= \frac{1}{n}\sum_{i=1}^{n}\left(\frac{\overbrace{Y_i - W_{h,i}(X_i)Y_i - \sum_{\substack{j=1\\j\neq i}}^{n} W_{h,j}(X_i)Y_j}^{=-\hat{f}_h(X_i)}}{1 - W_{h,i}(X_i)}\right)^2 = \frac{1}{n}\sum_{i=1}^{n}\left(\frac{Y_i - \hat{f}_h(X_i)}{1 - W_{h,i}(X_i)}\right)^2.$$

□

Règle finale appliquée Il est très important de noter que ce qui a été présenté jusqu'à ce point reste le plus pertinent lorsque la régression non paramétrique est elle-même l'objectif final. Nous rappelons que dans notre étude, la régression non paramétrique est une étape importante mais intermédiaire. Plus en détail, la régression résultante sera transformée de manière non-linéaire par la formule de Black-Scholes et sera ensuite dérivée 2 fois. Ceci étant dit, notre but ultime devrait être traduit dans l'algorithme pour qu'il privilégie des régressions plus lisses. Bien sûr, on comprend qu'il s'agit d'un art en soi et qu'il pourrait s'agir d'un domaine de recherche pour ce type de régressions non paramétriques.

Dans notre cas, afin d'avoir un algorithme qui fonctionne et qui produit des résultats assez satisfaisants, nous avons cherché une autre règle afin d'avoir des résultats plus proches de ceux de l'article [7] (en termes de forme des distributions RND estimées). A partir des tests effectués, nous avons conclu à l'utilisation de la règle dite de Silverman pour définir la valeur de la fenêtre et cette règle est la suivante : $h_{silverman} = 0.9n^{-1/5}min(\hat{\sigma}, \frac{I\hat{Q}R}{1.349})$, où $\hat{\sigma}$ est l'écart-type de (X_i) et $I\hat{Q}R$ est l'intervalle interquantile de (X_i) de nouveau.

2.2.3 Radial basis function neural network

Soit un jeu de données d'apprentissage

$$\{(X_i, Y_i)\}_{i=1}^{n},$$

où $X_i \in \mathbb{R}^d$ (ici $d = 1$) représente les variables explicatives le ratio S_T/K et $Y_i \in \mathbb{R}^+$ la volatilité implicite observée.

Architecture du réseau

Le réseau RBF que nous employons est constitué de trois couches :

Couche d'entrée On pose

$$x_i = \frac{S_T}{K_i}, \quad i = 1, \ldots, 50.$$

Couche cachée Soit $P \in \mathbb{N}^*$ le nombre de neurones. Chaque neurone p est caractérisé par

$$c_p \in \mathbb{R} \quad \text{(centre)}, \quad h_p > 0 \quad \text{(largeur)},$$

et la **fonction de base** gaussienne

$$\phi_p(x) = \exp\Big(-\tfrac{(x-c_p)^2}{h_p^2}\Big).$$

Les centres $\{c_p\}_{p=1}^P$ sont obtenus par `K-Means` (que nous détaillerons dans la partie suivante) sur les 50 valeurs de x, et chaque largeur h_p est choisie comme la distance moyenne aux autres centre

$$h_p = \frac{1}{P-1} \sum_{\substack{q=1 \\ q \neq p}}^{P} |c_p - c_q|.$$

Couche de sortie La sortie $\hat{\sigma}_{\text{imp}}(x)$ est une combinaison linéaire des activations, plus un biais

$$\hat{\sigma}_{\text{imp}}(x) = \sum_{p=1}^{P} w_p \, \phi_p(x) \; + \; w_0,$$

avec $(w_1, \ldots, w_P) \in \mathbb{R}^P$ et $w_0 \in \mathbb{R}$.

Clustering par `K-Means`

On note $K = P$. Pour positionner de manière optimale les centres des fonctions de base radiale (RBF) dans notre réseau, nous utilisons l'algorithme *K-Means*, un des plus classiques en apprentissage non supervisé. Il s'agit de partitionner un ensemble de points

$$\{x_i\}_{i=1}^n \subset \mathbb{R}^d$$

en K groupes (ou *clusters*) de sorte à minimiser la variance intra-cluster. Plus précisément, on cherche à résoudre le problème d'optimisation suivant :

$$\inf_{C_1,\ldots,C_K} \sum_{p=1}^{K} \sum_{x_i \in C_p} \big\|x_i - c_p\big\|_2^2,$$

où

- C_p est l'ensemble des indices des points affectés au p-ième cluster,
- $c_p = \frac{1}{|C_p|} \sum_{x_i \in C_p} x_i \in \mathbb{R}^d$ est le centroïde (ou centre de gravité) du cluster C_p,
- $\|\cdot\|_2$ désigne la norme euclidienne dans $\mathbb{R}^d$.

