
***Machine Learning for Finance***

Deep Parametric PDE Method : Application to Option Pricing

ENSAE IPParis

Julien Schaeffer, Noureddine Oulid Azouz, Paul Vialard

2022

# Table des matières

# 1 Introduction

Les équations aux dérivées partielles (EDP) paramétriques en haute dimension posent un défi majeur dans de nombreux domaines des sciences et de l'ingénierie. En finance, leur résolution est d'une importance capitale, notamment pour les tâches répétitives comme le contrôle des risques en temps réel, ou le calcul de CVA. Par ailleurs, la structuration de produits dérivés nécessite de pouvoir *pricer* fréquemment et efficacement à différents instants et états du marché pour différents paramètres du modèle.

Les méthodes modernes de *pricing* à l'aide de Monte-Carlo ont une efficacité plutôt limitée et les autres méthodes ont une faible tolérance aux grandes dimensions. Kathrin Glau et Linus Wunderlich proposent dans leur article *The Deep Parametric PDE Method : Application to Option Pricing*[3] de résoudre ce type d'équations à l'aide de *Deep Neural Networks (DNN)* qui offrent de bonnes approximations sans pour autant pâtir du fléau de la dimension.

Cependant, même si évaluer un réseau de neuronnes est rapide, la phase d'entraînement peut être relativement longue. Dans ce contexte, [3] propose une *deep parametric PDE method* pour entrainer le réseau sur tous les paramètres en même temps, contrairement à la méthode *deep Galerkin* qui n'entraîne que sur un ensemble de paramètres figé, le tout à partir d'une approche non-supervisée. Les auteurs reprennent l'architecture des *highway networks*, particulièrement adaptée pour l'entraînement de réseau de neurones très profonds[5] et qui ont montré leur performance dans la résolution d'EDP.

Après analyse des résultats du DNN, lors de laquelle nous proposons une évaluation basée sur la volatilité implicite, nous suggérerons une perspective d'amélioration permettant d'améliorer le calcul des sensibilités et ainsi étudier le potentiel de cette approche dans une optique de couverture.

# 2 Modèle

L'objectif du modèle est d'entraîner un réseau de neurones capable de résoudre l'EDP suivante pour n'importe quel paramètre $\mu \in \mathscr{P}$ :

$$\begin{aligned} \partial_t u(t,x;\mu) + \mathscr{L}_x^\mu u(t,x;\mu) &= f(t,x;\mu), \quad && (t,x) \in \mathscr{Q} = (0,T) \times \Omega \\ u(0,x;\mu) &= g(x;\mu), && x \in \Omega \\ u(t,x;\mu) &= u_\Sigma(t,x;\mu), && (t,x) \in \Sigma = (0,T) \times \partial\Omega \end{aligned}$$

où $\mathscr{L}_x^\mu$ désigne un opérateur différentiel du second ordre opérant sur la variable d'état x.

## 2.1 Fonction de perte

La première étape est de définir un problème de minimisation. Comme dans la méthode *deep Galerkin*, la *deep parametric PDE method* utilise l'EDP sous forme de moindres carrés pour définir des résidus. La fonction de perte à utiliser pour entraîner le réseau est alors obtenue en intégrant l'écart à l'équation différentielle sur l'ensemble $\mathscr{P}$ des paramètres du modèle.

Pour une fonction donnée $u : \mathscr{Q} \times \mathscr{P} \to \mathbb{R}$, la perte définie sur les résidus de l'EDP est :

$$\mathscr{J}(u) = \mathscr{J}_{\text{int}}(u) + \mathscr{J}_{\text{ic}}(u) + c_{\text{bc}}\mathscr{J}_{\text{bc}}(u)$$

où l'erreur intérieur est définie comme :

$$\mathscr{J}_{\text{int}}(u) = |\mathscr{Q} \times \mathscr{P}|^{-1} \int_{\mathscr{P}} \int_{\mathscr{Q}} \left(\partial_t u(t,x;\mu) + \mathscr{L}_x^\mu u(t,x;\mu) - f(t,x;\mu)\right)^2 \, \mathrm{d}(t,x)\mathrm{d}\mu$$

où $|\mathscr{Q} \times \mathscr{P}|$ est la taille du domaine, et l'erreur initiale définie comme :

$$\mathscr{J}_{\text{ic}}(u) = |\Omega \times \mathscr{P}|^{-1} \int_{\mathscr{P}} \int_{\Omega} (u(0,x;\mu) - g(x;\mu))^2 \, \mathrm{d}x \, \mathrm{d}\mu$$

L'erreur sur la frontière est également :

$$\mathscr{J}_{\text{bc}}(u) = |\Sigma \times \mathscr{P}|^{-1} \int_{\mathscr{P}} \|u(\mu) - u_\Sigma(\mu)\|^2_{H^{1/2}(\Sigma)} \, \mathrm{d}\mu$$

et est pondéré par un facteur $c_{\text{bc}} \geq 0$, qui est choisi égal à zéro dans [3] car les résultats obtenus avec cette valeur sont jugés satisfaisants. Comme cette approche est non-supervisée, aucune donnée n'est nécessaire.

