ECOLE NATIONALE SUPÉRIEURE D'INFORMATIQUE
POUR L'INDUSTRIE ET L'ENTREPRISE

Rapport de projet Programmation Avancée
Filière Mathématiques appliqués

Détermination du prix des options européennes : Étude de l'équation de Black & Scholes

Présenté par :
ABERQI Nezar
BARNICHA Mohammed Yassine

Supervisé par :
Mr Vincent TORRI

Année universitaire : 2023 - 2024

Table des matières

Introduction Générale **3**

1 Etude de l'équation Black & Scholes **4**
1.1 Présentation de l'équation : 4
1.2 Introduction 4
1.3 Méthode des différences finis 5
1.4 Discrétisation du domaine de résolution 5
1.5 Etude de la résolution par θ-schéma : 6
1.5.1 Objectif de la résolution 7
1.6 Crank Nicholson comme modèle d'un θ-schéma : 7
1.6.1 Résolution du système matriciel tridiagonal : 9

2 Equation de Black Scholes réduite **11**
2.1 Passage de l'EDP complete a l'EDP réduite 11
2.1.1 Résolution de l'équation réduite : 12
2.1.2 Application du schéma implicite : 12
2.2 Remarque importante 13
2.3 Résolution 13
2.3.1 Représentation Matricielle 13

3 Application de la theorie a la machine **14**
3.1 Conception 14
3.1.1 Diagramme UML 14
3.1.2 Description de l'UML et Explication 15
3.2 Realisation 17
3.3 Affichage graphique SDL2 19
3.4 SDL Graphics Rendering 19
3.4.1 RenderWindow_2plots 19
3.4.2 RenderWindow1plot 20

Introduction et Objectifs

Le présent rapport vise à présenter de manière détaillée notre projet réalisé dans le cadre de l'Unité d'Enseignement "Programmation Avancée Projet" du Semestre 3 dans lequel nous avons porté notre choix sur le sujet dédié à l'analyse de l'équation de **Black-Scholes** considérée comme modèle mathématique fondamental en finance pour déterminer les prix des **options** (européennes ou américaines) .
Ce rapport a donc pour objectif de résumer notre parcours, en présentant les différentes approches utilisées, les obstacles rencontrés tant sur le plan mathématique que technique, ainsi que les solutions que nous avons conçues et mises en œuvre pour développer une solution permettant de résoudre la problématique du sujet.
Notre démarche s'est initiée par une phase de recherche conséquente centrée sur l'équation de Black-Scholes et sur les méthodes numériques de résolution d'équations différentielles partielles (**EDP**), notamment les techniques de différences finies, avec un intérêt particulier qui a été accordé aux méthodes de **discrétisation** implicite et de Crank-Nicolson, reconnues pour leur pertinence, leur stabilité et leur efficacité dans le contexte des EDP. Suite à l'assimilation des concepts théoriques, nous avons procédé à l'application concrète de ces méthodes aux EDP spécifiques mentionnées dans notre sujet.
Cette étape pratique a été suivie d'un travail rigoureux sur l'architecture logicielle, matérialisé par la création d'un diagramme UML résumant les différentes classes que l'on fera intervenir dans nos étapes de résolution.
Cette démarche structurée nous a permis de conceptualiser et d'organiser l'implémentation de la solution en language C++, en anticipant les interactions entre les différents composants du programme.
Dans les sections suivantes, nous détaillerons chaque étape de notre projet, en mettant en lumière les raisonnements adoptés, les défis surmontés et les connaissances acquises tout au long de ce parcours.

Chapitre 1

Etude de l'équation Black & Scholes

1.1 Présentation de l'équation :

1.2 Introduction

L'équation de Black-Scholes, du nom de ses créateurs Fischer Black et Myron Scholes, a été présentée en 1973 et représente une percée majeure dans l'évaluation des options financières. Le modèle permet de déterminer le prix d'une option européenne.
L'équation aux dérivées partielles dite **complète** (EDP) , qui décrit l'évolution du prix de l'option estr donnée par la formule suivante :
$C(t,S) : [0,T] \times [0,L] \longrightarrow \mathbb{R}$:

$$\frac{\partial C}{\partial t} + rS\frac{\partial C}{\partial S} + \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 C}{\partial S^2} = rC \tag{1.1}$$

où r est le taux d'intérêt sans risque, σ est la volatilité de l'actif sous-jacent et T est le temps d'échéance de l'option.
Pour faciliter les calculs et pour une application plus directe, une forme réduite de cette équation est souvent employée, cette version plus simplifiée est obtenue par un changement de variable qui élimine les dépendances en r et S.
Les changements de variables qu'on va détailler par la suite vont mener à l'équation dite **réduite** vérifiée par la fonction $\tilde{C}(\tilde{t},\tilde{S}) : [0,T] \times [0,L] \longrightarrow \mathbb{R}$

$$\frac{\partial \tilde{C}}{\partial \tilde{t}} = \mu \frac{\partial^2 \tilde{C}}{\partial \tilde{S}^2}. \tag{1.2}$$

La version réduite est avantageuse car elle offre une complexité réduite et permet d'obtenir des solutions approximatives rapidement, ce qui est indispensable pour le trading et l'analyse financière en temps réel.

A propos de l'équation réduite : On remarque que l'équation réduite contient un coefficient inconnu, qui n'est pas exprimé explicitement en fonction des paramètres de l'option ou du marché, il va donc falloir déterminer l'expression exacte du coefficient μ dans cette équation.

1.3 Méthode des différences finis

// theta w l implicite w l expli -> Méthodes de différence finis.
// Intro sur différen finis
// Le fait qu'on utilise la discrétisation du domaine de résolution

1.4 Discrétisation du domaine de résolution

Notre domaine de résolution concerne les intervalles de :

- Variation du *temps* t : $[0, T]$
- Variation du *prix* S : $[0, L]$

Afin d'implémenter les différentes méthodes de différences finis par la suite, on discrétisera le domaine $\mathscr{D} = [0, T] \times [0, L]$ comme suit :

- $\boldsymbol{t} = (t)_{\boldsymbol{m}}$ $\forall m \in [0, T]$ avec $(t)_{\mathrm{m}}$ = m*dt et dt = $\frac{T}{M}$.
- $\boldsymbol{S} = (S)_{\boldsymbol{j}}$ $\forall j \in [0, T]$ avec $(S)_{\mathrm{j}}$ = j*ds et ds = $\frac{L}{N}$.