On résout cette minimisation par l'algorithme suivant, nommé *Lloyd* :

1. **Initialisation** : choisir K centres $c_1^{(0)}, \ldots, c_K^{(0)}$ (par exemple au hasard ou via la méthode KMeans++).
2. **Assignation** : pour chaque point x_i, déterminer le cluster le plus proche :

$$\text{affectation}(x_i) \; = \; \arg\min_{1 \leq p \leq K} \|x_i - c_p^{(t)}\|_2^2.$$

3. **Recentrage** : recalculez chaque centroïde comme la moyenne des points qui lui sont affectés :

$$c_p^{(t+1)} \; = \; \frac{1}{|C_p^{(t)}|} \sum_{x_i \in C_p^{(t)}} x_i.$$

4. Répéter les étapes 2 et 3 jusqu'à convergence (les centroïdes ne changent plus significativement).

Preuve : L'algorithme de Lloyd cherche à minimiser la *fonction de coût*

$$J(C_1, \dots, C_K;\, c_1, \dots, c_K) \;=\; \sum_{p=1}^{K} \sum_{x_i \in C_p} \big\| x_i - c_p \big\|_2^2,$$

où les ensembles de clusters C_p forment une partition de l'ensemble des points $\{x_i\}_{i=1}^n$, et où $c_p \in \mathbb{R}^d$ est le centroïde du p-ième cluster.

L'algorithme alterne deux étapes, et on montre que chacune ne fait qu'abaisser (ou maintenir égal) la valeur de J.

À itération t, on dispose de centres $\{c_p^{(t)}\}$. On construit alors la partition $C_1^{(t)}, \dots, C_K^{(t)}$ par

$$x_i \;\longmapsto\; p^*(i) \;=\; \arg\min_{1 \le p \le K} \; \|x_i - c_p^{(t)}\|_2^2.$$

Soit $J^{(t)} = J\big(C_1^{(t)}, \dots, C_K^{(t)};\, c_1^{(t)}, \dots, c_K^{(t)}\big)$. Comme chaque point est affecté au centre le plus proche, on minimise la somme des distances pour centres *fixes*, d'où

$$J\big(C_1^{(t)}, \dots, C_K^{(t)};\, c_1^{(t)}, \dots, c_K^{(t)}\big) \;\le\; J\big(C_1^{(t-1)}, \dots, C_K^{(t-1)};\, c_1^{(t)}, \dots, c_K^{(t)}\big).$$

On recalcule ensuite chaque centroïde par la moyenne des points qui lui sont affectés :

$$c_p^{(t+1)} \;=\; \frac{1}{|C_p^{(t)}|} \sum_{x_i \in C_p^{(t)}} x_i.$$

Pour une partition $C_1^{(t)}, \dots, C_K^{(t)}$ fixée, le choix du centroïde moyen est la *solution exacte* qui minimise $\sum_{x_i \in C_p^{(t)}} \|x_i - c_p\|_2^2$ en c_p. Dès lors,

$$J\big(C_1^{(t)}, \dots, C_K^{(t)};\, c_1^{(t+1)}, \dots, c_K^{(t+1)}\big) \;\le\; J\big(C_1^{(t)}, \dots, C_K^{(t)};\, c_1^{(t)}, \dots, c_K^{(t)}\big).$$

Ainsi, à chaque itération on a

$$J^{(t+1)} \;\le\; J^{(t)},$$

c'est-à-dire que la fonction de coût est décroissante.

Les partitions possibles de n points en K clusters sont au plus finies ($\le K^n$). L'algorithme ne peut pas strictement diminuer J plus de ce nombre de fois. Dès qu'il arrive à une configuration pour laquelle ni l'assignation, ni le recentrage ne font baisser J, les centres et clusters restent inchangés et l'algorithme converge.

□

Nous avons retenu $K = 5$ neurones cachés pour notre réseau RBF. En effet :

— Cinq centres suffisent à capturer la forme générale de la courbe de volatilité implicite (un trop grand nombre risquerait de sur-ajuster le bruit et d'augmenter l'erreur dans l'estimation).

— Ce choix provient d'expérimentations empiriques montrant qu'au-delà de 5 RBF, l'amélioration de la précision devient marginale par rapport à la complexité ajoutée.

Entraînement : régularisation et résolution

Pour chaque trajectoire de volatilités $\{y_i\}_{i=1}^{50}$, on résout le problème de *Ridge regression* pour déterminer les poids w et le biais w_0 :

$$\inf_{w\in\mathbb{R}^{\mathbb{P}},w_0\in\mathbb{R}} \left\|\Phi\, w + w_0\mathbf{1} - y\right\|_2^2 \;+\; \lambda\,\|w\|_2^2,$$

où

$$\Phi_{i,p} = \phi_p(x_i), \quad \Phi \in \mathbb{R}^{50\times P}, \quad y = (y_1,\dots,y_{50})^\top, \quad \lambda > 0 \text{ (paramètre de régularisation).}$$

La solution analytique se calcule via

$$\begin{pmatrix} w \\ w_0 \end{pmatrix} = \Big(\underbrace{\begin{bmatrix}\Phi & \mathbf{1}\end{bmatrix}^\top \begin{bmatrix}\Phi & \mathbf{1}\end{bmatrix}}_{=:A^\top A} + \lambda\,\mathrm{diag}(I_P, 0)\Big)^{-1} \begin{pmatrix}\Phi & \mathbf{1}\end{pmatrix}^\top y.$$

où $A^\top A$ est une matrice symétrique réelle donc diagonalisable par le théorème spectrale. Comme $\lambda > 0$ et les coefficients de A sont strictement positifs, $A^\top A + \lambda\,\mathrm{diag}(I_P, 0)$ est inversible car les coefficients diagonaux sont strictement positifs dans une base.

