École Nationale Supérieure d'Informatique
pour l'Industrie et l'Entreprise

ENSIIE-Évry

Projet PAP 2A 2020-2021

Équation de Black et Scholes

Rathea UTH, Terence NGO

17 janvier 2021

Projet PAP 2A 2020-2021

Rathea UTH, Terence NGO

ENSIIE, Évry

Table des matières

1 Introduction

Dans le monde d'aujourd'hui, la finance joue un rôle des plus importants et est parfois à l'origine de crises mondiales. Il apparaît alors important que la finance soit basée sur des modèles solides permettant d'évaluer les risques et les prix. En mathématique financière, on modélise généralement le prix des actifs de manière continue. En 1973, Fischer Black et Myron Scholes ont dévéloppé un modèle pour calculer le prix d'une option, dite **européenne**, liant le prix de cette option aux variations de l'actif sous-jacent (que l'on peut considérer comme une action). Leur modélisation du prix de l'actif sous-jacent S_t vérifie l'équation différentielle stochastique suivante :

$$\mathrm{d}S_t = \mu S_t \mathrm{d}t + \sigma S_t \mathrm{d}W_t = S_t(\mu \mathrm{d}t + \sigma \mathrm{d}W_t)$$

où σ est la volatilité, μ une dérive, et W_t est un processus de Wiener.

2 Équation aux dérivées partielles (EDP) de Black-Scholes

2.1 EDP complète

En utilisant la formule d'Itô, nous pouvons montrer que la fonction prix du put $C\,(t, S)$ qui dépend du temps t auquel on achète le produit et du prix de l'actif sous-jacent S_t à l'instant t vérifie l'équation aux dérivées partielles (EDP) suivantes définie sur $[0, T] \times [0, L]$:

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

avec T le temps terminal, r le taux d'intérêt du marché et σ à nouveau la volatilité de l'actif. Il est à noter que la condition est terminale, et non initiale, puisqu'on connait la valeur du prix à l'instant T mais pas à l'instant 0. Pour résoudre cette EDP, on donne la condition à l'instant T (la condition initiale) et sur les bords. Les conditions sont alors nécessaires :

$$C(T, s) \qquad \text{condition initiale} \tag{2}$$

$$C(t, 0) \qquad \text{condition au bord} \tag{3}$$

$$C(t, L) \qquad \text{condition au bord} \tag{4}$$

On utilisera 2 types de condition terminale pour $C(T, s)$, nommées **payoff**, dépendant d'un paramètre $K > 0$ appelé le **strike** :

1. Un **put** : $C(T, s) = \max(0, K - s)$ pour $s \in [0, L]$ et dans ce cas :
   - $C(t, 0) = Ke^{-r(T-t)}$ pour tout $t \in [0, T]$
   - $C(t, L) = 0$ pour tout $t \in [0, T]$
2. Un **call** : $C(T, s) = \max(0, s - K)$ pour $s \in [0, L]$ et dans ce cas :
   - $C(t, 0) = 0$ pour tout $t \in [0, T]$
   - $C(t, L) = Ke^{-r(t-T)}$ pour tout $t \in [0, T]$

Nous utiliserons, pour approcher numériquement cette EDP (1), le schéma de Crank-Nicolson (implicite pour $\frac{\partial^2 C}{\partial S^2}$), qui consiste à approcher les dérivées partielles par leurs quotients différentiels, sur une grille de points.

2.2 EDP réduite

On peut aussi effectuer un changement de variable de telle sorte que l'équation (1) devient

$$\frac{\partial \tilde{C}}{\partial \tilde{t}} = \mu \frac{\partial^2 \tilde{C}}{\partial \tilde{s}^2} \tag{5}$$

Nous utiliserons, pour approcher numériquement cette EDP (5), le schéma aux différences finies implicite, qui consiste à approcher les dérivées partielles par leur quotients différentiels, sur une grille de points.

3 Objectif

Notre objectif est de produire un code **C++** adapté à **la méthode de Crank-Nicolson** dans le cas **implicite** et **la méthode des différences finies** dans le cas **implicite** pour résoudre les problèmes (1) et (5), respectivement. Nous allons considérer 2 cas de cette équation aux dérivées partielles (EDP); EDP complète et EDP réduite. À chaque cas, nous allons calculer $C(0, s)$ pour tout $s \in [0, L]$ dans le cas d'un achat (call) et d'une vente (put).

4 Premier cas : EDP complète

Dans ce cas, il nous est demandé d'obtenir la solution approchée en utilisant **la méthode de Crank-Nicholson**.

