Méthode des différences finies explicites pour le pricing d'options via l'équation aux dérivées partielles de Black-Scholes-Merton

ALVAREZ Samuel
DUONG Simon
LEPROULT Lucas

Encadré par Roxanna Dumitrescu

Contents

1 Formulation et présentation du programme

Ce projet s'intéresse à l'estimation du prix d'un contrat dérivé en résolvant une équation aux dérivées partielles (EDP) via l'utilisation du schéma de différences finies explicites, dans le cadre du modèle de Black-Scholes-Merton. Avant de passer à la partie algorithmique, nous introduisons brièvement les éléments mathématiques essentiels à la compréhension de ce projet.

Rappelons l'EDP de Black-Scholes d'une option ($(\Pi_t)_{t\geq 0}$ étant le processus de payoff de celle-ci), sur un actif sous jacent dont le processus de prix est donné par $(S_t)_{t\geq 0}$, est:

$$r\Pi_t = rS_t\frac{\partial \Pi_t}{\partial S_t} + \frac{1}{2}\sigma^2 S_t^2 \frac{\partial^2 \Pi_t}{\partial S_t^2} - \frac{\partial \Pi_t}{\partial t} \tag{EDP}$$

L'objectif est donc, à partir du schéma de différences finies explicites, de fournir des solutions numériques à l'équation (EDP). En effet, le schéma par différences finies va nous permettre de transformer le problème en temps continu, en un ensemble fini d'équations à résoudre. Nous allons raisonner par récurrence rétrograde en partant de la condition "terminale" du payoff d'un call vanille (i.e son payoff à maturité), jusqu'à la date $t = 0$. Nous allons donc discrétiser l'intervalle $[0, T]$ et $[S_{\min}, S_{\max}]$ respectivement en N et I sous intervalles discrets. Sans grande perte de généralités, on suppose que $S_{\min} = 0$, nous allons donc discrétiser les intervalles $[0, T]$ et $[0, S_{\max}]$.
En supposant que $(x, t) \mapsto \Pi(x, t)$ est de classe $C^{2,1}$, nous avons, en utilisant des développements de Taylor à l'ordre 1 et 2 que (pour Δ_t et Δ_x assez petits):

$$\frac{\partial \Pi(x,t)}{\partial t} \approx \frac{\Pi(x, t+\Delta_t) - \Pi(x,t)}{\Delta_t}$$

$$\frac{\partial^2 \Pi(x,t)}{\partial x^2} \approx \frac{\Pi(x+\Delta_x, t) - 2\Pi(x,t) + \Pi(x-\Delta_x, t)}{(\Delta_x)^2}$$

et,

$$\frac{\partial \Pi(x,t)}{\partial x} \approx \frac{\Pi(x+\Delta_x, t) - \Pi(x-\Delta_x, t)}{2\Delta_x}$$

Ainsi, pour tout $n \in \{1, 2, \ldots, N\}$ et $i \in \{1, 2, \ldots, I\}$ (EDP) peut se réecrire de manière discrete:

$$r\Pi_i^n \approx rS_i\frac{\Pi_{i+1}^n - \Pi_{i-1}^n}{2\Delta_x} + \frac{1}{2}\sigma^2 S_i^2 \frac{\Pi_{i+1}^n - 2\Pi_i^n + \Pi_{i-1}^n}{(\Delta_x)^2} - \frac{\Pi_i^{n+1} - \Pi_i^n}{\Delta_t} \tag{1}$$

Ce qui nous donne, en réarrangeant les termes, une relation entre les prix aux dates n et $n+1$, ainsi qu'aux prix $i-1$, i et $i+1$:

$$\Pi_i^{n+1} = a_i\Pi_{i-1}^n + b_i\Pi_i^n + c_i\Pi_{i+1}^n \tag{EDP Dynamique}$$