Preuve : Posons

$$A = \begin{bmatrix}\Phi & \mathbf{1}\end{bmatrix} \in \mathbb{R}^{n\times(P+1)}, \qquad \theta = \begin{pmatrix} w \\ w_0 \end{pmatrix} \in \mathbb{R}^{P+1}.$$

Alors le critère s'écrit

$$J(\theta) = \|A\,\theta - y\|_2^2 \;+\; \lambda\,\|w\|_2^2.$$

Introduisons la matrice de régularisation

$$R = \mathrm{diag}(\underbrace{1,\dots,1}_{P \text{ fois}}, 0) \;\in\; \mathbb{R}^{(P+1)\times(P+1)},$$

de sorte que $\theta^\top R\,\theta = w^\top w$.

On a

$$\|A\theta - y\|_2^2 = (A\theta - y)^\top(A\theta - y) = \theta^\top A^\top A\,\theta \;-\; 2\,y^\top A\,\theta \;+\; y^\top y.$$

Donc

$$J(\theta) = \theta^\top A^\top A\,\theta \;-\; 2\,y^\top A\,\theta \;+\; y^\top y \;+\; \lambda\,\theta^\top R\,\theta.$$

Regroupons les termes quadratiques :

$$J(\theta) = \theta^\top\big(A^\top A + \lambda\,R\big)\theta \;-\; 2\,y^\top A\,\theta \;+\; y^\top y.$$

Le gradient de J par rapport à θ est, notant que $A^\top A + \lambda R$ est symétrique :

$$\nabla_\theta J(\theta) = 2\,(A^\top A + \lambda\,R)\,\theta \;-\; 2\,A^\top y.$$

L'annulation du gradient donne l'équation normale :

$$(A^\top A + \lambda\,R)\,\theta = A^\top y.$$

La matrice $A^\top A + \lambda\,R$ est définie positive pour $\lambda > 0$, donc inversible. On en déduit la solution fermée :

$$\theta^\star = \begin{pmatrix} w^\star \\ w_0^\star \end{pmatrix} = \big(A^\top A + \lambda\,R\big)^{-1} A^\top y.$$

Revenant à la notation $A = [\Phi\ \mathbf{1}]$ et $R = \mathrm{diag}(I_P, 0)$, on obtient :

$$\boxed{\begin{pmatrix} w^\star \\ w_0^\star \end{pmatrix} = \Big(\underbrace{\begin{bmatrix}\Phi & \mathbf{1}\end{bmatrix}^\top \begin{bmatrix}\Phi & \mathbf{1}\end{bmatrix}}_{=:A^\top A} + \lambda\,\mathrm{diag}(I_P,\,0)\Big)^{-1} \begin{pmatrix}\Phi & \mathbf{1}\end{pmatrix}^\top y.}$$

□

Nous choisissons expérimentalement $\lambda = 10^{-1}$ car des tests ont montré que ce choix empêche le sur-ajustement tout en conservant la flexibilité du modèle.

2.3 Utilisation pour l'estimation de la RND

Une fois $\hat{\sigma}_{\text{imp}}(x)$ obtenu, on reconstitue le prix de *call* par la formule fermée de Black-Scholes :

$$\widehat{C}(K) = C_{\text{BS}}\big(\hat{\sigma}_{\text{imp}}(S/K),\, K\big),$$

et la densité risque-neutre par la formule de Breeden–Litzenberger :

$$f_{\text{RN}}(K) = e^{rT}\,\frac{\partial^2 \widehat{C}}{\partial K^2}(K).$$

2.4 Comparaison des densités de probabilité

Pour tester la qualité de l'interpolation, comme dans [7], nous utilisons le test de Kolmogorov-Smirnov [9]. Ce test teste l'hypothèse nulle que la densité de probabilité estimée est bien la densité de probabilité risque-neutre du modèle. Sur ce point, [7] donne assez peu de précisions. Nous en reparlons dans la section suivante.

3 Resultats

Dans cette section, nous expliquons comment nous avons tenté de reproduire les résultats de [7].

3.1 Considérations pratiques

Quand nous avons implémenté la méthodologie décrite au paragraphe précédent, nous avons rencontré les difficultés suivantes.

— Premièrement, la formule du prix des *calls* dans le modèle de Bakshi, que l'on peut lire dans [7], est fausse car le terme E, comme les termes C et D, existe lui aussi en deux versions. La bonne formule peut être trouvée dans l'article original de Bakshi : [1]. (Et de manière générale, nous avons trouvé beaucoup de coquilles dans les formules dans la version de [7] qui nous a été transmise : formule 6 page 1841, première équation, colonne de droite a la page 1841, terme E dans la formule de Bakshi)

— Le calcul de la volatilité implicite est tout un art. Nous avons d'abord essayé d'implémenter la méthode de Newton pour inverser la formule de Black Scholes, mais nous sommes tombés sur les problèmes numériques pour les *calls* très dans la monnaie. Dans ce cas, la méthode de Newton ne convergeait pas. Nous avons fait des recherches et avons découvert que le calcul de la volatilité implicite est assez subtil. Finalement, nous avons utilisé la librairie python `py_vollib`, qui implémente la méthode expliquée dans [6].