Les intégrales définies précédemment sont évaluées par Monte Carlo :

$$\mathscr{J}_{\text{int}}(u) \approx \sum_{i=1}^{N} \left(\partial_t u\left(t^{(i)}, x^{(i)}; \mu^{(i)}\right) + \mathscr{L}_x^\mu u\left(t^{(i)}, x^{(i)}; \mu^{(i)}\right) - f\left(t^{(i)}, x^{(i)}; \mu^{(i)}\right)\right)^2 / N,$$

$$\mathscr{J}_{\text{ic}}(u) \approx \sum_{i=1}^{N} \left(u\left(0, \hat{x}^{(i)}; \hat{\mu}^{(i)}\right) - g\left(\hat{x}^{(i)}; \hat{\mu}^{(i)}\right)\right)^2 / N,$$

Où $\left(t^{(i)}, x^{(i)}, \mu^{(i)}\right) \in \mathscr{Q} \times \mathscr{P}$ et $\left(\hat{x}^{(i)}, \hat{\mu}^{(i)}\right) \in \Omega \times \mathscr{P}$ pour $i = 1, \ldots, N$ sont générés aléatoirement suivant une distribution uniforme. Dans [3], N a été choisi égal à 10000 par les auteurs car des approximations moins précises ont été observées pour des valeurs plus petites.

## 2.2 Pricing de Basket Options

Pour tester l'efficacité de cette méthode, les auteurs l'appliquent au *pricing* de *Basket Options* sous le modèle de Black-Scholes.

L'EDP concerne donc $u(t,x;\mu)$, le prix de l'option à l'instant $\tau = T - t$, pour des actif de prix $s_i = e^{x_i}$ :

$$u(t,x;\mu) = \text{Price}\left(T - t, e^x; \mu\right)$$
$$\text{Price}(\tau, s; \mu) = e^{-r(T-\tau)}\mathbb{E}\left(G\left(S_T(\mu)\right) \mid S_\tau(\mu) = s\right)$$

avec $d$ sous-jacents $S_\tau(\mu) = \left(S_\tau^1(\mu), \ldots, S_\tau^d(\mu)\right)$ et $G$ la fonction de payoff à maturité.

Le paramètre $\mu$ peut décrire les paramètres du modèle comme de l'option. Il contient ici :

— le taux sans risque $r$ ;

— les volatilités $\sigma_1, ..., \sigma_d$ ;

— les corrélations $\rho_{ij}, 1 < i, j < d$.

Dans le modèle de Black-Scholes, l'EDP est homogène, i.e., $f(t, x; \mu) = 0$ et l'opérateur s'écrit :

$$\mathscr{L}_x^\mu u(t, x; \mu) = ru(t, x; \mu) - \sum_{i=1}^{d} \left( r - \frac{\sigma_i^2}{2} \right) \partial_{x_i} u(t, x; \mu) - \sum_{i,j=1}^{d} \frac{\rho_{ij}\sigma_i\sigma_j}{2} \partial_{x_i x_j} u(t, x; \mu)$$

où $r = r(\mu), \sigma_i = \sigma_i(\mu)$ et $\rho_{ij} = \rho_{ij}(\mu)$ with $\rho_{ii} = 1$. Le domaine est restreint à un hypercube : $\Omega = (x_{\min}, x_{\max})^d$.

Par ailleurs, les auteurs séparent le prix de l'option en deux parties : la limite de non-arbitrage et la valeur temps. La limite de non-arbitrage capture le comportement asymptotique de l'option. Par conséquent, la valeur temps est bornée et se prête parfaitement à une approximation via un réseau de neurones. Cependant, la limite de non-arbitrage n'est en général pas "lisse", et les auteurs en considèrent une approximation $\widehat{u}_\lambda \in C^\infty$. Pour un *call* ou un *put*, le point anguleux provient de la fonction max, qui peut être très bien approximée par la fonction softplus, plus lisse. Pour une *Basket Call Option* européenne nous avons donc l'approximation de la limite de non-arbitrage :

$$\widehat{u}_\lambda(t, x; \mu) = \frac{1}{\lambda} \log \left( 1 + e^{\lambda \left( \frac{1}{d} \sum_{i=1}^{d} e^{x_i}/d - Ke^{-rt} \right)} \right), \quad \text{avec } \lambda > 0.$$

Cette approximation est d'autant plus vraie que le valeur de $\lambda$ est grande mais les auteurs la fixent à $\lambda = 10$ afin que les dérivées secondes ne deviennent pas trop importantes. Il ne reste alors plus qu'à approximer la valeur résiduelle $u(t, x; \mu) - \widehat{u}_\lambda(t, x; \mu)$ par un réseau de neurones.