4.1 Discrétisation de l'espace-temps

Il s'agit, en discrétisant le temps et l'espace, de déterminer une solution approchée de la fonction prix du put C qui vérifie l'EDP (1).

On discrétise donc l'intervalle temporel $[0, T]$ en $M = 1000$ sous-intervalles en posant les notations suivantes:

$$\Delta T = \frac{T}{M} \quad et \quad t_m = m\Delta T \quad \forall m \in \{0, ..., M\}$$

On fait de même avec l'intervalle spatial $[0, L]$ que l'on scinde en $N = 1000$ sous-intervalles:

$$\Delta s = \frac{L}{N} \quad et \quad s_i = i\Delta s \quad \forall i \in \{0, ..., N\}$$

4.2 Discrétisation des dérivées partielles

Pour tout $(m, i) \in \{0, \ldots, M-1\} \times \{1, \ldots, N-1\}$, nous approchons les dérivées partielles de l'EDP dans un **schéma de Crank-Nicholson** comme ci-dessous:

$$\begin{aligned}
&\frac{\partial C}{\partial t}(t_m, s_i) \approx \frac{1}{\triangle T}\left(C(t_{m+1}, s_i) - C(t_m, s_i)\right) \\
&\frac{\partial C}{\partial S}(t_m, s_i) \approx \frac{1}{2}\left(\frac{C(t_{m+1}, s_{i+1}) - C(t_{m+1}, s_{i-1})}{2\Delta s} + \frac{C(t_m, s_{i+1}) - C(t_m, s_{i-1})}{2\Delta s}\right) \\
&\frac{\partial^2 C}{\partial S^2}(t_m, s_i) \approx \frac{1}{2\Delta s^2}\left[\left(C(t_{m+1}, s_{i+1}) - 2C(t_{m+1}, s_i) + C(t_{m+1}, s_{i-1})\right) + \right. \\
&\qquad \left. + \left(C(t_m, s_{i+1}) - 2C(t_m, s_i) + C(t_m, s_{i-1})\right)\right]
\end{aligned}$$

Ainsi, en notant $C_m^i = C(t_m, s_i)$, on obtient alors :

$$\begin{array}{l}\frac{\partial C_m^i}{\partial t} \approx \frac{1}{\triangle T}\left(C_{m+1}^i - C_m^i\right)\\ \frac{\partial C_m^i}{\partial S} \approx \frac{1}{2}\left(\frac{C_{m+1}^{i+1} - C_{m+1}^{i-1}}{2\Delta s} + \frac{C_m^{i+1} - C_m^{i-1}}{2\Delta s}\right)\\ \frac{\partial^2 C_m^i}{\partial S^2} \approx \frac{1}{2\Delta s^2}\left[\left(C_{m+1}^{i+1} - 2C_{m+1}^i + C_{m+1}^{i-1}\right) + \left(C_m^{i+1} - 2C_m^i + C_m^{i-1}\right)\right]\end{array}$$

4.3 Résolution de l'équation pour un put

Pour l'option de vente (le put), on a la condition initiale $C(T, s) = \max(0, K - s)$ $\forall s \in [0, L]$ avec les conditions aux bords:

$$\begin{array}{l}C(t, 0) = Ke^{-r(T-t)} \quad \forall t \in [0, T]\\ C(t, L) = 0 \quad \forall t \in [0, T]\end{array}$$

En utilisant les approximations des dérivées partielles de l'EDP dans un **schéma de Crank-Nicholson**, l'équation (1) devient:

$$\begin{array}{l}\frac{1}{\Delta T}\left(C_{m+1}^i - C_m^i\right) + \frac{rS_i}{2}\left(\frac{C_{m+1}^{i+1} - C_{m+1}^{i-1}}{2\Delta s} + \frac{C_m^{i+1} - C_m^{i-1}}{2\Delta s}\right) + \frac{\sigma^2 S_i^2}{4\Delta s^2}\left(C_{m+1}^{i+1} - 2C_{m+1}^i + C_{m+1}^{i-1}\right)\\ + \frac{\sigma^2 S_i^2}{4\Delta s^2}\left(C_m^{i+1} - 2C_m^i + C_m^{i-1}\right) = \frac{r}{2}C_m^i + \frac{r}{2}C_{m+1}^i\end{array} \tag{6}$$