— Nous trouvons que l'auteur de [7] ne donne pas suffisament de détails sur la manière dont il a réalisé les tests de Kolmogorov-Smirnov. Combien d'échantillons ? Quelle méthode a-t-il utilisé pour tirer selon la densité de probabilité estimée ? Comment a-t-il géré le fait que les densités estimées ne sont pas exactement des densités : elles ne somment pas exactement à un et ont parfois des valeurs négatives. Pour implémenter les tests de Kolmogorov-Smirnov, nous avons premièrement traité les densités estimées pour les rendre positives. Dans notre code, les densités estimées sont représentées sous forme de tableau de nombre représentant les différentes valeurs des la densité sur une grille de 100 valeurs du *spots* allant de $0.8S_T$ a $1.2S_T$. Pour échantilloner, nous avons fait comme si ce tableau était un vecteur de probabilités (nous l'avons donc renormalisé pour qu'il somme à 1), et nous avons utilisé la fonction `random.choice` de la bibliothèque `numpy` pour tirer des réalisations d'une variable aléatoire qui suit une loi discrète avec probabilités données par le vecteur de probabilité. Nous avons utilisé 100 échantillons pour le test de Kolmogorov-Smirnov.

— Enfin, le plus gros soucis que nous avons rencontré est le suivant. Lorsque l'on ajoute du bruit gaussien aux prix d'options, pour les options très dans la monnaie, il arrive que le bruit fasse passer le prix du call en dessous de sa valeur intrinsèque $(S - Ke^{-rT})_+$, ce qui rend alors le calcul de la volatilité implicite impossible. Or pour les options dans la monnaie, le prix de l'option est quasiment égal à la valeur intrinsèque, donc, essentiellement une fois sur deux, le calcul de la volatilité implicite est impossible. A notre avis, ce soucis aurait dû être mentionné par l'auteur de [7]. Nous avons contourné ce problème en continuant à simuler des valeurs de bruits jusqu'à ce que le prix du call soit au-dessus de sa valeur intrinsèque. Ceci a permis de faire les calculs de volatilité implicite mais malheureusement, cela introduit un biais à la hausse sur les prix des calls dans la monnaie.

3.2 Résultats

Dans cette section nous présentons une partie de nos résultats. L'ensemble de nos résultats peut être consulté dans le dossier `results` dans le dépot Github de notre projet. Nous présentons nos résultats de la manière suivante : dans une premier paragraphe nous montrons des exemples de prix de calls générés avec la ligne interpolante calculée, pour différentes maturités, différents modèles et différentes méthodes d'interpolation. (Dans l'enchainement logique de la méthodologie, l'interpolation des prix de calls vient après celle de la volatilité implicite mais nous les présentons sur la même figure dans un soucis de concision). Dans un deuxième paragraphe, nous présentons les volatilités implicites générées ainsi que les lignes interpolantes calculées. Puis, nous présentons les densitées de probabilité risque-neutre estimées en dérivant les prix de *calls*. Dans un dernier paragraphe, nous présentons le tableau des résultats des tests de Kolmogorov-Smirnov, donnant, la moyenne et l'écart type des p-valeurs des tests ainsi que le pourcentage de rejet de l'hypothèse nulle. Nous avons produits ces résultats pour toutes les combinaisons possibles de paramètrent qui de la forme

$$\begin{aligned}(\text{modèle}, \text{type d'interpolation}, \text{maturité}) \in &\{\text{Black Scholes,Heston, Bakshi}\} \\ &\times \{\text{Splines cubiques,Régression à noyau, Rbf network}\} \\ &\times \{1, 2, 3, 4, 5, 6\}\end{aligned}$$

La maturité est mesurée en mois.

3.2.1 Prix de *Calls*

Dans cette section, nous présentons les prix de *calls* que nous avons générés ainsi que la reconstruction de la courbe de prix obtenue après avoir interpolé la volatilité implicite et retransformé la courbe de la volatilité en prix de *calls*.

Configuration des graphiques

Dans chaque graphique :

— Les $+$ représentent les prix de calls simulés par Monte Carlo avec un bruit gaussien.

— La courbe rouge trace l'interpolation obtenue par la méthode considérée.

— L'axe horizontal est la variable normalisée K/S (strike sur spot), et l'axe vertical le prix du call.