Avec, pour tout $i = 1, \ldots, I$, $S_i = S_{\min} + i\Delta_x = i\Delta_x$, et les coefficients suivants:

$$a_i = \frac{\sigma^2 i^2 \Delta_t}{2} - \frac{ri\Delta_t}{2} \tag{2}$$

$$b_i = 1 - \Delta_t(\sigma^2 i^2 + 2) \tag{3}$$

$$c_i = \frac{\sigma^2 i^2 \Delta_t}{2} + \frac{ri\Delta_t}{2} \tag{4}$$

Les sensibiltés seront alors données par:

$$\begin{aligned} \Delta_i &= \frac{\Pi_{i+1} - \Pi_{i-1}}{2\Delta x}, \\ \Gamma_i &= \frac{\Pi_{i+1} - 2\Pi_i + \Pi_{i-1}}{\Delta x^2}, \\ \Theta_i &= \frac{\Pi_i^{n+1} - \Pi_i^n}{\Delta t}. \end{aligned} \tag{G}$$

Le contexte mathématique étant posé, il s'agit désormais de présenter le programme dans sa globalité, ainsi que ses fonctionnalités. Dans cette partie, nous nous concentrons sur le fonctionnement du programme en sa globalité, nous rentrerons dans les détails de son architecture dans la prochaine section.
L'implémentation du Schéma par Différences Finies Explicites se base sur la l'implémentation de l'équation (EDP Dynamic), que l'on va appliquer récursivement le long d'une grille spatiale $[0, S_{\max}]$ et temporelle $[0, T]$.

Dans notre code, les classes précédant *MDF*, ont pour but d'introduire les méthodes liées aux options que nous mettons en jeu (payoff et paramètres intrinsèques: strike, volatilité etc..). Nous allons donc nous concentrer davantage sur la classe *MDF* et sur le choix des variables pour une implémentation cohérente.

Cette classe contient cinq fonctions dédiées aux calculs et à l'impression de la grille de payoffs, ainsi qu'à l'estimation du prix à la date 0 de l'option. Les fonctions *long_ pas, init_ cond, boundaries, dom* représentent les étapes "internes" de ce schéma. La fonction *prog_ pas* est à l'interface avec l'utilisateur.
Commençons par *long_ pas*, les entrées nécessaires sont, la maturité (T), le nombre de points sur la grille temporelle (N), ainsi que le nombre de points sur la grille spatiale (I). Nous avons donc :

$$\Delta_t = \frac{T}{N-1}, \tag{5}$$

$$\Delta_x = \frac{S_{\max}}{I-1}, \tag{6}$$

$$S_{\text{val}} = [S_1, \ldots, S_i], \quad \forall i = 1, \ldots, I, \quad S_i = i\Delta_x. \tag{7}$$

Le vecteur S_{val} correspond aux valeurs discrètes de prix du sous-jacent. Celles-ci nous permettront d'estimer le prix de l'option à la date 0, en fonction d'un spot price S_0 donné.

Notons que nous devons également imposer une condition de *stabilité de l'algorithme*, donnée par :

$\Delta_t \leq \frac{\Delta_x^2}{2\sigma^2}$. Cependant, celle-ci est assez restrictive. En pratique, nous avons constaté que, dans certains cas où la condition n'était pas respectée, les résultats restaient tout de même cohérents, à condition que la quantité

$\Delta_t - \frac{\Delta_x^2}{2\sigma^2}$ soit assez petite (plus petite qu'un certain ε fixé).