## 2.3 Architecture

Dans [3], le réseau utilisé est une variante des *highway networks*[5] qui se montrent efficaces dans l'approximation des EDP. En effet, cette architecture a montré sa performance dans l'entraînement de réseaux de neurones très profonds[5] qui sont particulièrement efficaces dans la résolution d'EDP. Les *highway networks* se caractérisent alors par des portes (*gates*) de contrôles qui apprennent à réguler le flux d'informations à travers le réseau. Cette caractéristique directement inspirée de celle des*Long Short-Term Memory*[4](LSTM) permet alors à l'information de circuler le long de réseaux très denses et profonds sans atténuation. Cela facilite *in fine* l'entraînement par descente de gradient stochastique.

Après une première couche dense, plusieurs couches de portes sont appliquées pour arriver à un scalaire en *output*. En renommant les variables d'*input* $(t, x; \mu)$ en $h^0 \in \mathbb{R}^n$ avec $n = 1 + d + n_\mu$, on a la première couche dense :

$$h^1 = \psi \left( W^0 h^0 + b^0 \right) \in \mathbb{R}^m,$$

où $W^0 \in \mathbb{R}^{m \times n}, b^0 \in \mathbb{R}^m$ pour $m$ noeuds dans chaque couche. La fonction d'activation $\psi$ utilisée est la fonction tangente hyperbolique. Ensuite, pour $l = 1, \ldots, L$ où $L$ est le nombre de couches, on a :

$$h^{l+1} = \left( 1 - g^l \right) \odot h^{l+1/2} + z^l \odot h^l \in \mathbb{R}^m,$$

où les portes $g^l$ et $z^l$ "modulent" le résultat de $h^l$ et celui de la couche intermédiaire $h^{l+1/2}$. L'opérateur $\odot$ représente la multiplication terme à terme de vecteurs. Le résultat de la couche intermédiaire $h^{l+1/2}$ inclut une porte supplémentaire $r^l$ pouvant éliminer une partie de l'information transmise par la couche précédente ("*forget gate*") :

$$h^{l+1/2} = \psi\left(U^{h,l}h^0 + W^{h,l}\left(h^l \odot r^l\right) + b^{H,l}\right) \in \mathbb{R}^m.$$

Les trois portes ont la même structure :

$$\begin{aligned} g^l &= \psi\left(U^{g,l}h^0 + W^{g,l}h^l + b^{g,l}\right), \\ r^l &= \psi\left(U^{r,l}h^0 + W^{r,l}h^l + b^{r,l}\right) \\ z^l &= \psi\left(U^{z,l}h^0 + W^{z,l}h^l + b^{z,l}\right). \end{aligned}$$

Les poids et les biais ont tous la même taille, $U^{*,l} \in \mathbb{R}^{m\times n}, W^{*,l} \in \mathbb{R}^{m\times m}$ et $b^{*,l} \in \mathbb{R}^m$ et sont des paramètres entraînables du réseaux de neurones.

Une dernière couche dense et l'ajout de la localisation donnent la fonction du prix de l'option :

$$u^\theta_{\text{DNN}}(t,x;\mu) = \widehat{u}_\lambda(t,x;\mu) + W^{L+1}h^{L+1} + b^{L+1},$$

où $W^{L+1} \in \mathbb{R}^{1\times m}, b^{L+1} \in \mathbb{R}$. On cherche le paramètre $\hat{\theta}$ qui minimise la fonction de perte :

$$\hat{\theta} = \arg\min_\theta \mathscr{J}\left(u^\theta_{\text{DNN}}\right)$$

Ce qui défini la solution de l'EDP *deep parametric* :

$$u(t,x;\mu) \approx u_{\text{DPDE}}(t,x;\mu) = u^{\widehat{\theta}}_{\text{DNN}}(t,x;\mu).$$

Finalement, la fonction $u_{\text{DPDE}}$ est un réseau de neurones qui approxime la solution de l'EDP à tout instant, état du marché et pour n'importe quel paramètre du modèle. Ainsi, en entraînant le réseau une seule fois, nous sommes en mesure de résoudre en théorie toute la famille des EDP.

## 2.4 Dans la pratique

### 2.4.1 Paramètres de Black-Scholes

Dans le modèle de Black-Scholes, les paramètres sont le strike $K$, le taux sans risque $r$, les volatilités $\sigma_i$ et les corrélations $\rho_{ij}$. Cependant, en redimmensionnant les prix des actifs les auteurs transforment le problème en un problème de *pricing* d'option de strike fixe. Dans [3] le strike est donc choisi fixé à 100. Le taux sans risque $r$, les volatilités $\sigma_i$ sont deux des entrées du vecteur de paramètres.