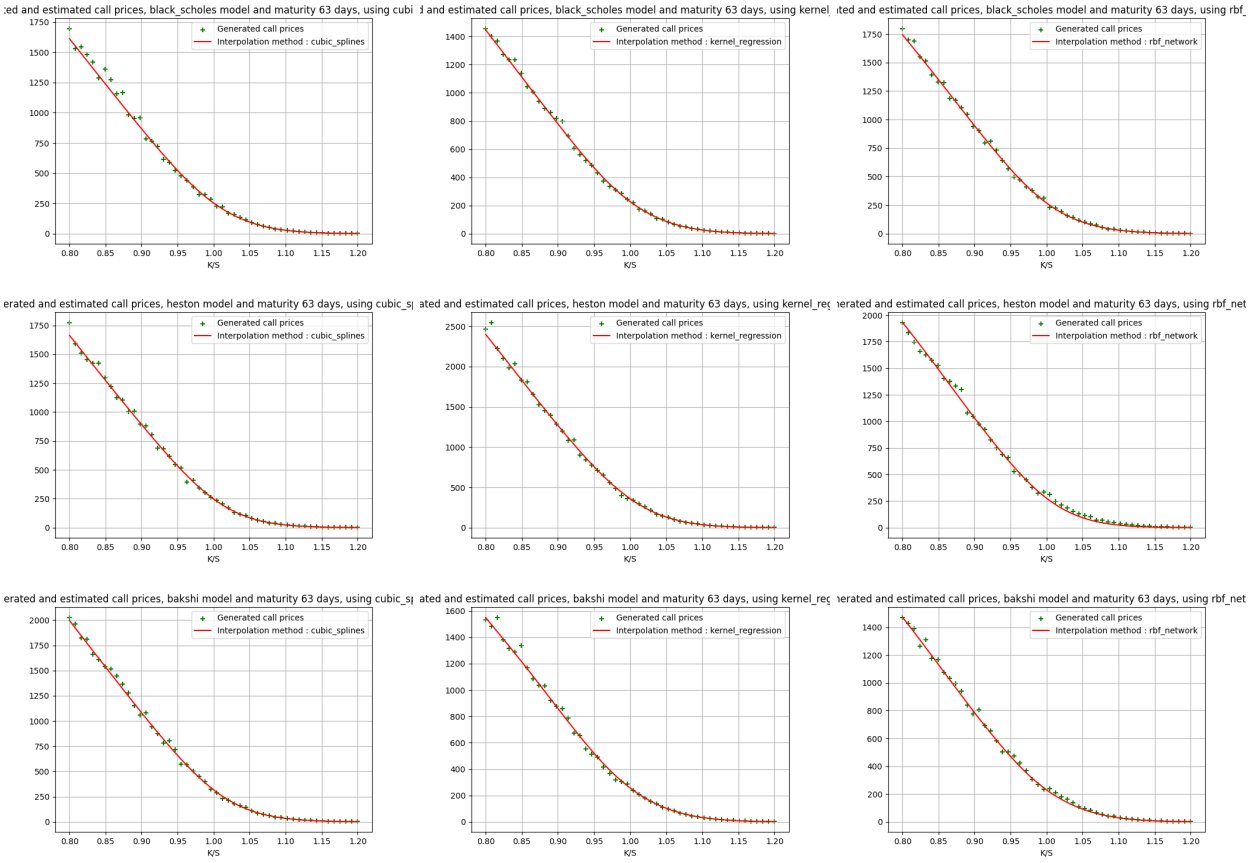


FIGURE 3 – Prix de *calls* générés et reconstruction par interpolation de la courbe de volatilité implicite. De haut en bas : modèle de Black-Scholes, modèle de Heston, modèle de Bakshi. De gauche a droite : splines cubiques, regression a noyau et *rbf neural network*. Maturité : 3 mois.

Analyse des résultats

Spline cubique. L'interpolation par spline cubique produit une courbe très lisse, fidèle au nuage de points simulés. Elle présente toutefois parfois de légers sur- ou sous-ajustements près des extrêmes (K/S très petit).

Régression à noyau. La méthode par régression à noyau offre une excellente robustesse et suit de près la décroissance monotone des prix de *calls*. On observe un ajustement quasi parfait au centre de la distribution, avec une très légère sous-estimation des valeurs hors de la monnaie.

Réseau RBF. Le réseau de neurones à fonctions de base radiale (RBF) combine souplesse et capacité à capturer la courbure locale. Il suit rigoureusement la forme théorique tout en évitant les oscillations excessives.