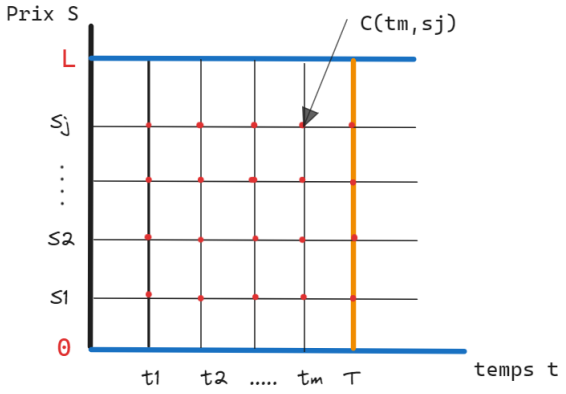

FIGURE 1.1 – Maillage du domaine de résolution $\mathscr{D}$

Analyse du domaine de résolution :

l'EDP de Black & Scholes combine des conditions aux bords et une condition terminale pour déterminer la valeur de l'option à tout moment avant l'échéance, en tenant compte des mouvements du prix de l'actif sous-jacent.

⋆ Les **conditions aux bords**, représentées par des *lignes horizontales* indiquent les cas :

- $S = 0$: Le prix de l'actif sous-jacent est nul.
- $S = L$: Le prix de l'actif sous-jacent atteint une limite L.

Ces conditions nous permettent de spécifier comment le prix de l'option devrait se comporter lorsque le prix de l'actif sous-jacent atteint certaines limites, telles que 0 ou L. Par exemple, pour une option d'achat, nous avons comme indiqué dans la problématique du projet une condition aux bords basses indiquant que lorsque le prix de l'actif tend vers 0, la valeur de l'option doit être nulle.

⋆ La **condition Terminale**, représentée par *la ligne verticale* la condition terminale est une condition qui spécifie la valeur de l'option à l'échéance de l'option.

1.5 Etude de la résolution par θ-schéma :

Le schéma θ est une méthode numérique utilisée pour résoudre des équations aux dérivées partielles (EDP), en particulier celles qui sont paraboliques, comme l'équation de la **chaleur** ou de **diffusion**.
Il s'agit d'une combinaison linéaire des schémas ***explicite*** et ***implicite***, permettant de bénéficier de la stabilité de l'approche implicite et de la simplicité de l'approche explicite.
Le paramètre θ permet de moduler entre ces deux extrêmes.

- $\theta = 0$, le schéma est purement explicite.
- $\theta = 1$, le schéma est purement implicite
- $\theta = 0.5$), le schéma est dit de schéma de Cranck-Nicolson équilibre.

Considérons une fonction $u(t,x)$ ou t et x sont des variables temporelles et spatiales respectivement, le θ-schema permet de discrétiser les différentes dérivés partielles comme suit :

$$u(t,x) \approx \theta u_i^{n+1} + (1-\theta)u_i^n$$
$$\frac{\partial u}{\partial t} \approx \frac{1}{\Delta t}(u_i^{n+1} - u_i^n)$$
$$\frac{\partial u}{\partial x} \approx \theta \frac{u_{i+1}^{n+1} - u_{i-1}^{n+1}}{2\Delta x} + (1-\theta)\frac{u_{i+1}^{n} - u_{i-1}^{n}}{2\Delta x}$$
$$\frac{\partial^2 u}{\partial x^2} \approx \theta \frac{u_{i+1}^{n+1} - 2u_i^{n+1} + u_{i-1}^{n+1}}{\Delta x^2} + (1-\theta)\frac{u_{i+1}^{n} - 2u_i^{n} + u_{i-1}^{n}}{\Delta x^2}$$

Utiliser la méthode θ permet de contrôler la stabilité et la précision de la résolution, ce qui est essentiel dans de nombreuses simulations numériques.

1.5.1 Objectif de la résolution

Notre objectif va être de déterminer la valeur de l'option au temps initial 0 : $\{C(0,S_j), \forall j \in [1,N-1]\}$ dans le cas complet et réduit $\{\widetilde{C}(T,\widetilde{S}_j), \forall j \in [1,N-1]\}$.
Notre objectif va aussi porter sur la représentation graphique de $\{C(0,S_j), \forall j \in [1,N-1]\}$ à la fois dans le cas complet et réduit afin de visualiser les deux approches de résolution et effectuer une analyse d'erreur.

1.6 Crank Nicholson comme modèle d'un θ-schéma :

En choisissant θ= 0.5 , le schéma de Crank-Nicolson va nous permettre dans un premier temps de résoudre l'équation différentielle de Black-Scholes complète.
En va inclure les discrétisation du modèle dans l'équation complète et développer l'expression de l'équation afin de retrouver une expression explicite du prix de l'option au temps n C_j^n en fonction de son prix au temps C_j^{n+1}.
Remarque : L'idée de base étant qu'on connaît au préalable les valeurs C_j^{n+1}, puisqu'on commence par une condition terminale et qu'on revient à rebours dans le temps afin de déterminer les valeurs C_j^n antérieures. On développe l'EDP complète en introduisant les différentes discrétisations :
Soit n $\in[1,M-1]$ et j $\in[1,N-1]$ on a :

$$\frac{1}{\Delta t}(C_j^{n+1}-C_j^n)+\frac{rj}{4}(C_{j+1}^{n+1}-C_{j-1}^{n+1}+C_{j+1}^n-C_{j-1}^n)+\frac{1}{4}\sigma^2 j^2(C_{j+1}^{n+1}-2C_j^{n+1}+C_{j-1}^{n+1}+C_{j+1}^n-2C_j^n+C_{j-1}^n) \\ -\frac{r}{2}(C_j^{n+1}+C^n)=0 \quad (1.3)$$

Pour pouvoir résoudre l'équation, on essaiera de regrouper les termes dont l'aspect temporel est similaire : l'instant t d'indice n ou $n+1$,

$$(\frac{1}{\Delta t}-\frac{r}{2})C_j^{n+1}+\frac{rj}{4}(C_{j+1}^{n+1}-C_{j-1}^{n+1})+\frac{\sigma^2 j^2}{4}(C_{j+1}^{n+1}-2C_j^{n+1}+C_{j-1}^{n+1})= \\ (\frac{1}{\Delta t}+\frac{r}{2})C_j^n-\frac{rj}{4}(C_{j+1}^n-C_{j-1}^n)-\frac{\sigma^2 j^2}{4}(C_{j+1}^n-2C_j^n+C_{j-1}^n) \quad (1.4)$$

Pour mieux exprimer l'équation, on peut penser à éliminer t par une simple multiplication et on essaient d'exprimer les coefficients existants devant chaque valeur du prix de l'option durant les deux temps distincts.