Quand $d > 2$, paramétrer la matrice de corrélations est une tache complexe car elle doit mener à une matrice de covariance symétrique et semi-définie positive. Plutôt que d'utiliser la factorisation de Cholesky, ce qui mènerait à $d(d-1)/2$ paramètres pour définir les corrélations, les auteurs utilisent l'approche [2], ce qui réduit le nombre de paramètres à $(d-1)$. Le vecteur de paramètres est ainsi donné par :

$$\mu = (r, \sigma_1, \ldots, \sigma_d, \hat{\rho}_1, \ldots, \hat{\rho}_{d-1})$$

### 2.4.2 Paramètres internes du réseau

La minimisation de la fonction de perte dans le réseau de neurones est un problème d'optimisation non-linéaire et non-convexe. Dans [3] il est résolu par une variante de la méthode de descente de gradient avec un critère d'arrêt : si la perte ne s'est pas améliorée pendant les 50 dernières *epochs* (de 10 *batches* chacune) l'optimisation prend fin et les poids de la perte minimale observée sont utilisés. L'algorithme Adam est utilisé pour optimiser la descente de gradient

Pour que la descente de gradient soit plus efficace, les valeurs initiales des poids sont choisies grandes et les *inputs* sont normalisés. Le temps, les valeurs des actifs et les paramètres subissent donc une transformation linéaire qui les renvoient dans le domaine $[-1, 1]$.

Après une optimisation des hyper-paramètres par recherche randomisée, le réseau [3] contient 9 couches de 90 neurones chacune. Le *learning rate* de départ est fixé à 0,001.

# 3 Résultats

## 3.1 Par écarts de prix

Après avoir entraîné le réseau, il convient de comparer ses *outputs* à un *pricer* de référence. Dans le cas univarié (1 actif dans le basket), il est possible d'utiliser la formule de Black-Scholes car cela revient à *pricer* un Call ou un Put :

$$c(t, x; \mu) = \mathrm{BS}\left(t, e^x, r, \sigma, K\right) = \Phi\left(d_1\right) e^x - \Phi\left(d_2\right) K e^{-rt} \text{ par exemple pour le Call,}$$
$$\text{avec } d_1 = \frac{1}{\sigma\sqrt{t}}\left(x - \ln(K) + rt + \frac{\sigma^2 t}{2}\right) \text{ et } d_2 = d_1 - \sigma\sqrt{t}.$$

Dans le cas unidimensionnel, le réseau semble très bien fonctionner avec une erreur absolue inférieure à 0.05.

Ensuite, il reste à évaluer la performance dans le cas de la *basket option* avec n actifs. Pour obtenir le *pricer* de référence, la méthode du lissage du payoff présentée dans [1] est employée. Cette méthode permettant d'obtenir un prix de référence présente le défaut d'être très coûteuse en temps de calcul car elle nécessite le calcul d'intégrales en grande dimension. C'est pourquoi elle ne pourrait remplacer la méthode développée dans l'article, qui elle est très rapide pour calculer un prix. Les résultats sont présentés dans le tableau suivant :

| | Reference pricer | Deep Galerkin | Deep parametric PDE |
|---|---|---|---|
| value | 3.9217 | 3.9335 | 3.9327 |
| rel. error | 0.130% | 0.431% | 0.411% |
| offline runtime | – | 28 min pour chaque $\hat{\mu}$ | 59 min une fois $\forall \mu \in \mathscr{P}$ |
| online runtime | 1.34 s | 39.9 ms | 41.9 ms |

Nous pouvons observer que le *pricer* de référence nécessite plus d'une seconde pour calculer un prix là où les méthodes reposant sur des réseaux mettent quelques dizaines de millisecondes : c'est le temps de calculer une sortie pour un réseau de neurones. De plus, ces dernières semblent très efficaces puisque l'erreur relative de ces méthodes est de l'ordre de 0.4% par rapport à la référence. *In fine*, nous observons que la méthode Deep

Galerkin nécessite un temps de calcul hors-ligne de 28 minutes pour un set de paramètre fixé, là où la méthode Deep Parametric PDE nécessite 59 minutes, mais ici pour tous les paramètres possibles sur notre domaine. Cette perte de temps initiale par rapport à la méthode Deep Galerkin se révèle donc très avantageuse par la suite, car elle ne nécessitera qu'une seule phase calculatoire. La méthode *Deep Parametric PDE* n'utilisant que 0.2 ms de plus que la méthode Deep Galerkin pour calculer un prix, nous pouvons en conclure que la méthode Deep Parametric PDE constitue une vraie amélioration de la méthode Deep Galerkin existante.