Dans l'ordre logique, passons à *init_ cond*. Cette fonction établit les conditions initiales indispensables pour les calculs ultérieurs, en définissant les valeurs de départ nécessaires à la propagation temporelle selon le schéma des différences finies. Une fois la grille spatiale d'extrémités $S_{\min} = 0$ et $S_{\max}$ initialisée par *long_ pas*, il s'agit désormais d'appliquer les conditions initiales (valeurs à maturité de l'option sur le sous-jacent S).

$$\Pi(S_i, T) = \text{Payoff}(S_{\text{val}}[i]), \quad \forall i = 1, \ldots, I.$$

Un vecteur, noté Old, est initialisé pour stocker les valeurs de l'option au temps final $t = T$. Ce vecteur est rempli en appliquant le Payoff à chaque point de la grille spatiale S_i :

$$\text{Old}[i] = \text{Payoff}(S_{\text{val}}[i]), \quad \forall i = 0, 1, \ldots, I.$$

La fonction *boundaries* impose les conditions limites sur les deux extrémités de la grille spatiale, à savoir $S_{\min} = 0$ et $S_{\max}$. Ces conditions sont essentielles pour garantir que la solution du schéma des différences finies reste cohérente et respecte les hypothèses du modèle.

Au bord gauche ($S = S_{\min} = 0$), la condition est définie par :

$$\Pi(S_{\min}, t) = \text{borne\_inf}(t, S_{\min}),$$

Au bord droit ($S = S_{\max}$), la condition limite est donnée par :

$$\Pi(S_{\max}, t) = \text{borne\_sup}(t, S_{\max}),$$

Où, les fonctions *borne_ sup* et *borne_ inf*, définies dans la classe *EDP*, calculent les conditions aux bords associées à chaque option. Par exemple :

$$C(S_{\max}, t) = S_{\max} - Ke^{-r(T-t)}, \quad P(S_{\max}, t) = 0.$$

Respectivement pour le Call et le Put (Par relation de parité).

La fonction *dom* met à jour les valeurs de l'option entre deux instants temporels consécutifs t^n et t^{n+1} en appliquant le schéma des différences finies explicite. Cette mise à jour s'effectue point par point sur la grille spatiale, à l'exception des bords ($S_{\min}$ et $S_{\max}$), déjà fixés par la fonction *boundaries*.

Pour chaque point i ($i = 1, \ldots, I$), la fonction calcule les coefficients dynamiques a_i, b_i, et c_i, déterminés à partir des paramètres σ, r, S_i. Ces coefficients sont utilisés pour évaluer la nouvelle valeur de l'option :

$$\text{New}[i] = a_i\text{Old}[i-1] + b_i\text{Old}[i] + c_i\text{Old}[i+1].$$

Les sensibilités Δ (Delta), Γ (Gamma), et Θ (Theta) sont également calculées (d'après les équations (G)) pour chaque point spatial i, en utilisant les valeurs actuelles (Old) et mises à jour (New).

La boucle principale parcourt uniquement les points internes ($i = 1, \ldots, I-1$), les nouvelles valeurs (New) remplacent les anciennes (Old) à la fin de chaque étape temporelle, et les sensibilités (Δ, Γ, Θ) sont enregistrées dans leur vecteur respectif.

La fonction *prog_ pas* pilote l'évolution temporelle dans le schéma des différences finies explicite, de la maturité ($t = T$)

jusqu'au temps initial ($t = 0$), tout en enregistrant les résultats dans un fichier CSV. À chaque étape temporelle, les conditions limites ($S_{\min}$, $S_{\max}$) sont appliquées via *boundaries*, et les valeurs internes ainsi que les sensibilités (Δ, Γ, Θ) sont mises à jour par *dom*.

Le fichier CSV généré contient un tableau structuré avec une ligne par point spatial et les colonnes suivantes :

$$S,\ t,\ \Pi,\ \Delta,\ \Gamma,\ \Theta.$$

Chaque ligne associe une valeur du prix du sous-jacent S, un instant temporel t, le prix de l'option (Π) et ses sensibilités (Δ, Γ, Θ).