En utilisant la discrétisation de l'espace avec $s_i = i\Delta s$, on trouve alors :

$$\begin{array}{l}\left(-\frac{ri}{4} + \frac{\sigma^2 i^2}{4}\right)C_m^{i-1} - \left(\frac{1}{\Delta T} + \frac{r}{2} + \frac{\sigma^2 i^2}{2}\right)C_m^i + \left(\frac{ri}{4} + \frac{\sigma^2 i^2}{4}\right)C_m^{i+1}\\ = -\left(-\frac{ri}{4} + \frac{\sigma^2 i^2}{4}\right)C_{m+1}^{i-1} + \left(-\frac{1}{\Delta T} + \frac{r}{2} + \frac{\sigma^2 i^2}{2}\right)C_{m+1}^i - \left(\frac{ri}{4} + \frac{\sigma^2 i^2}{4}\right)C_{m+1}^{i+1}\end{array}$$

Pour $i = 1$:

$$\begin{array}{l}\left(-\frac{r}{4} + \frac{\sigma^2}{4}\right)C_m^0 - \left(\frac{1}{\Delta T} + \frac{r}{2} + \frac{\sigma^2}{2}\right)C_m^1 + \left(\frac{r}{4} + \frac{\sigma^2}{4}\right)C_m^2\\ = -\left(-\frac{r}{4} + \frac{\sigma^2}{4}\right)C_{m+1}^0 + \left(-\frac{1}{\Delta T} + \frac{r}{2} + \frac{\sigma^2}{2}\right)C_{m+1}^1 - \left(\frac{r}{4} + \frac{\sigma^2}{4}\right)C_{m+1}^2\end{array}$$

Sachant que pour un put, $C_m^0 = Ke^{-r(T-t_m)}$ pour tout m, il vient alors :

$$\begin{array}{l}\left(-\frac{r}{4} + \frac{\sigma^2}{4}\right)Ke^{-r(T-t_m)} - \left(\frac{1}{\Delta T} + \frac{r}{2} + \frac{\sigma^2}{2}\right)C_m^1 + \left(\frac{r}{4} + \frac{\sigma^2}{4}\right)C_m^2\\ = -\left(-\frac{r}{4} + \frac{\sigma^2}{4}\right)Ke^{-r(T-t_{m+1})} + \left(-\frac{1}{\Delta T} + \frac{r}{2} + \frac{\sigma^2}{2}\right)C_{m+1}^1 - \left(\frac{r}{4} + \frac{\sigma^2}{4}\right)C_{m+1}^2\end{array}$$

Cette formule motive l'introduction de la suite $(u_m)_m$ un peu plus loin.

Pour $i = N - 1$:

$$\begin{array}{l}\left(-\frac{r(N-1)}{4} + \frac{\sigma^2(N-1)^2}{4}\right)C_m^{N-2} - \left(\frac{1}{\Delta T} + \frac{r}{2} + \frac{\sigma^2(N-1)^2}{2}\right)C_m^{N-1} + \left(\frac{r(N-1)}{4} + \frac{\sigma^2(N-1)^2}{4}\right)C_m^N\\ = -\left(-\frac{r(N-1)}{4} + \frac{\sigma^2(N-1)^2}{4}\right)C_{m+1}^{N-2} + \left(-\frac{1}{\Delta T} + \frac{r}{2} + \frac{\sigma^2(N-1)^2}{2}\right)C_{m+1}^{N-1} - \left(\frac{r(N-1)}{4} + \frac{\sigma^2(N-1)^2}{4}\right)C_{m+1}^N\end{array}$$

Comme $C_m^N = C(t_m, L) = 0$ pour tout m, alors on a :

$$\left(-\frac{r(N-1)}{4} + \frac{\sigma^2(N-1)^2}{4}\right) C_m^{N-2} - \left(\frac{1}{\Delta T} + \frac{r}{2} + \frac{\sigma^2(N-1)^2}{2}\right) C_m^{N-1}$$

$$= -\left(-\frac{r(N-1)}{4} + \frac{\sigma^2(N-1)^2}{4}\right) C_{m+1}^{N-2} + \left(-\frac{1}{\Delta T} + \frac{r}{2} + \frac{\sigma^2(N-1)^2}{2}\right) C_{m+1}^{N-1}$$

Soient:

$$a_i = -\frac{ri}{4} + \frac{\sigma^2 i^2}{4}, b_i = \frac{1}{\Delta T} + \frac{r}{2} + \frac{\sigma^2 i^2}{2}$$
$$c_i = \frac{ri}{4} + \frac{\sigma^2 i^2}{4}, d_i = -\frac{1}{\Delta T} + \frac{r}{2} + \frac{\sigma^2 i^2}{2}$$