3.2.2 Volatilités implicites

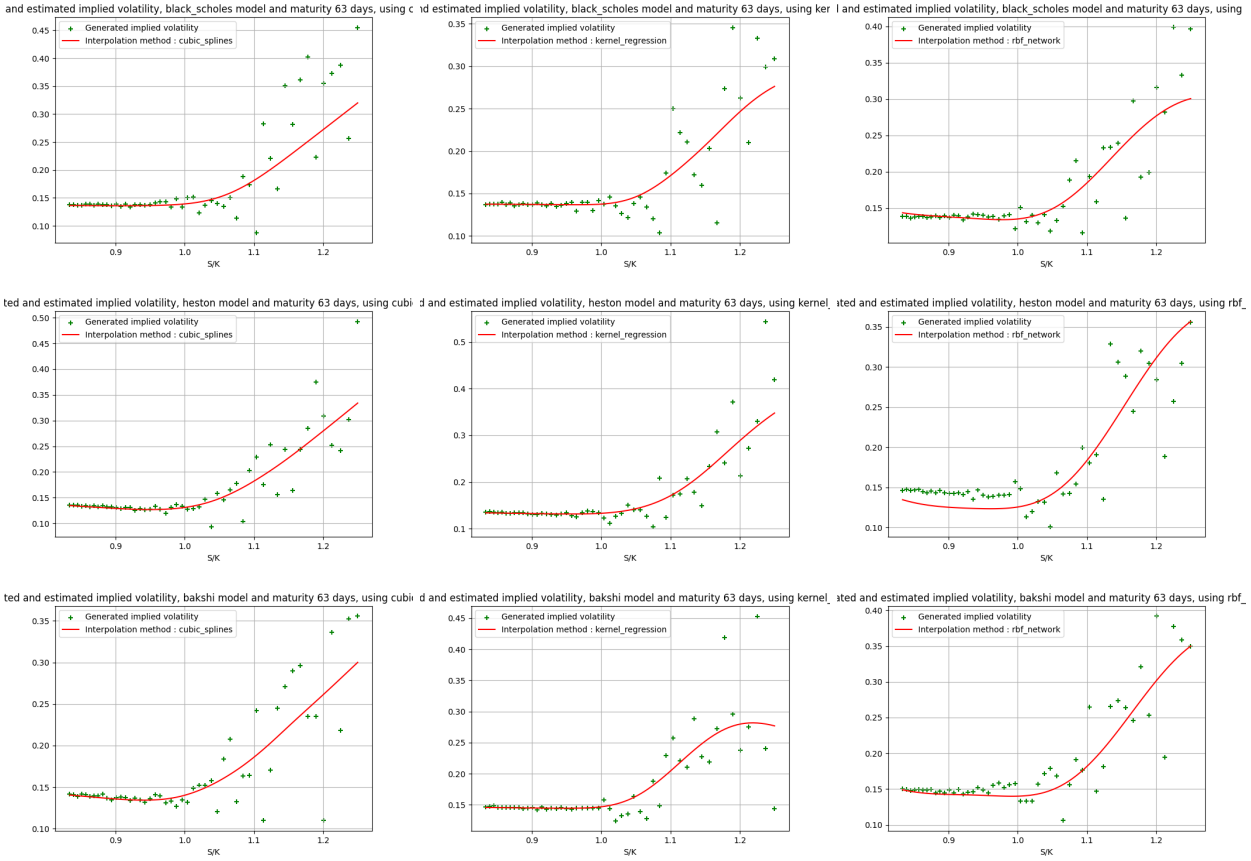


FIGURE 4 – Volatilités implicites des données générées et interpolation. De haut en bas : modèle de Black-Scholes, modèle de Heston, modèle de Bakshi. De gauche a droite : splines cubiques, regression a noyau et *rbf neural network*. Maturité : 3 mois.

Analyse des résultats

Spline cubique. L'interpolation par splines cubiques restitue une courbe globalement lisse et monotone. Elle capte bien le léger creux autour de $S/K \approx 1$, mais elle manque de réactivité aux extrémités : pour $S/K < 0{,}9$ et $S/K > 1{,}2$, la pente est sous-estimée et la convexité de la « smile » est atténuée.

Régression à noyau. La régression à noyau se révèle la plus fidèle au nuage de volatilités simulées. Elle maintient une plateforme stable autour du « at-the-money », puis monte avec agilité aux ailes, reproduisant la convexité accentuée pour les options hors-de-la-monnaie.

Réseau RBF. Le réseau RBF tend à lisser excessivement les variations extrêmes. Si la partie centrale ($0{,}9 \leq S/K \leq 1{,}1$) est bien ajustée, les aspirations de la « smile » aux ailes sont trop atténuées : la montée rapide de la volatilité pour $S/K > 1{,}1$ est sous-estimée, indiquant un sous-ajustement du modèle face aux valeurs hors-de-la-monnaie.