$$C_{j-1}^{n+1}(-\frac{rj\Delta t}{4}+\frac{\sigma^2 j^2\Delta t}{4})+C_j^{n+1}(1-\frac{r\Delta t}{2}-\frac{\sigma^2 j^2\Delta t}{2})+C_{j+1}^{n+1}(\frac{rj\Delta t}{4}+\frac{\sigma^2 j^2\Delta t}{4})= \\ C_{j-1}^n(\frac{rj\Delta t}{4}-\frac{\sigma^2 j^2\Delta t}{4})+C_j^n(1+\frac{r\Delta t}{2}+\frac{\sigma^2 j^2\Delta t}{2})+C_{j+1}^n(-\frac{rj\Delta t}{4}-\frac{\sigma^2 j^2\Delta t}{4}) \quad (1.5)$$

Remarque : Les coefficients des membres de l'équation dépendent du prix de l'option j, On effectuera par la suite une indexation des différents coefficients pour une meilleur lisibilité et interprétabilité de l'équation objet de notre analyse.

$$\boxed{u_j C_{j-1}^{n+1}+v_j C_j^{n+1}+w_j C_{j+1}^{n+1}=a_j C_{j-1}^n+b_j C_j^n+c_j C_{j+1}^n} \quad (1.6)$$

Expressions des coefficients :
Soit j $\in[1, N-1]$:

$$u_j = \frac{j\Delta t}{4}(\sigma^2 j - r)$$
$$v_j = 1 - \frac{1}{2}r\Delta t - \frac{1}{2}\sigma^2 j^2 \Delta t$$
$$w_j = \frac{j\Delta t}{4}(\sigma^2 j + r)$$
$$a_j = \frac{j\Delta t}{4}(r - \sigma^2 j) = -u_j$$
$$b_j = 1 + \frac{1}{2}r\Delta t + \frac{1}{2}\sigma^2 j^2 \Delta t$$
$$c_j = -\frac{j\Delta t}{4}(\sigma^2 j + r) = -w_j$$

-> Comme l'équation récursive est vérifiée par tous les prix j, on pensera donc à effectuer une opération de ***vectorisation*** afin de résoudre l'équation le plus optimalement possible et éviter les calculs par des boucles redondantes.
Pour résoudre les différents systèmes linéaires correspondants aux temps n $\in[0, M]$, on regroupe les coefficients de l'équation (1.6) dans des matrices carrées, afin de se positionner dans le système matriciel suivant :

$$\underbrace{\begin{pmatrix} u_1 C_0^{n+1} \\ 0 \\ \vdots \\ 0 \\ w_{N-1} C_N^{n+1} \end{pmatrix}}_{(N-1)\times 1} + \underbrace{\begin{pmatrix} v_1 & w_1 & 0 & \dots & 0 \\ u_2 & v_2 & \ddots & \ddots & \vdots \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & \ddots & \ddots & \ddots & w_{N-2} \\ 0 & \dots & 0 & u_{N-1} & v_{N-1} \end{pmatrix}}_{(N-1)\times(N-1)} \underbrace{\begin{pmatrix} C_1^{n+1} \\ \vdots \\ \vdots \\ \vdots \\ C_{N-1}^{n+1} \end{pmatrix}}_{(N-1)\times 1} = \underbrace{\begin{pmatrix} -u_1 C_0^n \\ 0 \\ \vdots \\ 0 \\ -w_{N-1} C_N^n \end{pmatrix}}_{(N-1)\times 1} + \underbrace{\begin{pmatrix} b_1 & -w_1 & 0 & \dots & 0 \\ -u_2 & b_2 & \ddots & \ddots & \vdots \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & \ddots & \ddots & \ddots & -w_{N-2} \\ 0 & \dots & 0 & -u_{N-1} & b_{N-1} \end{pmatrix}}_{(N-1)\times(N-1)} \underbrace{\begin{pmatrix} C_1^n \\ \vdots \\ \vdots \\ \vdots \\ C_{N-1}^n \end{pmatrix}}_{N-1)\times 1}$$

$$\underbrace{\begin{pmatrix} v_1 & w_1 & 0 & \dots & 0 \\ u_2 & v_2 & \ddots & \ddots & \vdots \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & \ddots & \ddots & \ddots & w_{N-2} \\ 0 & \dots & 0 & u_{N-1} & v_{N-1} \end{pmatrix}}_{(N-1)\times(N-1)} \underbrace{\begin{pmatrix} C_1^{n+1} \\ \vdots \\ \vdots \\ \vdots \\ C_{N-1}^{n+1} \end{pmatrix}}_{(N-1)\times 1} + \underbrace{\begin{pmatrix} u_1(C_0^{n+1} + C_0^n) \\ 0 \\ \vdots \\ 0 \\ w_{N-1}(C_N^{n+1} + C_N^n) \end{pmatrix}}_{(N-1)\times 1} = \underbrace{\begin{pmatrix} b_1 & -w_1 & 0 & \dots & 0 \\ -u_2 & b_2 & \ddots & \ddots & \vdots \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & \ddots & \ddots & \ddots & -w_{N-2} \\ 0 & \dots & 0 & -u_{N-1} & b_{N-1} \end{pmatrix}}_{(N-1)\times(N-1)} \underbrace{\begin{pmatrix} C_1^n \\ \vdots \\ \vdots \\ \vdots \\ C_{N-1}^n \end{pmatrix}}_{(N-1)\times 1}$$

On exprime la formule de recurrence finale par la relation :

$$\boxed{U_1 C^{n+1} + b_{n+1} = U_2 C^n}$$

- Les matrices U_i sont les matrices des coefficients exprimés précédemment
- Le vecteur colonne b représente les deux conditions aux bords basse et hausse vérifiées par l'option C.

1.6.1 Résolution du système matriciel tridiagonal :

Notre objectif étant de déterminer le vecteur C^n, nous allons donc suivre les étapes suivantes :

Etape 0 : *Déterminer les valeurs des vecteurs des coefficients* :

$$U = \begin{pmatrix} u_1 \\ u_2 \\ \vdots \\ u_{N-1} \end{pmatrix} \in \mathbb{R}^{N-1}, \quad V = \begin{pmatrix} v_1 \\ v_2 \\ \vdots \\ v_{N-1} \end{pmatrix} \in \mathbb{R}^{N-1}, \quad W = \begin{pmatrix} w_1 \\ w_2 \\ \vdots \\ w_{N-1} \end{pmatrix} \in \mathbb{R}^{N-1}, \quad B = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_{N-1} \end{pmatrix} \in \mathbb{R}^{N-1}$$

Etape 1 : *Représentation des matrices en mémoire*
On remarque que les matrices U_1 et U_2 sont tridiagonales, ainsi, ces matrices sont essentiellement constituées de valeurs nulles.
Pour remédier à ceci, on pourrait opter pour une méthode de stockage beaucoup plus efficace pour éviter de stocker $(N-1)^2$ valeurs en mémoires majoritairement creuses :

+ **Stockage en Bande (Band Storage)**
  - Principe : Stocke uniquement les bandes diagonales non nulles.
  - Complexité en mémoire : $3N$ éléments pour une matrice $N \times N$.
+ **Stockage Diagonal Compact (Compact Diagonal Storage)**
  - Principe : Stocke les éléments non nuls des trois diagonales.
  - Complexité en mémoire : $3N-2$ éléments.
+ **Object Matrice avec les diagonales comme membres de donnees :**
  - Principe : Représente une matrice tridagonale comme un objet avec trois membres de données pour la diagonale, la diagonale inférieure (lower diagonal) et la diagonale supérieure (upper diagonal).
  - Complexité en mémoire : $3N-2$ éléments.