L'équation matricielle devient donc :

$$u_m + AC_m = -u_{m+1} + BC_{m+1}$$

où

$$A = \begin{pmatrix} -b_1 & c_1 & 0 & \dots & \dots & 0 & 0 \\ a_2 & -b_2 & c_2 & 0 & \dots & 0 & 0 \\ 0 & \ddots & \ddots & \ddots & \ddots & & \vdots \\ \vdots & \ddots & a_i & -b_i & c_i & \ddots & \vdots \\ \vdots & & \ddots & \ddots & \ddots & \ddots & 0 \\ 0 & 0 & \dots & \ddots & \ddots & \ddots & c_{N-2} \\ 0 & 0 & \dots & \dots & 0 & a_{N-1} & -b_{N-1} \end{pmatrix}$$

$$B = \begin{pmatrix} d_1 & -c_1 & 0 & \dots & \dots & 0 & 0 \\ -a_2 & d_2 & -c_2 & 0 & \dots & 0 & 0 \\ 0 & \ddots & \ddots & \ddots & \ddots & & \vdots \\ \vdots & \ddots & -a_i & d_i & -c_i & \ddots & \vdots \\ \vdots & & \ddots & \ddots & \ddots & \ddots & 0 \\ 0 & 0 & \dots & \ddots & \ddots & \ddots & -c_{N-2} \\ 0 & 0 & \dots & \dots & 0 & -a_{N-1} & d_{N-1} \end{pmatrix}$$

$$u_m = \begin{pmatrix} \left(\frac{\sigma^2}{4} - \frac{r}{4}\right) K e^{-r(T-t_m)} \\ 0 \\ 0 \\ \vdots \\ \vdots \\ 0 \\ 0 \end{pmatrix}$$

et enfin

$$C_m = \begin{pmatrix} C_m^1 \\ C_m^2 \\ \vdots \\ \vdots \\ C_m^{N-1} \end{pmatrix}$$

Remarque:
Puisque $C_m^0 = Ke^{-r(T-t_m)}$ *et* $C_m^N = 0$ *sont également connus pour tout* m*, il est redondant de les inclure dans le vecteur* C_m*.*

Finalement:

$$\boxed{AC_m = -(u_m + u_{m+1}) + BC_{m+1} \qquad \forall m \in \{0, \dots, M-1\}} \tag{7}$$

En remarquant que A est une matrice (tridiagonale) dont les mineurs sont tous non nuls (ce que nous avons vérifié numériquement en déterminant leurs valeurs avec les paramètres $K, r, \sigma...$ fixés par l'énoncé), il est possible de la décomposer selon la factorisation LU. Par ailleurs, étant donné que $C_M = \max(0, K - s_i)_{1 \le i \le N-1}$ est connu, il est possible d'atteindre successivement $C_{M-1}, \dots, C_0$ par itération.

4.4 Résolution de l'équation pour un call

Pour ce qui est des options call:

$$\begin{aligned} &C_T^s = C(T, s) = \max(0, s - K) \quad \forall s \in [0, L] \\ &C_t^N = C(t, L) = Ke^{-r(t-T)} \quad \forall t \in [0, T] \\ &C_t^0 = C(t, 0) = 0 \quad \forall t \in [0, T] \end{aligned}$$

Les seuls changements sont ces 3 conditions, qui se traduisent par les manières suivantes au niveau de la relation de récurrence matricielle:

- Comment $C_M = (C(T, s_i))_{1 \le i \le N-1}$ est initialisée. Dans le cas d'une option call, $C_M = \max(0, s_i - K)_{1 \le i \le N-1}$.
- Comment la suite $(u_m)_m$ est définie. En effet, réécrivons la relation de récurrence lorsque $i = 1$ et $i = N - 1$:

  Pour $i = 1$:

$$\left(-\frac{r}{4} + \frac{\sigma^2}{4}\right) C_m^0 - \left(\frac{1}{\Delta T} + \frac{r}{2} + \frac{\sigma^2}{2}\right) C_m^1 + \left(\frac{r}{4} + \frac{\sigma^2}{4}\right) C_m^2$$

$$= -\left(-\frac{r}{4} + \frac{\sigma^2}{4}\right) C_{m+1}^0 + \left(-\frac{1}{\Delta T} + \frac{r}{2} + \frac{\sigma^2}{2}\right) C_{m+1}^1 - \left(\frac{r}{4} + \frac{\sigma^2}{4}\right) C_{m+1}^2$$