Une fois $t = 0$ atteint, le programme interpole la valeur de l'option pour un spot donné S_0 selon :

$$\Pi(S_0, 0) \approx \text{New}[i] + \frac{S_0 - S_{\text{val}}[i]}{S_{\text{val}}[i+1] - S_{\text{val}}[i]}(\text{New}[i+1] - \text{New}[i]).$$

La mise à jour temporelle est réalisée à chaque itération par :

$$\text{pass\_t} \leftarrow \text{pres\_t}, \quad \text{pres\_t} \leftarrow \text{pres\_t} - \Delta t.$$

En cas de S_0 hors de la grille ($S_0 < S_{\min}$ ou $S_0 > S_{\max}$), une erreur est signalée.
Prog_pas coordonne donc la propagation temporelle, l'enregistrement des résultats dans le fichier CSV et l'estimation du prix initial avec cohérence.

2 Architecture générale du programme

Afin de résoudre numériquement l'équation de Black–Scholes par différences finies explicites, nous avons structuré notre programme en plusieurs blocs de classes interconnectés :

1. **Classes de Payoff**

   Les différentes formes de payoff (Call, Put, Asset-or-Nothing, etc.) sont implémentées sous forme de *classes dérivées* héritant d'une classe abstraite *Payoff*. Cette dernière définit la méthode virtuelle pure *operator()(S)*, permettant de calculer le payoff pour tout prix du sous-jacent S. Cela nous offre un polymorphisme : le code appelant n'a pas besoin de connaître le *type exact* d'option pour obtenir son payoff.

2. **Classe *Option***

   Les paramètres caractéristiques de chaque option (le payoff, le strike K, la maturité T, la volatilité σ, la barrière éventuelle, etc.) sont encapsulés dans la classe *Option*. Grâce à un pointeur *Payoff**, on associe dynamiquement le payoff adéquat à l'option. Cela sert de référentiel central pour la suite (taux d'intérêt, barrière, etc.), accessible via des getters.

3. **Classes *EDP* et *BSEDP***

   - *EDP* est une interface (classe abstraite) spécifiant les coefficients d'une équation de type convection-diffusion (diffusion, convection, source, etc.) ainsi que les conditions aux bords et la condition initiale.
   - *BSEDP*, qui en hérite, fournit les expressions spécifiques à l'équation de Black–Scholes (par exemple $\frac{1}{2}\sigma^2 S^2$ pour la diffusion, $r\,S$ pour la convection, etc.). Elle exploite l'objet *Option* pour déterminer le payoff et les valeurs limites en fonction du type d'option (Call, Put, Barrière, etc.).

4. **Mise en œuvre des Différences Finies (*MDF* et *MDFE*)**

   - *MDF* est une classe de base qui définit la structure générale de la méthode : gestion de la discrétisation en temps et en espace, allocation des vecteurs, etc.
   - *MDFE* (pour MDF Explicite) implémente le schéma explicite : dans ses méthodes (*dom*, *boundaries*, etc.), elle met à jour les valeurs de l'option sur la grille spatio-temporelle, calcule les sensibilités (Δ, Γ, Θ) et enregistre le résultat dans un fichier CSV, tout en permettant l'interpolation du prix pour un spot S_0.

5. **Fonction *main()***

   Dans le point d'entrée du programme, on :

   (a) Instancie les payoffs et les options correspondantes.
   (b) Crée les objets *BSEDP* pour l'équation de Black–Scholes et les objets *MDFE* pour la résolution explicite.
   (c) Lance les calculs (méthode *prog_pas*) et extrait le prix interpolé à la date $t = 0$.
   (d) Écrit les données (prix et sensibilités) dans des fichiers CSV.

Cette architecture modulaire permet de changer aisément de payoff (grâce au polymorphisme), de varier les paramètres de l'option (via la classe *Option*) ou de remplacer le schéma numérique (en dérivant une nouvelle classe de *MDF*). Cela facilite la maintenabilité et l'extensibilité du programme.

3 Annexes

3.1 Problèmes rencontrés

Lors de la mise en œuvre de ce projet, deux principaux problèmes se sont présentés.