Par la suite, nous allons choisir la méthode orientée objet afin de stocker les matrices des coefficients de façon optimale. **Etape 2 :** *Calculer les conditions au bords de la fonction du prix*
On calcule par la suite les conditions aux bords pour chaque instant n $\in[0,M]$ afin de déterminer la valeur du vecteur b.
Etape 3 : *Resolution du systeme lineaire* :
Notre objectif étant de resoudre le système linéaire suivant :

$$\boxed{U_2 C^n = U_1 C^{n+1} + b_{n+1} \forall n \in [0,M]}$$

qui est sous le forme de :

$$\boxed{Ax = b}$$

Il existe donc plusieurs algorithmes qu'on pourrait appliquer afin de déterminer C^n :

1. **Factorisation LU**
   - *Principe* : La factorisation LU décompose la matrice A en un produit de deux matrices, une matrice triangulaire inférieure L et une matrice triangulaire supérieure U telles que $A = LU$. Cela permet de résoudre le système en deux étapes, en résolvant d'abord $Ly = b$ puis $Ux = y$.
   - *Avantage en termes de complexité en mémoire* : La complexité en mémoire est relativement faible car on peut stocker L et U dans l'espace occupé par A.
   - *Complexité de calcul* : La factorisation LU a une complexité de calcul de $O(n^3)$ pour une matrice dense, mais pour une matrice tridiagonale, cela se réduit à $O(n)$ car seule une fraction des éléments doit être manipulée.
2. **Factorisation QR**
   - *Principe* : La factorisation QR décompose A en un produit d'une matrice orthogonale Q et une matrice triangulaire supérieure R, donc $A = QR$. Le système est ensuite résolu en calculant $Q^\top b$ suivi de la résolution de $Rx = Q^\top b$.
   - *Avantage en termes de complexité en mémoire* : Elle nécessite plus de mémoire que la méthode LU pour stocker Q et R.
   - *Complexité de calcul* : Pour une matrice dense, la factorisation QR a une complexité de $O(n^3)$, mais des optimisations pour les matrices tridiagonales peuvent réduire la complexité.
3. **Algorithme de Thomas [Élimination tridiagonale]**
   - *Principe* : Il s'agit d'une adaptation spécialisée de la factorisation LU pour les matrices tridiagonales. L'algorithme de Thomas élimine les éléments en dehors de la diagonale principale en une seule passe à travers les éléments de la matrice.
   - *Avantage en termes de complexité en mémoire* : **Ne nécessite que le stockage des trois diagonales de la matrice, ce qui est très efficace en termes de mémoire.**
   - é de calcul La complexité de l'algorithme de Thomas est $O(n)$, ce qui en fait l'une des méthodes les plus rapides pour les matrices tridiagonales.

Notre choix dans ce projet a porté sur L'**algorithme de Thomas** pour résoudre le système linéaire en raison de sa complexité linéaire et de sa faible empreinte mémoire.

Chapitre 2

Equation de Black Scholes réduite

2.1 Passage de l'EDP complete a l'EDP réduite

L'équation de Black & Scholes complete est exprimée par l'EDP suivante :

$$\frac{\partial C}{\partial t} + rS\frac{\partial C}{\partial S} + \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 C}{\partial S^2} - rC = 0 \tag{2.1}$$

Nous pourrons espérer que la valeur d'une option soit un processus **positif**, suggérant ainsi une fonction positive, peut-être exponentielle.
On peut remarquer également que pour chaque $\frac{\partial}{\partial S}$ il y a un terme S correspondant.
Bien que cela soit nécessaire pour que les dimensions fonctionnent, cela pourrait aussi suggérer que chaque fois que nous dérivons, nous obtenons la même chose plus un terme S. Cela sonne comme rappelant la fonction exponentielle.
Avec tout cela dit, le fait de brancher un **processus exponentiel** semble être une bonne idée, et serait l'une des premières choses à essayer.
En reprenant la dernière observation, la deuxième chose à remarquer avant de commencer est que l'EDP est linéaire et semble avoir des paires d'opérateurs $S\frac{\partial}{\partial S}$.
Réorganiser l'opérateur du second ordre de cette manière semble être une première étape sensée avant un changement de variables. Cela donne :

$$\frac{\partial C}{\partial t} + \frac{1}{2}\sigma^2 \left(S\frac{\partial}{\partial S}\right)^2 C + \left(r - \frac{1}{2}\sigma^2\right) S\frac{\partial C}{\partial S} - rC = 0.$$

Maintenant en se basant sur notre intuition précédente on va poser le premier changement de variable :

$$\boxed{S = \exp(y)}$$

ceci nous mène à :

$$\frac{\partial C}{\partial t} + \frac{1}{2}\sigma^2 \frac{\partial^2 C}{\partial y^2} + \left(r - \frac{1}{2}\sigma^2\right)\frac{\partial C}{\partial y} - rC = 0.$$

En tenant compte du fait que nous cherchons à déterminer la valeur de l'option à un moment antérieur à sa maturité, nous introduisons la transformation

$$\boxed{\tilde{t} = T - t}$$

, menant à l'équation :

$$\frac{\partial C}{\partial \tilde{t}} - \frac{1}{2}\sigma^2 \frac{\partial^2 C}{\partial y^2} - (r - \frac{1}{2}\sigma^2)\frac{\partial C}{\partial y} + rC = 0$$

En examinant le dernier terme et en le comparant à la dérivée temporelle, nous postulons que $C = \exp(rt)$ pourrait être une partie de la solution. Cette hypothèse nous amène à substituer

$$\boxed{\tilde{C} = \exp(r\tilde{t})C}$$

, simplifiant l'équation en :

$$\frac{\partial \tilde{C}}{\partial \tilde{t}} - \frac{1}{2}\sigma^2 \frac{\partial^2 \tilde{C}}{\partial y^2} - (r - \frac{1}{2}\sigma^2)\frac{\partial \tilde{C}}{\partial y} = 0$$