  Sachant que pour un call, $C_m^0 = 0$ pour tout m, il vient alors :

$$\overset{call}{\Longleftrightarrow} -\left(\frac{1}{\Delta T} + \frac{r}{2} + \frac{\sigma^2}{2}\right) C_m^1 + \left(\frac{r}{4} + \frac{\sigma^2}{4}\right) C_m^2$$

$$= \left(-\frac{1}{\Delta T} + \frac{r}{2} + \frac{\sigma^2}{2}\right) C_{m+1}^1 - \left(\frac{r}{4} + \frac{\sigma^2}{4}\right) C_{m+1}^2$$

Pour $i = N-1$:

$$\left(-\tfrac{r(N-1)}{4} + \tfrac{\sigma^2(N-1)^2}{4}\right) C_m^{N-2} - \left(\tfrac{1}{\Delta T} + \tfrac{r}{2} + \tfrac{\sigma^2(N-1)^2}{2}\right) C_m^{N-1} + \left(\tfrac{r(N-1)}{4} + \tfrac{\sigma^2(N-1)^2}{4}\right) C_m^N$$

$$= -\left(-\tfrac{r(N-1)}{4} + \tfrac{\sigma^2(N-1)^2}{4}\right) C_{m+1}^{N-2} + \left(-\tfrac{1}{\Delta T} + \tfrac{r}{2} + \tfrac{\sigma^2(N-1)^2}{2}\right) C_{m+1}^{N-1} - \left(\tfrac{r(N-1)}{4} + \tfrac{\sigma^2(N-1)^2}{4}\right) C_{m+1}^N$$

Par suite, pour les options calls, on a $C_m^N = Ke^{-r(t_m - T)}$. On trouve alors :

$$\left(\frac{r(N-1)}{4} + \frac{\sigma^2(N-1)^2}{4}\right) C_m^N = \left(\frac{r(N-1)}{4} + \frac{\sigma^2(N-1)^2}{4}\right) Ke^{-r(t_m - T)}$$

Et :

$$-\left(\frac{r(N-1)}{4} + \frac{\sigma^2(N-1)^2}{4}\right) C_{m+1}^N = -\left(\frac{r(N-1)}{4} + \frac{\sigma^2(N-1)^2}{4}\right) Ke^{-r(t_{m+1} - T)}$$

En d'autres termes, $(u_m)_m$ devient:

$$u_m = \begin{pmatrix} 0 \\ 0 \\ \vdots \\ \vdots \\ 0 \\ 0 \\ \left(\frac{r(N-1)}{4} + \frac{\sigma^2(N-1)^2}{4}\right) Ke^{-r(t_m - T)} \end{pmatrix}$$

- Par conséquent:

$$\boxed{AC_m = -(u_m + u_{m+1}) + BC_{m+1} \qquad \forall m \in \{0, \ldots, M-1\}}$$

Une fois de plus, on peut utiliser la décomposition LU de A pour simplifier la résolution du système à chaque étape.

4.5 Algorithme et solution

Afin de résoudre (7), nous devons trouver l'inverse de matrix A qui est de taille $(N-1)$. Cela est trop coûteux, c'est pourquoi nous utilisons l' **Algorithme de Crout** qui est un outil très puissant, utilisé pour résoudre le système de la forme $Ax = b$.

Algorithme de Crout

Cet algorithme fonctionne dans les deux étapes importantes suivantes :

1) Décomposition de la matrice $A = LU$ où L et U sont respectivement les matrices triangulaire inférieure et supérieure.

2) Résolution de $Ax = b$ en résolvant $Ly = b$ pour y puis résolvez $Ux = y$ pour x qui sont plus faciles à résoudre que le système original.

Les tracés obtenus avec la méthode de **Crank-Nicholson** implicite pour résoudre (1) dans le cas d'une option call et put sont les suivants :