1. **Génération de deux fichiers de sortie identiques**

   Au début, la fonction *prog_pas()* ne prenait aucun argument permettant de nommer le fichier de sortie. Ainsi, lorsque nous souhaitions lancer la résolution pour deux options différentes, le second fichier de résultats écrasait systématiquement le premier.

   Pour éviter cela, nous avons ajouté un paramètre (par exemple `const std::string& f`) à *prog_pas()*, afin de pouvoir choisir un nom distinct à chaque appel. Même si ce paramètre n'intervient pas directement dans les calculs internes de la fonction, il garantit que deux simulations successives ne produisent pas de fichiers portant le même nom. Cette simple modification permet de conserver plusieurs résultats indépendants sans les écraser.

2. **Estimation de la valeur de l'option en $(0, S_0)$ et interpolation**

   L'objectif essentiel en matière de pricing d'options est de connaître la valeur $\Pi(0, S_0)$ au temps initial, pour un prix du sous-jacent S_0. Cependant, dans notre approche par différences finies, nous construisons une grille spatiale discrète $\{S_i\}$, ce qui signifie que S_0 peut ne pas coïncider avec l'un des points $\{S_i\}$. Sans traitement spécifique, il serait alors impossible de récupérer directement la valeur de Π à la date $t = 0$, en S_0.

   Pour résoudre ce problème, nous avons mis en place une *interpolation linéaire* entre les deux points de la grille encadrant S_0. Concrètement, si $S_i \leq S_0 \leq S_{i+1}$, alors

   $$\Pi(0, S_0) \approx \Pi(0, S_i) \; + \; \frac{S_0 - S_i}{S_{i+1} - S_i} \left[\Pi(0, S_{i+1}) - \Pi(0, S_i)\right].$$

   Ainsi, nous obtenons une approximation continue de la fonction $\Pi(0, \cdot)$ à partir des seuls points calculés sur la grille. Sans cette étape d'interpolation, nous serions passés à côté de l'intérêt même du modèle, car il est indispensable de pouvoir donner un prix pour n'importe quel spot S_0. De plus, cette méthode nous permet de signaler le cas où S_0 se trouve en dehors de la plage discrétisée ($S_0 < S_{\min}$ ou $S_0 > S_{\max}$), déclenchant alors un message d'erreur.

3.2 Aperçu général de l'architecture du programme

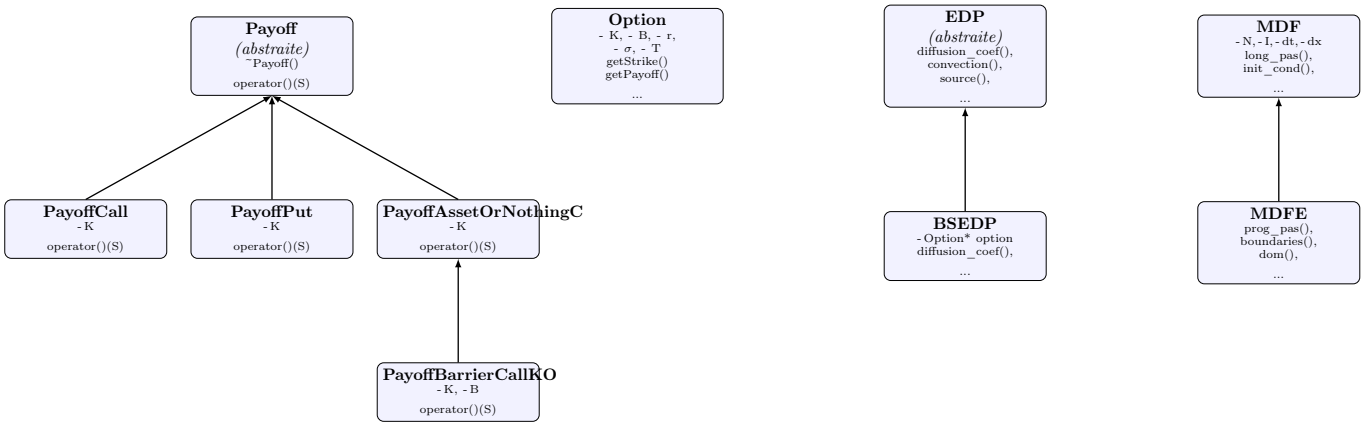

Figure 1: Diagramme simplifié des classes (Payoff, EDP, Option, MDF).