## 3.2 Avec la volatilité implicite

La comparaison avec le *pricer* de référence peut se faire avec les prix, mais aussi avec la formule généralisée de la volatilité implicite :

$$\mathrm{BS}\left(t, \sum_{i=1}^{d} e^{x_i}/d, r, \sigma_{\mathrm{iv}}, K\right) = c$$

Cette méthode d'évaluation peut se révéler fondamentale pour le praticien, car c'est en général la volatilité implicite qu'il utilise pour comparer différents prix de marché.

Cependant, pour cette méthode d'évaluation l'article mentionne des difficultés potentielles, notamment loin dans la monnaie. Pour bien comprendre cela, nous proposons de visualiser la surface de la volatilité implicite pour différentes valeurs des sous-jacents $s_1$ et $s_2$. Afin de calculer cette volatilité implicite, nous utilisons la méthode de Newton-Raphson et reprenons le *pricer* de référence de [1] ainsi que le réseau des auteurs. En considérant la formule de Black-Scholes comme une fonction dépendant uniquement de $\sigma$, et en partant d'une volatilité $\sigma_0$, nous utilisons la formule de récurrence suivante :

$$\sigma_{n+1} = \sigma_n - \frac{BS_{Call}(\sigma_n) - Price}{Vega_{Call}(\sigma_n)}$$

où *Price* correspond au prix donné par le *pricer* de référence ou le réseau de neurones profond (DNN), jusqu'à obtenir une valeur satisfaisante de $\sigma$. Dans les cas où l'algorithme ne convergerait pas, nous décidons d'utiliser une approche naïve, à savoir partir d'une volatilité $\sigma_0$ élevée et de la diminuer jusqu'à obtenir une valeur satisfaisante.

Nous présentons les résultats obtenus ci-dessous :

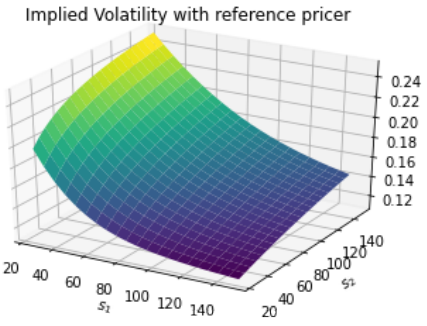


FIGURE 1 – Volatilité implicite avec le pricer de référence : la surface semble correcte

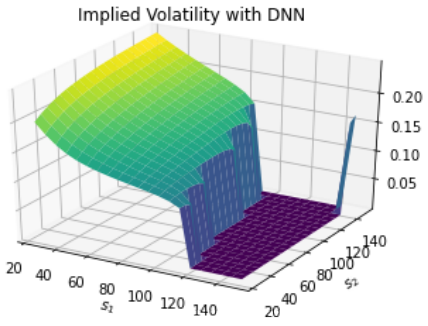


FIGURE 2 – Volatilité implicite avec le pricer DPDE : la surface s'effondre pour $s_1 + s_2 > 140$

Il apparaît alors nettement que la volatilité implicite ne peut être utilisée sur tout le domaine défini par $(s_1, s_2)$ pour comparer les deux *pricers*. En effet, bien que la volatilité implicite semble correcte pour le *pricer* de référence (figure 1), les valeurs de cette dernière s'effondrent subitement vers 0 pour le DNN de [3] (figure 2) dès lors que $s_1 + s_2$ semble dépasser un seuil d'environ 140. Compte tenu que la valeur du strike $K$ choisie ici est de 100, nous sommes effectivement loin dans la monnaie.

Pour comprendre le problème, nous avons tracé la formule de Black-Scholes en fonction de $\sigma$ avec $s_1$, $s_2$ fixés tels que $s_1 + s_2 > 140$.

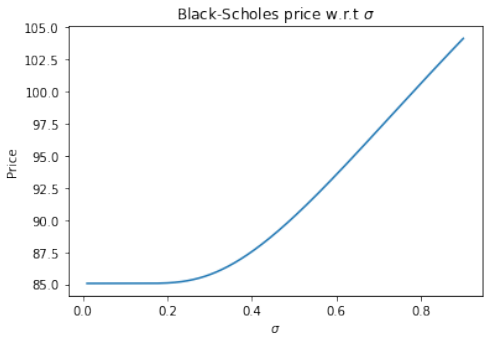


FIGURE 3 – Black-Scholes en fonction de $\sigma$ avec $S = 130$, $K = 100$, $T = 4$ et $r = 0.2$

Nous observons que pour des faibles valeur de volatilité (typiquement inférieures à 0.2), un faible changement de prix peut avoir une grande conséquence sur la volatilité implicite retenue. C'est à ce problème que nous semblons confrontés. En effet, la valeur minimale obtenue sur ce graphique pour le prix est de 85.0671. Avec les mêmes paramètres, le *pricer* de référence donne un prix de 85.0688 et le DNN 85.0561. Bien que l'écart entre le DNN et le *pricer* de référence soit de l'ordre de $10^{-2}$, nous observons bien que le *pricer* de référence peut "trouver" une volatilité implicite correspondante via la formule de Black-Scholes là où ce n'est pas possible pour le DNN. Ceci explique la surface de volatilité implicite qui s'effondre pour certaines valeurs pour le DNN, et justifie que la volatilité implicite ne peut être utilisée qu'avec parcimonie pour évaluer le DNN, constituant alors une limite pour le praticien.