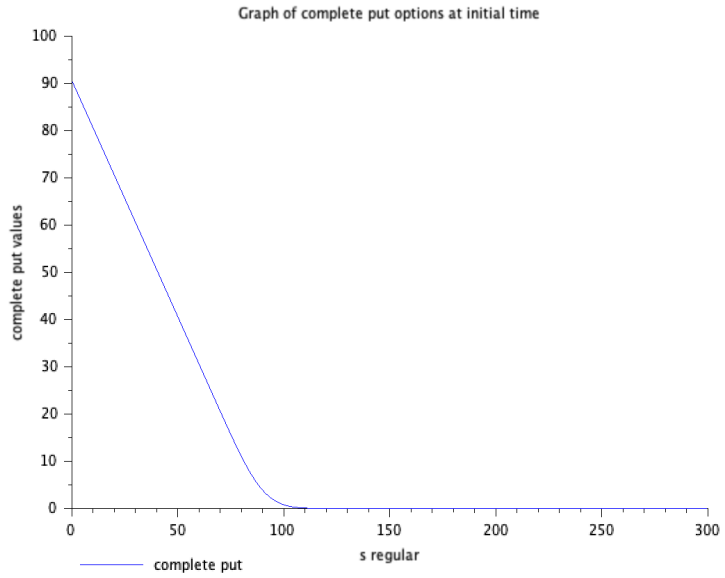

Figure 1: Graphe d'un put solution de (1), avec discrétisation régulière des s

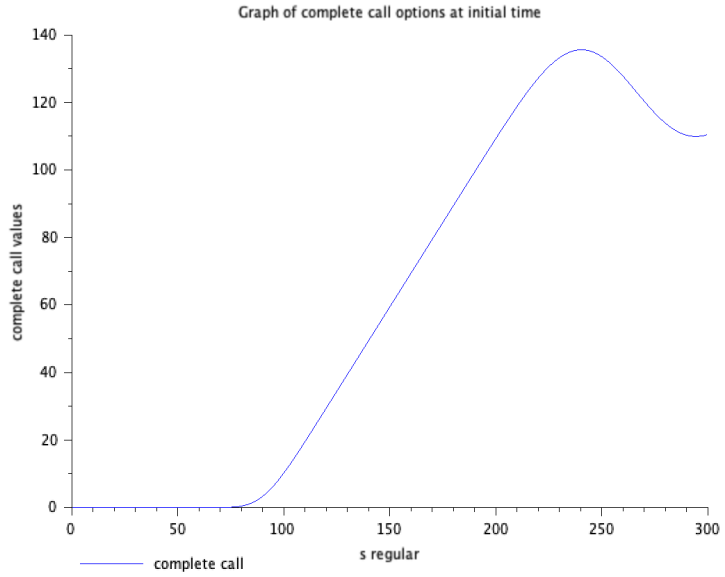

Figure 2: Graphe d'un call solution de (1), avec discrétisation régulière des s

Le graphe d'une option put ne commence pas de 100 qui est la valeur de $C_m^0 = Ke^{-r(T-t_m)}$ quand $t_m = T$ mais de $C_m^1 = 90.1928$ puisqu'on n'a pas inclus le premier point $s = 0$ dans notre vecteur solution C_m.

Dans le cas d'un call, comme $C_m^0 = 0$, cela ne se voit pas sur le graphe.

5 Second cas: EDP réduite

5.1 Passage de l'équation complète à la version réduite

Il est également possible d'obtenir l'EDP (5) :

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

à partir de (1) en effectuant les changements de variables suivantes:

- $\tilde{t} = T - t$
- $\tilde{s} = log\left(\frac{s}{K}\right) + \left(r - \frac{\sigma^2}{2}\right) \times \tilde{t}$
- $\tilde{C}(\tilde{t}_{|t}, \tilde{s}_{|s}) = e^{r\tilde{t}} \times C(t, s)$
- $\mu = \frac{1}{2}\sigma^2$

Remarque:
Le changement de variable de s pose problème lorsque $s = 0$ à cause du logarithme (on a alors $\tilde{s} = -\infty$), mais nous pouvons toutefois calculer $\tilde{C}(\tilde{t}, -\infty)$ via la relation entre $\tilde{C}(\tilde{t}, -\infty)$ et $C(t, 0)$ qui lui, est bien défini.

En fait, $\tilde{s}$ prend désormais ses valeurs dans $\mathbb{R}$ entier.
Lorsque nous procéderons à la discrétisation de l'intervalle spatial, nous isolerons le cas $s_0 = 0$ des autres cas $s_i \neq 0$ ne posant pas de problème de définition de $\tilde{s}$.

5.2 Résolution de l'équation pour un call

5.2.1 Discrétisation des intervalles

Comme annoncé, nous allons utiliser la méthode des différences finies pour approximer (5). Cette méthode nécessite de discrétiser les intervalles en jeu avec des pas constants.
Ainsi, on découpe l'intervalle $[0, T]$ dans lequel se trouve $\tilde{t}$ en M segments réguliers.

Pour $\tilde{s}$ cependant, comme précisé ci-dessus, nous allons isoler le cas $s = 0$, de telle sorte à découper l'intervalle $[\tilde{s}_{|s_1}, \tilde{s}_{|s_N}] = [\log\left(\frac{L}{NK}\right), \log\left(\frac{L}{K}\right)]$ en $N - 1$ pas réguliers.