3.2.3 Densités de probabilité risque-neutre

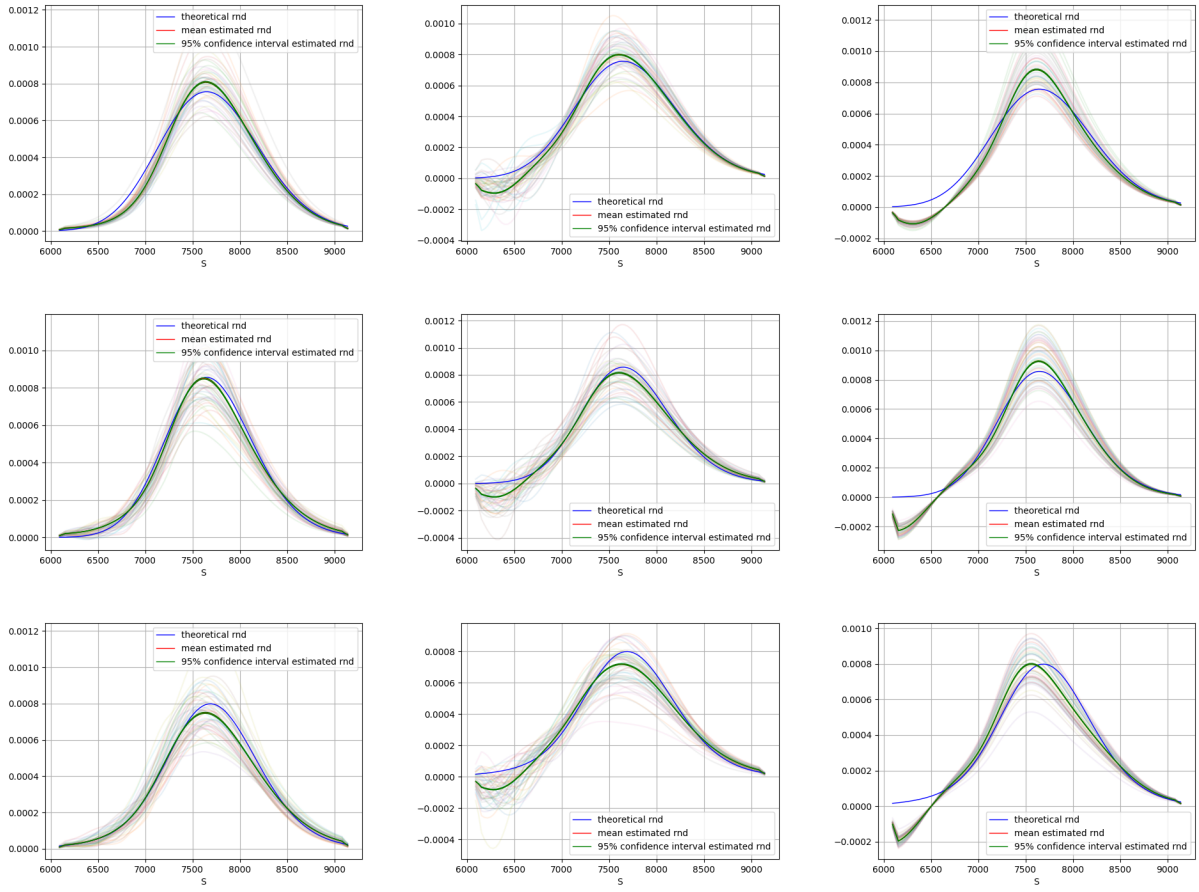


FIGURE 5 – Densités de probabilité risque-neutre estimées - moyenne, intervalle de confiance à 95% et comparaison avec la vraie densité de probabilité risque-neutre. De haut en bas : modèle de Black-Scholes, modèle de Heston, modèle de Bakshi. De gauche a droite : splines cubiques, regression a noyau et *rbf neural network*. Maturité : 3 mois.

Analyse des densités

Spline cubique. La densité risque-neutre estimée par splines cubiques (voir figures de gauche) suit globalement la forme en cloche de la RND théorique (courbe bleue). L'intervalle de confiance à 95 % (courbe verte) est étroit au voisinage du mode, mais s'élargit légèrement aux extrémités, traduisant une moindre précision sur les queues de distribution. Les oscillations résiduelles sont quasi nulles, ce qui témoigne d'une interpolation très lisse, au prix d'une possible sous-estimation de la queue gauche (légère dépression pour $S \lesssim 6500$).

Régression à noyau. La régression à noyau (figures centrales) offre l'ajustement le plus fidèle : la moyenne estimée (rouge) coïncide presque partout avec la RND théorique, et l'intervalle de confiance reste contenu autour de la vraie densité, y compris dans les régions de faible probabilité. Seule une infime sous-estimation apparaît très légèrement pour $S \lesssim 6500$, sans impact significatif sur l'allure générale.

Réseau RBF. Le réseau RBF (figures de droite) tend à lisser davantage les fluctuations, au détriment de la queue gauche qui devient même négative autour de $S \approx 6300$. Bien que le pic central soit correctement localisé, la dispersion des trajectoires individuelles (nuage transparent) est plus marquée, et l'intervalle de confiance s'écarte sensiblement de la densité théorique sur les extrémités.