# 4 Un axe d'amélioration : les grecques

Des difficultés à évaluer les grecques sont rapidement mentionnées dans l'article. En effet, les auteurs expliquent que malgré l'efficacité du DNN pour calculer le prix de la basket option, les calculs des grecques ne sont pas aussi fiables et parfois éloignés de leurs vraies valeurs. Pourtant, les différentes sensibilités sont très utiles pour le praticien, notamment s'il veut pouvoir se couvrir convenablement. Dans cette section, nous allons plus particulièrement nous intéresser au $\delta$, à savoir la dérivée du prix de la *basket option* par rapport à un de ses sous-jacents. Sans perte de généralité, nous n'étudions ici que le $\delta$ par rapport au premier sous-jacent :

$$\delta = \frac{\partial P}{\partial S_1}$$

Nous nous proposons donc d'afficher dans un premier temps le $\delta$ de la *basket option* obtenu d'une part avec le *pricer* de référence, et d'autre part de comparer le $\delta$ obtenu avec le DNN. Pour cela, nous revenons à la définition de la dérivée, à savoir le taux d'accroissement. Ci-dessous, nous affichons le $\delta$ obtenu avec cette méthode :

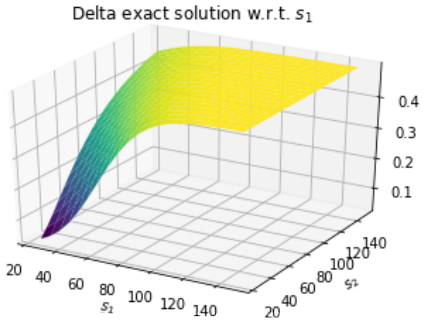


FIGURE 4 – $\delta$ calculé à partir du pricer de référence, $s_2$ fixé

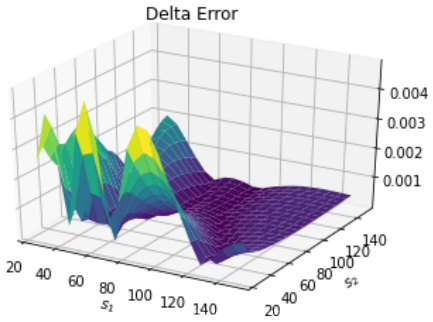


FIGURE 5 – Erreur absolue entre le $\delta$ précédent et celui du pricer DPDE

Ces graphiques laissent penser dans un premier temps que l'approximation du $\delta$ est très correcte. En effet, la *mean squared error* est de l'ordre de 0.001. Cependant, ces surfaces sont réalisées avec un petit échantillon de points (21x21). Pour vérifier le propos de l'article, à savoir que le calcul du $\delta$ n'est pas précis avec le DNN, nous convenons d'étudier avec un grand nombre de points (10000) une trajectoire du $\delta$ avec $s_2$ fixé. Les résultats ci-dessous confirment alors que le DNN éprouve des difficultés pour le calcul du $\delta$ : nous remarquons une certaine irrégularité par rapport au $\delta$ calculé avec le *pricer* de référence.

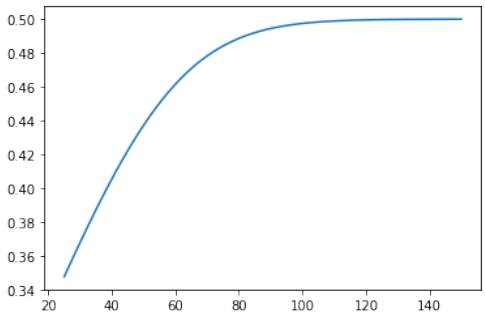


FIGURE 6 – $\delta$ calculé à partir du pricer de référence

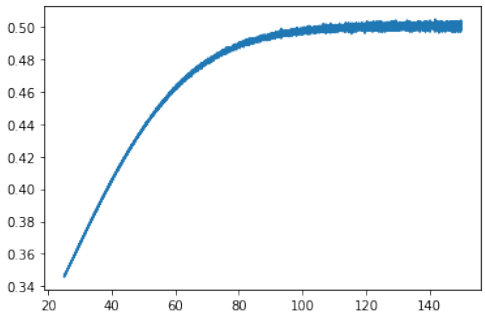


FIGURE 7 – $\delta$ calculé à partir du pricer DPDE

En effet, la *mean squared error* vaut ici 0.0595, ce qui est davantage en accord avec l'article.

Pour tenter de corriger ce défaut, nous avons comme idée de modifier la fonction de perte du réseau de neurones.