***Remarque*:**
Ce choix risque de poser problème si l'on veut par la suite afficher la courbe d'erreur entre les solutions de (1) et (5), à cause de la discordance entre les points s résultants.

Ainsi:

- Discrétisation des $\tilde{t}$:
  Posons $\Delta\tilde{t} = \Delta T = \frac{T}{M}$ et $\forall m \in \{0, ..., M\} \quad \tilde{t}_m = m\Delta T$.

  Remarque:
  Attention, $\tilde{t}_i$ va croissant tandis qu'au contraire, $\tilde{t}$ décroît, si bien que:
  $\tilde{t}_m = m\Delta T = T - t \iff t = (M - m)\Delta T \iff t = t_{M-m}$
  Par conséquent, le point t qui est "transformé" en $\tilde{t}_m$ n'est pas t_m mais t_{M-m}.

- Discrétisation des $\tilde{s}$:
  Posons $\Delta\tilde{S} = \frac{1}{N-1}\left(\log\left(\frac{L}{K}\right) - \log\left(\frac{s_1}{K}\right)\right) = \frac{1}{N-1}\log(N)$
  et:

$$\forall i \in \{1,..,N\} \quad \tilde{s}_i = \underbrace{\log\left(\frac{s_1}{K}\right) + \left(r - \frac{\sigma^2}{2}\right)T}_{=\tilde{s}_1} + (i-1) \times \Delta\tilde{S}$$

5.2.2 Relation de récurrence

Comme pour l'EDP complète, notons $\tilde{C}_m^i = \tilde{C}(\tilde{t}_m, \tilde{s}_i)$.
Nous pouvons maintenant procéder à l'approximation des dérivées partielles :

- Comme pour l'EDP complète :

$$\frac{\partial \tilde{C}_m^i}{\partial \tilde{t}} = \frac{1}{\Delta T}\left(\tilde{C}_{m+1}^i - \tilde{C}_m^i\right)$$

  car le découpage est identique.

- C'est plus compliqué pour la dérivée partielle spatiale d'ordre 2. Rigoureusement :

$$\forall i \in \{1,...,N-1\}, \forall m \in \{0,...,M\} \qquad \frac{\partial^2 \tilde{C}_m^i}{\partial \tilde{s}^2} = \frac{1}{\Delta\tilde{S}}\left(\frac{\tilde{C}_{m+1}^{i+1} - \tilde{C}_{m+1}^i}{\Delta\tilde{S}} - \frac{\tilde{C}_{m+1}^i - \tilde{C}_{m+1}^{i-1}}{\Delta\tilde{S}}\right)$$

- L'approximation précédente se réécrit:

$$\frac{\partial^2 \tilde{C}_m^i}{\partial \tilde{s}^2} = \frac{\tilde{C}_{m+1}^{i+1} - 2\tilde{C}_{m+1}^i + \tilde{C}_{m+1}^{i-1}}{(\Delta\tilde{S})^2}$$

Il découle des 2 points précédents injectés dans l'équation (5), que:

$$\forall i \in \{1,...,N-1\}, \forall m \in \{0,...,M\} \qquad \frac{1}{\Delta T}\left(\tilde{C}_{m+1}^i - \tilde{C}_m^i\right) = \frac{\sigma^2}{2}\frac{\tilde{C}_{m+1}^{i+1} - 2\tilde{C}_{m+1}^i + \tilde{C}_{m+1}^{i-1}}{(\Delta\tilde{S})^2}$$

Soit:

$$\forall i \in \{1,...,N-1\}, \forall m \in \{0,...,M\} \qquad -\frac{\sigma^2\Delta T}{2(\Delta\tilde{S})^2}\tilde{C}_{m+1}^{i-1} + \left(1 + \frac{\sigma^2\Delta T}{(\Delta\tilde{S})^2}\right)\tilde{C}_{m+1}^i - \frac{\sigma^2\Delta T}{2(\Delta\tilde{S})^2}\tilde{C}_{m+1}^{i+1} = \tilde{C}_m^i$$

En posant

$$\mu = -\frac{\sigma^2\Delta T}{2(\Delta\tilde{S})^2} \quad \text{et} \quad \lambda = 1 + \frac{\sigma^2\Delta T}{(\Delta\tilde{S})^2}$$

on obtient alors :

$$\forall i \in \{1,...,N-1\}, \forall m \in \{0,...,M\} \qquad \mu\tilde{C}_{m+1}^{i-1} + \lambda\tilde{C}_{m+1}^i + \mu\tilde{C}_{m+1}^{i+1} = \tilde{C}_m^i$$