Finalement, en considérant $\tilde{t}$ comme un analogue du temps et y comme un analogue de l'espace, nous effectuons une dernière transformation

$$\boxed{\tilde{s} = y + (r - \frac{1}{2}\sigma^2)(T - \tilde{t})}$$

nous menant ainsi à l'équation dite de de la **chaleur** bien connue :

$$\frac{\partial \tilde{C}}{\partial \tilde{t}} = \frac{1}{2}\sigma^2 \frac{\partial^2 \tilde{C}}{\partial \tilde{s}^2}$$

Ce qui nous permet donc de reconnaître le coefficient μ dans l'équation réduite du projet :

$$\boxed{\mu = \frac{1}{2}\sigma^2}$$

2.1.1 Résolution de l'équation réduite :

Notre objectif étant de déterminer le prix de l'option à l'instant t = 0, nous allons en premier temps déterminer à quoi correspond cette valeur si on travaille avec la nouvelle fonction . En se basant sur les changements de variables effectuées précédemment pour atteindre l'équation réduite finale on peut exprimer $C(0, S_j)$ en fonction de $\tilde{C}$ par la relation suivante :

$$\boxed{C(0,s) = exp(-rT)\tilde{C}(T, log(s))}$$

Remarque importante : Comme l'indique la derniere equation permettant d'évaluer C(0,.), évaluer $log(s)$ nécessite que s commence d'un nombre l très proche de 0 au lieu de 0 lui-même pour éviter le calcul de $\tilde{C}$ en $-\infty$.

2.1.2 Application du schéma implicite :

Supposons qu'on est positionné dans un point (t_n, S_j), alors l' application du θ-schéma pour le cas implicite $\theta = 1$ implique les formules de discrétisations suivantes :

$$\frac{\partial \tilde{C}}{\partial \tilde{t}} \approx \frac{1}{\Delta \tilde{t}}(\tilde{C}_j^{n+1} - \tilde{C}_j^n)$$
$$\frac{\partial^2 \tilde{C}}{\partial \tilde{S}^2} \approx \frac{\tilde{C}_{j+1}^{n+1} - 2\tilde{C}_j^{n+1} + \tilde{C}_{j-1}^{n+1}}{\Delta \tilde{S}^2}$$

2.2 Remarque importante

La nouvelle fonction $\tilde{C}$ est caractérisée par une condition initiale et non pas terminale comme dans le cas de l'équation complète. Ceci suggère qu'on avancera dans le temps à partir de $t = 0$. Ainsi, on cherchera à retrouver les $\tilde{C}_j^{n+1}$ dans une approche dite **progressive**.

2.3 Résolution

On remplace les différentielles par leurs discrétisations respectives et on obtient l'équation suivante en posant $K = \frac{\mu \Delta t}{(\Delta s)^2}$, d'où :

$$\boxed{mj = -Km+1j-1+\Big(1+2K\Big)m+1j-Km+1j+1} \tag{2.2}$$

2.3.1 Représentation Matricielle

En notation matricielle, on obtient :

$$M_1 = \begin{pmatrix} b & c & 0 & \dots & 0 \\ a & \ddots & \ddots & \ddots & \vdots \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & \ddots & \ddots & \ddots & c \\ 0 & \dots & 0 & a & b \end{pmatrix} \in \mathscr{M}_{N-1}() \tag{2.3}$$

avec $a = c = -K$ et $b = 1 + 2K$. De plus, on note :

$$\boxed{C_n = (C_{n,1}, \dots, C_{n,N-1})^{\mathbf{T}} \in \mathbb{R}^{N-1} \text{ et } b_m = (a_1 C_{n,0}, 0, \dots, 0, c_{N-1} C_{n,N})^{\mathbf{T}} \in \mathbb{R}^{N-1}} \tag{2.4}$$

On obiten alors le système suivant :