## 4.1 Fonction de perte et sensibilité

Nous proposons dans cette partie d'introduire des modifications de la fonction de perte initiale afin de tenir compte des sensibilités. Dans un premiers temps, nous avons envisagé

deux méthodes.

La première consiste à implémenter un delta de "référence" à partir du *pricer* de référence, et augmenter la fonction de perte de l'écart entre la dérivée de l'option à ce delta de référence. La fonction de perte (à l'intérieur du domaine) s'écrit alors :

$$\mathscr{J}(u) = \mathscr{J}_{\text{int}}(u) + \mathscr{J}_{\text{delta, int}}(u)$$

où nous conservons la perte $\mathscr{J}_{\text{int}}(u)$ des auteurs, augmentée de la perte $\mathscr{J}_{\text{delta, int}}(u)$ mentionnée précédemment et définie comme :

$$\mathscr{J}_{\text{int}}(u) \approx \sum_{i=1}^{N} \left(\partial_t u\left(t^{(i)}, x^{(i)}; \mu^{(i)}\right) + \mathscr{L}_x^{\mu} u\left(t^{(i)}, x^{(i)}; \mu^{(i)}\right) - f\left(t^{(i)}, x^{(i)}; \mu^{(i)}\right)\right)^2 / N,$$

$$\mathscr{J}_{\text{delta, int}}(u) \approx \sum_{i=1}^{N} \left(\partial_{x_1} u\left(t^{(i)}, x^{(i)}; \mu^{(i)}\right) - \delta\left(t^{(i)}, x^{(i)}; \mu^{(i)}\right)\right)^2 / N,$$

Malheureusement, notre implémentation a souffert du temps de calcul relativement important du *pricer* de référence. En effet, cette approche nécessite de calculer pour les 10000 combinaisons de paramètres le taux d'accroissement (donc 2 *pricing* à chaque fois) durant la phase d'entraînement de chaque *epoch*. Cette modification viendrait à la fois allonger grandement la durée d'entraînement du modèle et le priver de son caractères non-supervisée ce qui nuancerait sérieusement les performances de cette approche.

Nous proposons alors, dans la même idée que celle des auteurs d'approcher l'EPD du prix, d'ajouter à cette dernière l'EDP de la sensibilité par rapport au premier sous-jacent. Malheureusement, nous ne connaissons pas de formule close de cette EDP dans le cas multi-dimensionnel mais nous proposons de faire apparaître la dérivée temporelle du delta dans la perte originelle. Le principal avantage de cette approche et que les réseaux de neurones permettent d'accéder facilement à l'approximation de cette dérivée. La nouvelle perte s'écrit alors :

$$\mathscr{J}(u) = \mathscr{J}_{\text{int}}(u) + \mathscr{J}_{\text{delta, int}}(u)$$

où

$$\mathscr{J}_{\text{delta, int}}(u) \approx \sum_{i=1}^{N} \left(\frac{\partial[\partial_t u\left(t^{(i)}, x^{(i)}; \mu^{(i)}\right) + \mathscr{L}_x^{\mu} u\left(t^{(i)}, x^{(i)}; \mu^{(i)}\right) - f\left(t^{(i)}, x^{(i)}; \mu^{(i)}\right)]}{\partial_t}\right)^2 / N$$

L'ajout de cette nouvelle perte fait alors apparaître les termes $\partial_t \frac{\partial u}{\partial_{x_1}}$ et $\partial_t \frac{\partial u}{\partial_{x_2}}$ via la dérivée temporelle de l'opérateur $\mathscr{L}_x^{\mu}$. Nous pouvons noter que par ailleurs, si nous parvenons à résoudre parfaitement l'EDP originale, nous n'obtenons pas de nouvelles informations par cette EDP sur les sensibilités car elle est automatiquement satisfaite. Toutefois, comme nous faisons ici approximation numérique, ce sera effectivement une information supplémentaire pour l'entraînement. Les figures ci-dessous permettent de résumer les résultats de notre implémentation.

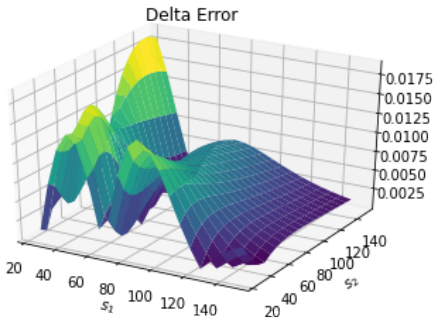


FIGURE 8 – Erreur absolue entre le $\delta$ de référence et celui du DPDE augmenté