3.2.4 Résultats des tests de Kolmogorov-Smirnov

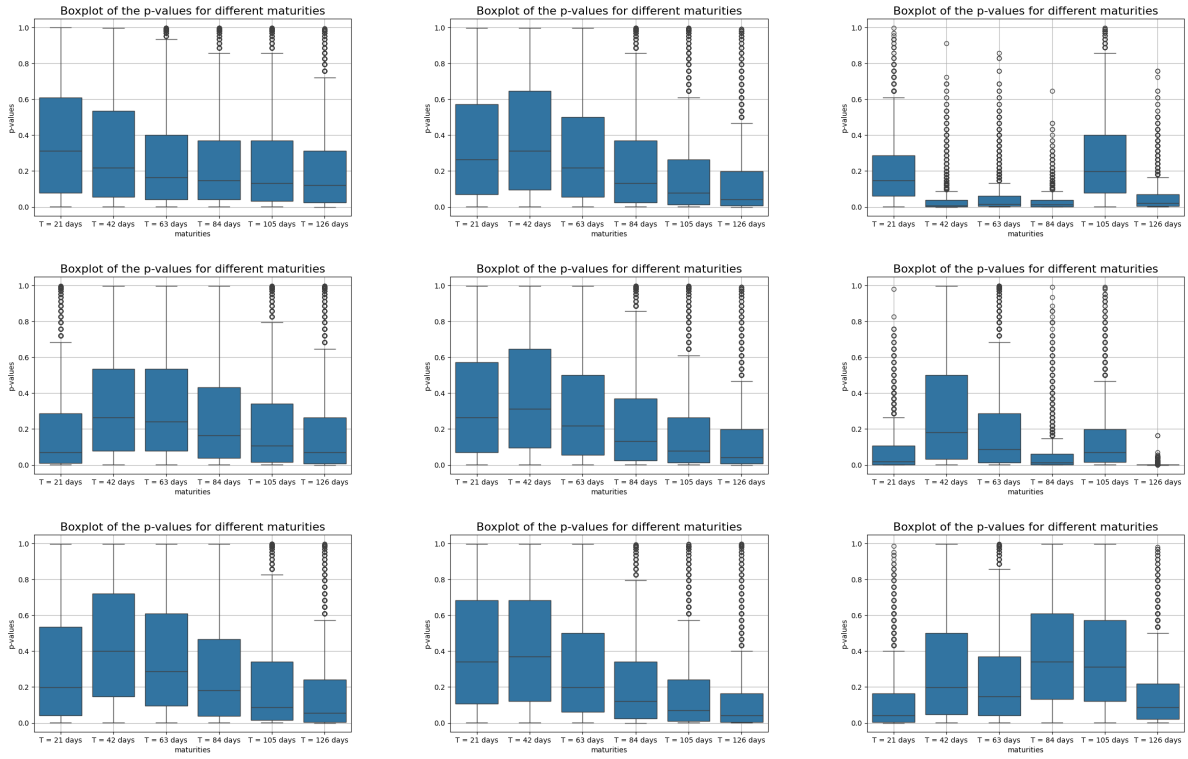


FIGURE 6 – *Boxplots* des p-valeurs provenant des tests de Kolmogorov-Smirnov pour différentes maturités. De haut en bas : modèle de Black-Scholes, modèle de Heston, modèle de Bakshi. De gauche a droite : splines cubiques, regression à noyau et *rbf neural network*. Maturité : 3 mois.

TABLE 2 – Résultats des tests de Kolmogorov-Smirnov. Les valeurs sont a lire verticalement par groupe de trois, de haut en bas : moyenne des p-valeur, écart type des p-valeurs, pourcentage de rejet de l'hypothèse nulle.

Modèle	Maturité (mois) Méthode d'interpolation	1	2	3	4	5	6
Bakshi	Splines cubiques	0.32	0.44	0.37	0.28	0.21	0.17
		0.31	0.31	0.30	0.28	0.26	0.23
		0.27	0.12	0.17	0.28	0.41	0.48
	Regression a noyau	0.41	0.41	0.30	0.22	0.17	0.13
		0.32	0.31	0.28	0.24	0.22	0.20
		0.15	0.14	0.23	0.34	0.45	0.53
	rbf network	0.12	0.30	0.24	0.39	0.36	0.16
		0.16	0.29	0.24	0.29	0.27	0.19
		0.54	0.26	0.28	0.13	0.13	0.38
Black Scholes	Splines cubiques	0.37	0.31	0.26	0.24	0.23	0.21
		0.31	0.30	0.26	0.25	0.25	0.24
		0.21	0.25	0.27	0.29	0.31	0.34
	Regression a noyau	0.34	0.38	0.31	0.24	0.18	0.15
		0.30	0.31	0.29	0.26	0.23	0.21
		0.21	0.18	0.24	0.33	0.43	0.52
	rbf network	0.20	0.04	0.05	0.03	0.27	0.05
		0.17	0.08	0.09	0.05	0.23	0.08
		0.21	0.79	0.71	0.83	0.18	0.67
Heston	Splines cubiques	0.19	0.33	0.33	0.27	0.22	0.18
		0.25	0.29	0.29	0.28	0.26	0.25
		0.45	0.20	0.20	0.29	0.38	0.46
	Regression a noyau	0.30	0.39	0.33	0.26	0.19	0.15
		0.29	0.32	0.30	0.27	0.24	0.21
		0.24	0.17	0.24	0.31	0.41	0.50
	rbf network	0.08	0.29	0.20	0.06	0.14	0.00
		0.13	0.30	0.25	0.11	0.18	0.01
		0.63	0.30	0.40	0.71	0.42	1.00

Globalement nous obtenons des résultats assez différents que [7], pour les tests de Kolmogorov-Smirnov. Dans nos tests, la moyenne des p-valeurs est généralement plus faible que dans les résultats de [7], et l'hypothèse nulle est beaucoup plus souvent rejetée (et même parfois elle est tout le temps rejetée, pour le modèle de Heston, la régression à noyau et la maturité 6 mois par exemple).

Aussi nous observons que plus la maturité est grande, moins les modèles reconstruisent bien la densité de probabilité risque-neutre. Il y a néanmoins un fait stylisé que nous retrouvons. Les *calls* avec maturité un mois donnent souvent lieu a une moins bonne qualité d'estimation de la densité de probabilité risque-neutre que les *calls* de maturité deux mois.

4 Références

Références

[1] Gurdip Bakshi, Charles Cao, and Zhiwu Chen. Empirical performance of alternative option pricing models. *The Journal of finance*, 52(5) :2003–2049, 1997.

[2] Fischer Black and Myron Scholes. The pricing of options and corporate liabilities. *Journal of political economy*, 81(3) :637–654, 1973.

[3] Robert R Bliss and Nikolaos Panigirtzoglou. Option-implied risk aversion estimates. *The journal of finance*, 59(1) :407–446, 2004.

[4] Eduardo García-Portugués. *Notes for Nonparametric Statistics*. Universidad Carlos III de Madrid, v6.9.0 edition, 2023. Available online at https://bookdown.org/egarpor/NP-UC3M/.

[5] Steven L Heston. A closed-form solution for options with stochastic volatility with applications to bond and currency options. *The review of financial studies*, 6(2) :327–343, 1993.

[6] Peter Jäckel. Let's be rational. *Wilmott*, 2015(75) :40–53, 2015.

[7] Wan-Ni Lai. Comparison of methods to estimate option implied risk-neutral densities. *Quantitative Finance*, 14(10) :1839–1855, 2014.

[8] Christian H Reinsch. Smoothing by spline functions. *Numerische mathematik*, 10(3) :177–183, 1967.

[9] Nikolai V Smirnov. On the estimation of the discrepancy between empirical curves of distribution for two independent samples. *Bull. Math. Univ. Moscou*, 2(2) :3–14, 1939.