$$\boxed{M_1 C^{m+1} = C^m - b_{m+1}} \tag{2.5}$$

$$\boxed{Ax = B} \tag{2.6}$$

Chapitre 3

Application de la theorie a la machine

3.1 Conception

- Expliquer lalgorithme de resolution - UML.Diag - Resume sur le diagramme UML

3.1.1 Diagramme UML

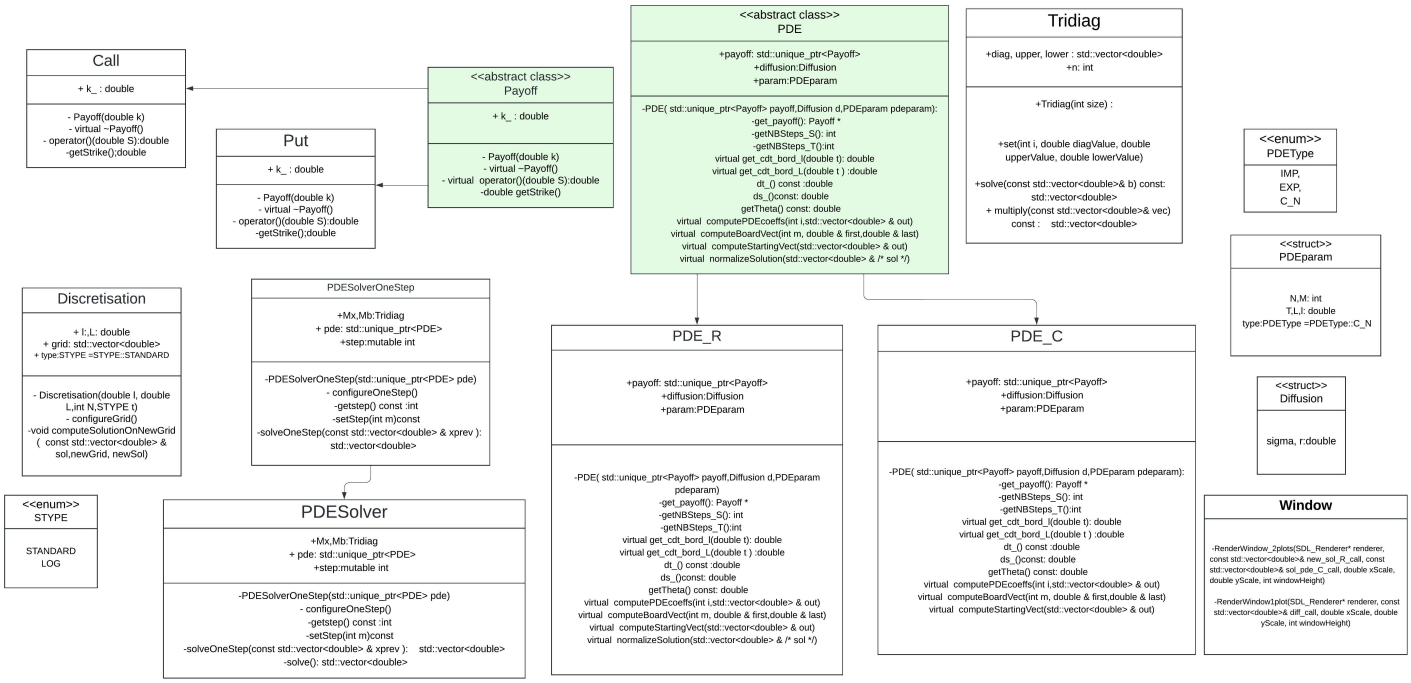

Figure 3.1 – Diagramme UML du projet

3.1.2 Description de l'UML et Explication

Dans ce projet, nous avons développé une architecture de code centrée autour de la classe PDE, conçue pour modéliser à la fois l'équation différentielle partielle (EDP) complète et réduite, grâce à l'utilisation judicieuse de l'héritage avec les classes dérivées PDE_C (pour la forme complète) et PDE_R (pour la forme réduite). Cette classe PDE intègre tous les paramètres essentiels de l'EDP, organisés en structures pour améliorer la lisibilité et la maintenabilité du code. Notamment, la structure `Diffusion` encapsule les paramètres du modèle de diffusion, tels que σ et r, tandis que `PDEParams` regroupe les paramètres numériques de l'EDP, y compris les domaines de résolution (l, L, T), les nombres d'intervalle (N et M), et un énumérateur pour sélectionner le schéma de calcul désiré (`IMPLICIT`, `EXPLICIT` ou `CRANCK NICHOLSON`).

Cette classe contient aussi un attribut représentant le payoff de l'option dont nous voulons calculer le prix. Cet attribut est représenté par un pointeur intelligent vers un objet de type `Payoff`. Cette décision est motivée par le caractère abstrait de la classe `Payoff`, qui ne peut être instanciée directement. En utilisant un pointeur intelligent, nous bénéficions d'une gestion de mémoire automatique et sécurisée, éliminant les risques de fuites de mémoire tout en permettant une flexibilité accrue dans le traitement des différents types d'options, tels que les *Calls* et les *Puts*.

La classe PDE est dotée de méthodes clés pour le calcul de l'EDP, y compris `ComputeBoardVect` pour le calcul des conditions aux bords (en 0 et L), `ComputeStartingVect` pour le calcul de la condition initiale (représentant la valeur en T de C pour l'EDP complète et la valeur de $\tilde{C}$ en 0 pour l'EDP réduite), et `computePDECoeffs` pour déterminer les coefficients des matrices tridiagonales nécessaires.

Ensuite, pour la résolution numérique de l'EDP, nous avons introduit la classe `PDESolver`, qui hérite de `PDESolverOneStep`. La classe `PDESolverOneStep` joue un rôle essentiel dans la solution de l'EDP entre deux instants successifs, fournissant la solution en t_{m+1} à partir de t_m pour l'EDP réduite, et inversement pour l'EDP complète. Cette classe est équipée des attributs suivants :

- `Mx` : un objet de la classe `Tridiag` représentant la matrice tridiagonale à résoudre pour effectuer la transition temporelle.
- `Mb` : également un objet `Tridiag`, utilisé pour calculer le second membre du système à résoudre à l'instant t_m.
- PDE : un `std::unique_ptr` vers l'objet PDE, garantissant une gestion de mémoire optimisée et sûre, en phase avec notre utilisation précédente de pointeurs intelligents pour le `Payoff`.
- `m` : un attribut mutable représentant l'instant actuel de la résolution, qui peut changer sans modifier l'objet lui-même.

Les méthodes principales de `PDESolverOneStep` incluent `configureOneStep`, qui calcule `Mx` et `Mb` à partir de l'objet PDE, et `solveOneStep`, qui résout le système tridiagonal à l'instant présent pour avancer à l'instant suivant. `PDESolver` étend `PDESolverOneStep` en ajoutant la méthode `solve`, permettant la résolution complète de l'EDP sur l'ensemble de l'intervalle $[0, M]$, en partant de la condition initiale calculée par la méthode `ComputeStartingVect` de PDE et en itérant avec `solveOneStep`.

Un aspect crucial de la résolution numérique de l'EDP concerne la représentation et la manipulation des matrices tridiagonales. À cet effet, la classe `Tridiag` a été conçue pour encapsuler efficacement une matrice tridiagonale. Elle se compose de trois vecteurs, `lower`, `upper`, et `diag`, représentant respectivement la sous-diagonale, la sur-diagonale, et la diagonale de la matrice. Cette structure permet non seulement une représentation compacte et efficiente des matrices tridiagonales mais aussi une manipulation précise lors des opérations de calcul. La méthode `multiply` de la classe `Tridiag` facilite la multiplication d'un vecteur par une matrice tridiagonale, tandis que la méthode `solve` implémente l'algorithme de Thomas pour résoudre le système linéaire $Ax = b$, où A est une matrice tridiagonale.

En plus des classes précédemment décrites, nous avons introduit la classe `Discretization`, conçue pour accomplir deux objectifs clés dans l'analyse de nos résultats. Le premier objectif est de calculer le vecteur représentant la variable S entre 0 et L, crucial pour tracer la solution de l'EDP avec SDL. Cette classe gère habilement les différences entre les EDP complètes et réduites. Pour l'EDP complète, le calcul de S est direct, avec $S_i = l + (L - l) \times \frac{i}{N}$. En revanche, pour l'EDP réduite, le processus est légèrement plus complexe : nous calculons d'abord $\tilde{s}_i = \log(l) + \log(\frac{L}{l}) \times \frac{i}{N}$, puis nous en déduisons $S_i = \exp(\tilde{s}_i)$.

Le second objectif de `Discretization` est de permettre la comparaison de la solution obtenue avec l'EDP réduite sur la même grille que celle utilisée pour l'EDP complète. Cela est d'une importance capitale, car les deux grilles diffèrent – l'une étant uniforme et l'autre log-uniforme. `Discretization` facilite le passage entre ces deux grilles par une interpolation linéaire en S de la solution. Cette capacité de transition et de comparaison entre différentes grilles de discrétisation est essentielle pour une analyse précise et complète de l'erreur entre les différentes approches de résolution.