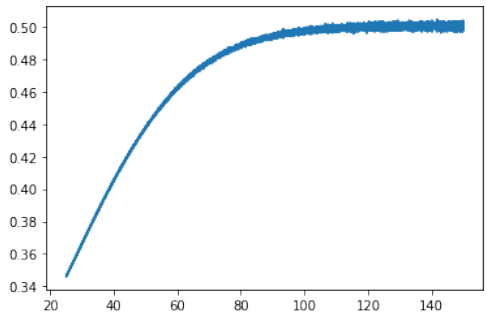


FIGURE 9 – $\delta$ calculé à partir du pricer DPDE augmenté

Malheureusement, nous ne parvenons pas à lisser le delta et observons même que sa mesure est dégradée. A priori cela provient d'une erreur lors de l'entraînement durant lequel la *loss* n'est parfois plus définie [1]. En effet, nous constatons que notre modèle s'entraîne sur une cinquantaine d'*epochs* seulement (voir annexe A.1 figure 10), durant lesquelles la *loss* semble diminuer correctement, avant de diverger soudainement. Cela pourrait s'expliquer par l'apparition de gradients trop importants après la secondes dérivation temporelle, une mauvaise gestion de *learning rate* ou encore des problèmes de normalisation. Nous écartons a priori cette dernière piste car les *inputs* semblent correctement transformés. Malgré nos efforts pour régulariser cette fonction de perte (ajustement de *learning rate*, *clipping* de gradient pour limiter son explosion), nous ne sommes malheureusement pas parvenus à entraîner le modèle sur plus d'*epochs* et nous pensons qu'il serait nécessaire de se pencher plus en profondeur sur l'architecture du modèle. Nous pourrions par exemple mener une recherche randomisée des hyperparamètres optimaux (nombre de noeuds et de couches) pour que la fonction de perte soit correctement définie. Cette étape nécessiterait toutefois une puissance de calcul très importante dont nous ne disposons pas actuellement, ce pourquoi nous ne pouvons pas la mener.

# 5 Conclusion

Nous avons présenté la méthode DPDE pour la résolution d'équations aux dérivées partielles paramétriques. L'utilisation de réseaux de neurones permet de résoudre précisément et rapidement des problèmes en grande dimension : il est possible d'évaluer la solution à différents instants, pour différents états et différents paramètres après une seule phase d'entraînement.

Appliqué au *pricing* d'options en finance, cette méthode s'avère beaucoup plus rapide que les méthodes traditionnelles, pour une très faible perte de précision. En comparaison avec la méthode *deep Galerkin*, cette méthode est équivalente mais possède l'avantage de pouvoir entraîner sur l'ensemble des paramètres du modèle en une seule fois.

La méthode DPDE fait néanmoins preuves de certaines lacunes quant au calcul des sensibilités. Nous avons essayé d'y apporter différentes solutions mais ce problème mériterait un travail plus en profondeur.

1. apparition de *NaN values*

# Références


[1] Christian Bayer, Markus Siebenmorgen et Raul Tempone. « Smoothing the payoff for efficient computation of Basket option prices ». In : *Quantitative Finance* 18.3 (2018), p. 491-505. doi : 10.1080/14697688.2017.1308003. eprint : https://doi.org/10.1080/14697688.2017.1308003. url : https://doi.org/10.1080/14697688.2017.1308003.

[2] Paul Doust. « Two Useful Techniques for Financial Modelling Problems ». In : *Applied Mathematical Finance* 17.3 (2010), p. 201-210. doi : 10.1080/13504860903257666. eprint : https://doi.org/10.1080/13504860903257666. url : https://doi.org/10.1080/13504860903257666.

[3] Kathrin Glau et Linus Wunderlich. *The Deep Parametric PDE Method : Application to Option Pricing*. 2020. doi : 10.48550/ARXIV.2012.06211. url : https://arxiv.org/abs/2012.06211.

[4] Sepp Hochreiter et Jürgen Schmidhuber. « Long Short-term Memory ». In : *Neural computation* 9 (déc. 1997), p. 1735-80. doi : 10.1162/neco.1997.9.8.1735.

[5] Rupesh Kumar Srivastava, Klaus Greff et Jürgen Schmidhuber. *Highway Networks*. 2015. doi : 10.48550/ARXIV.1505.00387. url : https://arxiv.org/abs/1505.00387.

# Annexe

# A Annexe A

## A.1 Fonction de perte

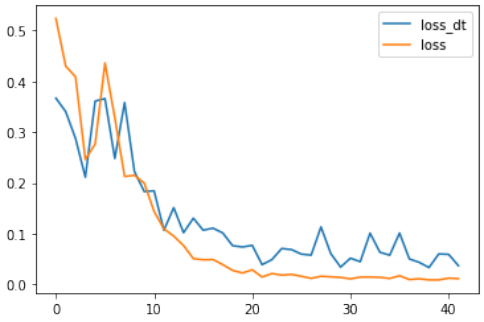


FIGURE 10 – Fonction de perte originale vs fonction de perte augmentée (moins régulière)