3.3 Plot pour quelques options.

Pour cette section, nous avons choisit les paramètres suivants:

- `K = 0.5` Strike de l'option.
- `r = 0.05` Taux sans risque (5%).
- `sigma = 0.2` Volatilité du sous-jacent (20%).
- `T = 1.00` Maturité de l'option (1 an).
- `x_max = 1.0` Valeur maximale du sous-jacent considérée dans la discrétisation spatiale.

- `I = 20` Nombre de points dans la grille spatiale.
- `N = 20` Nombre de points dans la grille temporelle.
- `B = 0.8` Barrière (pour l'option barrière).
- `S_0 = 0.2` Prix initial du sous-jacent.
- `epsilon = 0.05` Facteur de tolérance pour la condition de stabilité.

3.3.1 Call Européen

On rappelle le payoff d'un call Européen de strike K et de maturié T sur le sous jacent S:

$$C_T = (S_T - K)^+$$

Une implémentation 3D en python, des valeurs et sensibilités obtenues sur les deux grilles, de l'option donne le plot suivant:

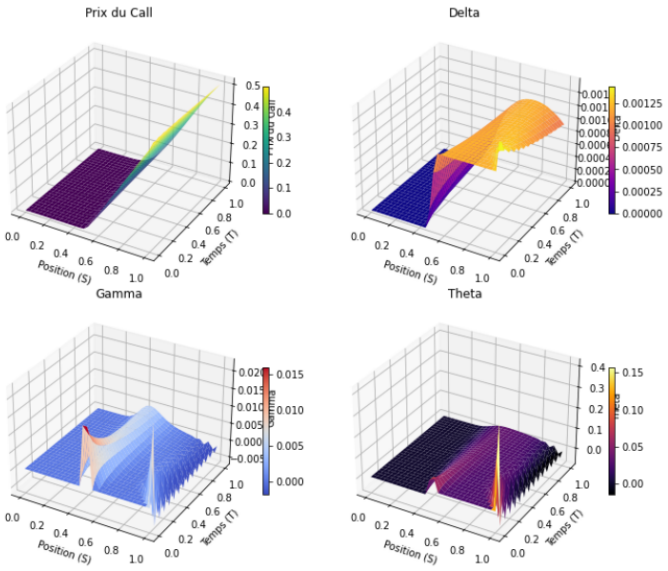

Figure 2: Call Européen.

3.3.2 Put Européen

On rappelle le payoff d'un Put Européen de strike K et de maturité T sur le sous-jacent S.

$$P_T = \max(K - S_T, 0).$$

On a donc:

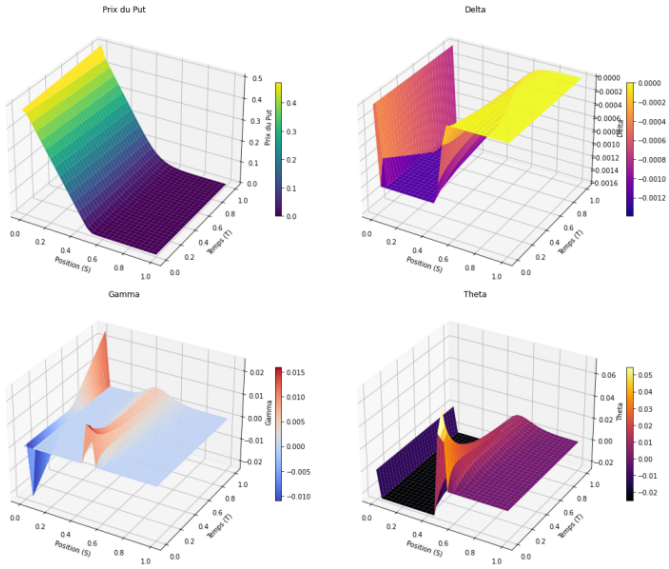

Figure 3: Put Européen.