3.2 Realisation

Resultats :

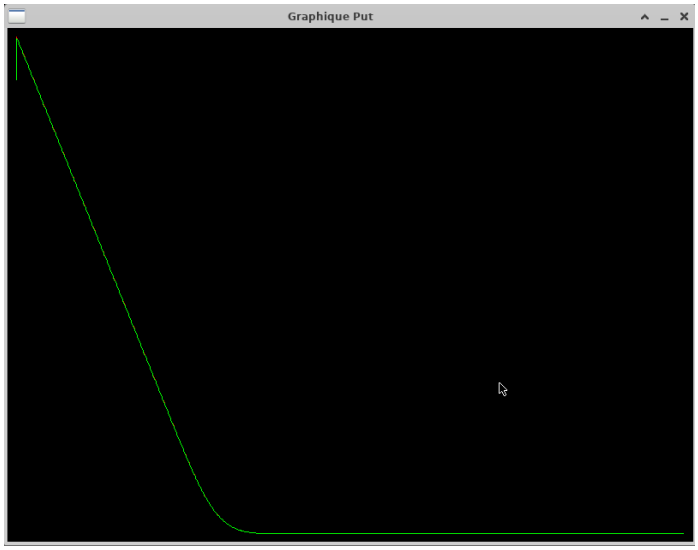

FIGURE 3.2 – Tracage des courbes des deux solutions pour un PUT

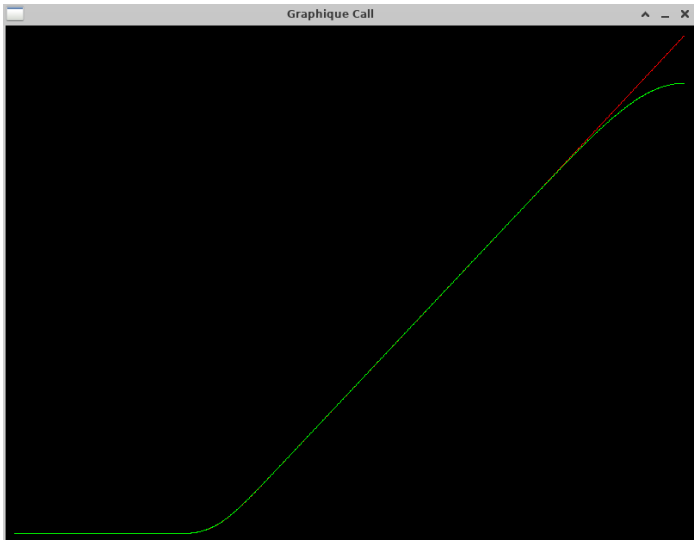

FIGURE 3.3 – Tracage des courbes des deux solutions pour un CALL

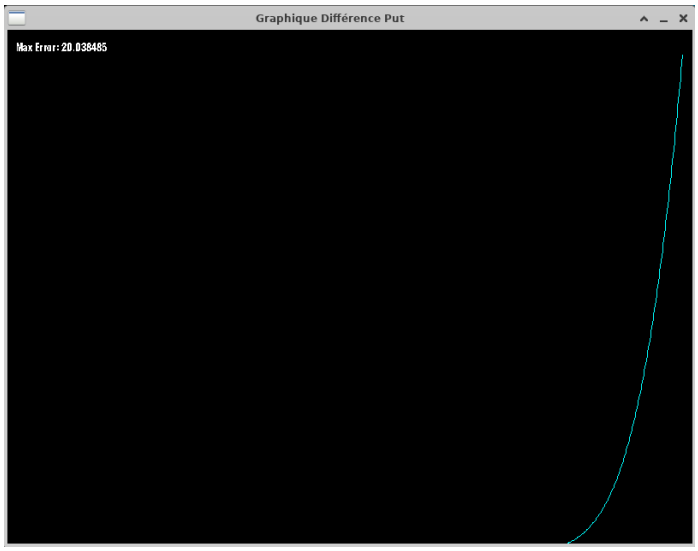

FIGURE 3.4 – la difference absolue entre les deux courbes pour un CALL

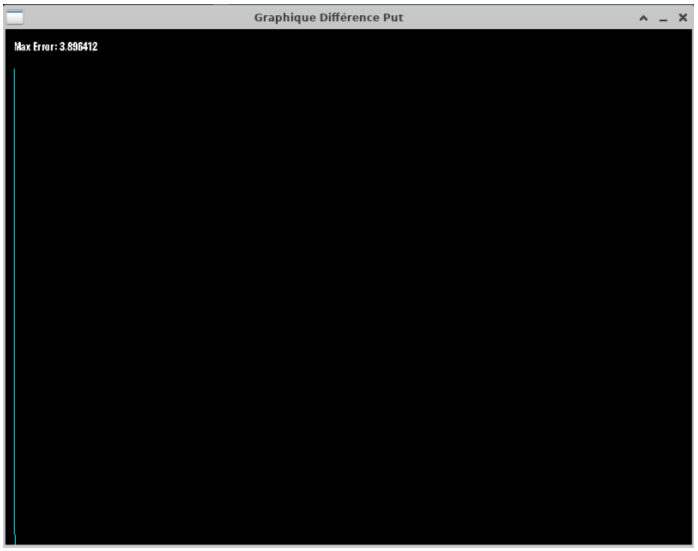

FIGURE 3.5 – la difference absolue entre les deux courbes pour un PUT

Dans notre analyse des graphiques comparant les prix des options Call et Put obtenus via les deux approches - l'EDP complète et l'EDP réduite - nous observons une concordance très satisfaisante.

Pour le Put : À l'exception du premier point ($S_0 = 0.3$), où l'erreur absolue atteint environ 3 unités de prix, l'erreur entre les deux solutions reste inférieure à 0.02 sur tous les autres points. La divergence notable au premier point s'explique par le fait que l'EDP réduite est résolue sur l'intervalle $[0.5, 300]$ et non $[0, 300]$, nécessitant ainsi une extrapolation pour $S_0 = 0.3$ depuis la valeur en 0.5 (au lieu d'une interpolation linéaire pour les autres points).

Pour le Call : Nous notons également une erreur absolue raisonnable :

- $\text{Err}(S_i) < 0.1$ pour $S_i < 235$
- $0.1 < \text{Err}(S_i) < 1$ pour $S_i < 258$

Cependant, au-delà d'une certaine valeur de S_i, l'erreur augmente de manière significative, atteignant jusqu'à 20 unités de prix vers la fin. Malgré une recherche approfondie d'éventuels bugs, cette augmentation semble cohérente avec la méthode utilisée. Dans l'EDP réduite, la discrétisation uniforme de la variable $\tilde{s}$ et le passage subséquent à la variable d'intérêt S, via une exponentielle, entraînent une discrétisation non uniforme en S. Cette méthode tend à discrétiser de manière plus dense les petites valeurs de S et moins densément les grandes valeurs. Par exemple, l'intervalle entre S_{N-2} et S_{N-1} est équivalent à celui d'une discrétisation avec 158 intervalles dans l'approche de l'EDP complète, ce qui affecte considérablement la précision des résultats pour les grandes valeurs de S dans l'EDP réduite.

Comparaison :

Enfin, pour valider nos résultats, nous les avons comparés à ceux obtenus avec la formule de Black-Scholes (la solution exacte de l'EDP complète, *la comparaison s'est faite sur R en utilisant des fichiers csv générés à partir de la discrétisation spatiale et les 4 vecteurs solutions*) : En effet, pour une option call européenne, la formule de Black-Scholes est :

$$C(S,t) = S\Phi(d_1) - Ke^{-r(T-t)}\Phi(d_2)$$

Et pour une option put européenne :

$$P(S,t) = Ke^{-r(T-t)}\Phi(-d_2) - S\Phi(-d_1)$$

où :

- $C(S,t)$ et $P(S,t)$ sont respectivement les prix de l'option call et put,
- S est le prix actuel de l'actif sous-jacent,
- K est le prix d'exercice de l'option,
- T est le temps jusqu'à l'échéance,
- t est le temps actuel,
- r est le taux d'intérêt sans risque,
- Φ est la fonction de répartition de la loi normale standard,
- $d_1 = \frac{\ln\left(\frac{S}{K}\right) + \left(r + \frac{\sigma^2}{2}\right)(T-t)}{\sigma\sqrt{T-t}}$,
- $d_2 = d_1 - \sigma\sqrt{T-t}$,
- et σ est la volatilité de l'actif sous-jacent.

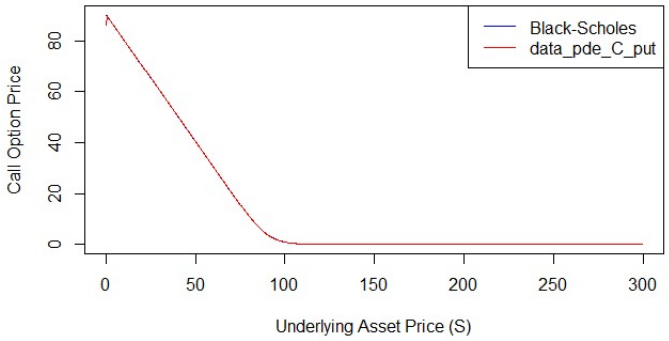

FIGURE 3.6 – Solution analytique BC vs solution PDE C : PUT

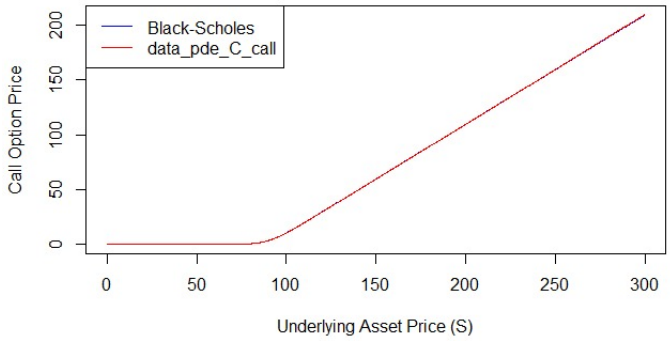

FIGURE 3.7 – Solution analytique BC vs solution PDE C : CALL

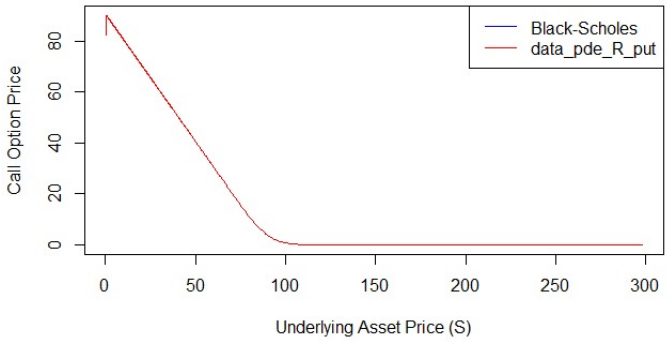

FIGURE 3.8 – Solution analytique BC vs solution PDE R : PUT

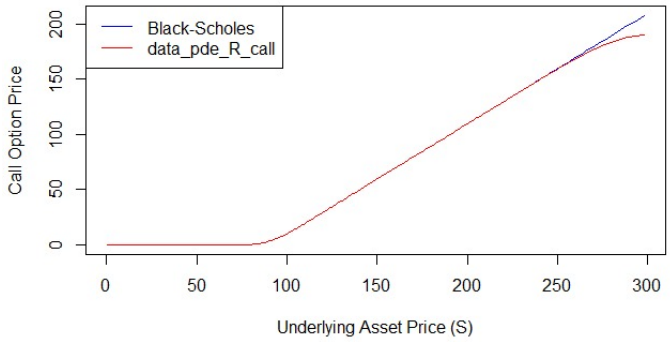

FIGURE 3.9 – Solution analytique BC vs solution PDE R : CALL

La comparaison des résultats obtenus avec l'EDP complète avec ceux de la formule fermée montre une excellente corrélation (erreur absolue < 0.1), validant ainsi l'efficacité de notre approche. Cette concordance nous a assurée quant à la fiabilité des calculs et la pertinence de nos méthodes de résolution.

3.3 Affichage graphique SDL2

3.4 SDL Graphics Rendering

Dans cette application , nous utilisons SDL (Simple DirectMedia Layer) pour afficher graphiquement les solutions des équations différentielles partielles (EDP) utilisées dans le modèle de Black-Scholes pour l'évaluation d'options. Le code comprend principalement deux fonctions pour le rendu graphique.

3.4.1 RenderWindow_2plots

La fonction `RenderWindow_2plots` est conçue pour tracer deux graphiques dans une seule fenêtre SDL. Elle prend en entrée un `renderer` SDL, deux vecteurs de `double` (représentant les solutions originale et nouvelle de l'EDP pour les options d'achat), des échelles pour les axes x et y, et la hauteur de la fenêtre. Cette fonction utilise des couleurs différentes pour distinguer entre la solution originale (rouge) et la nouvelle solution (verte) de l'EDP.

```
void RenderWindow_2plots(SDL_Renderer* renderer,
    const std::vector<double>& new_sol_R_call,
    const std::vector<double>& sol_pde_C_call,
    double xScale, double yScale,int windowHeight);
```

3.4.2 RenderWindow1plot

La fonction `RenderWindow1plot` est utilisée pour tracer un seul graphique dans une fenêtre SDL. Elle prend des entrées similaires à `RenderWindow_2plots` mais nécessite un seul vecteur de données. Cette fonction est principalement utilisée pour visualiser la différence ou l'erreur entre les deux solutions de l'EDP, représentée en cyan.

```
void RenderWindow1plot(SDL_Renderer* renderer,
    const std::vector<double>& diff_call,
    double xScale, double yScale,int windowHeight);
```

Ces fonctions démontrent l'utilisation de SDL pour le rendu de graphiques linéaires simples, où chaque point de données est connecté par des lignes. Les fonctions SDL comme `SDL_SetRenderDrawColor` et `SDL_RenderDrawLine` sont utilisées à des fins de dessin.