3.3.3 Asset Or Nothing Call

On rappelle le payoff d'une Option Asset Or Nothing Call de strike K et de maturité T sur le sous-jacent S:

$$A_T = S_T \mathbb{1}_{S_T \geq K}$$

On a donc:

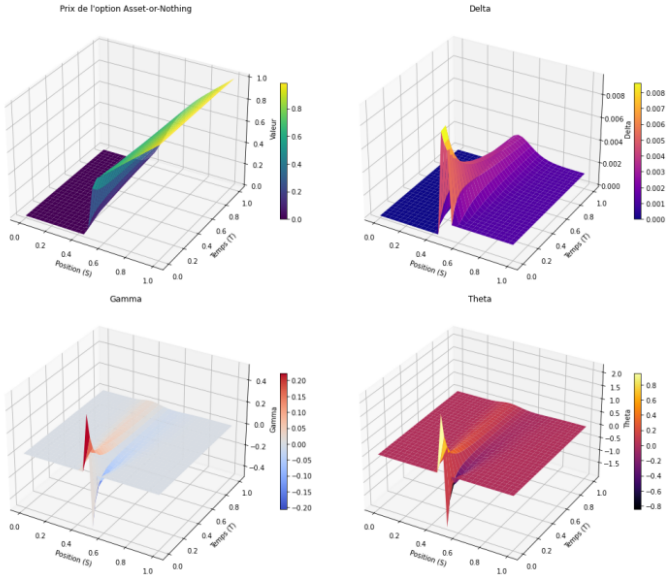

Figure 4: Asset Or Nothing Call.

3.3.4 Call Barrier Knock Out

On rappelle le payoff d'une Option Call Barrier Knock-Out de strike K et de maturité T sur le sous-jacent S, avec une barrière B *observée seulement à maturité* :

$$C_T^{B/KO} = (S_T - K)\,\mathbb{1}_{\{K \leq S_T < B\}}.$$

Autrement dit, l'option se comporte comme un call standard, mais si S_T atteint ou dépasse B à l'échéance, le payoff est nul.

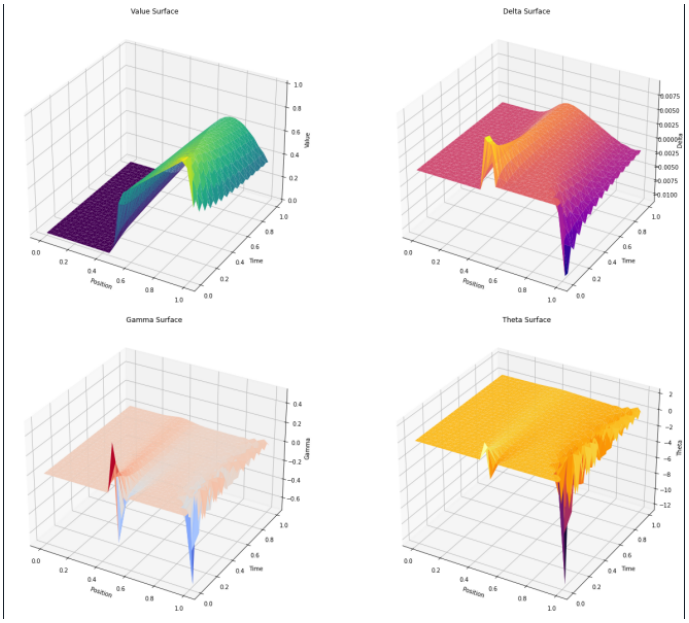

Figure 5: Call Barrier KO (barrière observée seulement à maturité).