Afin de pouvoir passer à la version matricielle, il convient de traiter les habituels cas particuliers $i = 1$ et $i = N-1$:

- Pour $i = 1$:

$$\mu\tilde{C}^0_{m+1} + \lambda\tilde{C}^1_{m+1} + \mu\tilde{C}^2_{m+1} = \tilde{C}^1_m$$

Or, pour un call: $\tilde{C}^0_m = e^{r\tilde{t}_m}C(t_{M-m}, 0) \overset{call}{=} 0$. Donc:

$$\lambda\tilde{C}^1_{m+1} + \mu\tilde{C}^2_{m+1} = \tilde{C}^1_m$$

- Pour $i = N - 1$:

$$\mu\tilde{C}^{N-2}_{m+1} + \lambda\tilde{C}^{N-1}_{m+1} + \mu\tilde{C}^N_{m+1} = \tilde{C}^{N-1}_m$$

La condition au bord de l'intervalle spatial d'une option call nous permet de même d'en déduire:

$$\tilde{C}^N_m = e^{r\tilde{t}_m}C(t_{M-m}, L) \overset{call}{=} e^{r\tilde{t}_m} \times Ke^{-r(t_{M-m}-T)} = Ke^{2rm\Delta T} = Ke^{2r\tilde{t}_m}$$

D'où:

$$\mu\tilde{C}^{N-2}_{m+1} + \lambda\tilde{C}^{N-1}_{m+1} + \mu Ke^{2r(m+1)\Delta T} = \tilde{C}^{N-1}_m$$

Par suite, voici la version matricielle:

$$\tilde{A}\tilde{C}_{m+1} + \tilde{u}_{m+1} = \tilde{C}_m$$

avec:

$$\tilde{A} = \begin{pmatrix} \lambda & \mu & 0 & \dots & \dots & 0 & 0 \\ \mu & \lambda & \mu & 0 & \dots & 0 & 0 \\ 0 & \ddots & \ddots & \ddots & \ddots & & \vdots \\ \vdots & \ddots & \mu & \lambda & \mu & \ddots & \vdots \\ \vdots & & \ddots & \ddots & \ddots & \ddots & 0 \\ 0 & 0 & \dots & \ddots & \ddots & \ddots & \mu \\ 0 & 0 & \dots & \dots & 0 & \mu & \lambda \end{pmatrix}$$

$$\tilde{C}_m = \begin{pmatrix} \tilde{C}^1_m \\ \tilde{C}^2_m \\ \vdots \\ \vdots \\ \tilde{C}^{N-1}_m \end{pmatrix}$$

et enfin:

$$\tilde{u}_m = \begin{pmatrix} 0 \\ 0 \\ \vdots \\ \vdots \\ 0 \\ 0 \\ \mu Ke^{2rm\Delta T} \end{pmatrix}$$

Finalement:

$$\boxed{\tilde{A}\tilde{C}_{m+1} = -\tilde{u}_{m+1} + \tilde{C}_m \qquad \forall m \in \{0, \dots, M-1\}} \tag{8}$$

avec une fois de plus $\tilde{A}$ qui est une matrice tridiagonale admettant une décomposition LU (le calcul numérique des mineurs montre qu'ils sont tous non nuls) au même titre que A dans l'équation (1).

Étant donné le changement de variable effectué, la condition temporelle n'est plus terminale mais initiale:

$$t = T = t_M = t_{M-0} \implies \tilde{t} = \tilde{t}_0 = 0$$

Dans ce cas, nous avons la connaissance de $\tilde{C}_0$. Puisque $\tilde{t} = 0$, alors:

$$\tilde{s} = \log\left(\frac{s}{K}\right) \iff s = Ke^{\tilde{s}}$$

Ainsi:

$$\forall i \in \{1, ..., N-1\} \quad \tilde{C}_0^i = e^{r0} C(T, Ke^{\tilde{s}_i}) \overset{call}{=} \max\{0, Ke^{\tilde{s}_i} - K\} = K\left(e^{\max\{\tilde{s}_i, 0\}} - 1\right)$$

et notre objectif est de déterminer $\tilde{C}_M$ (contient les valeurs du call "transformé" à l'instant initial $t = t_0 = 0$).

5.3 Résolution de l'équation pour un put

La discrétisation des intervalles de $\tilde{t}$ et $\tilde{s}$ est identique au cas du call.

Seule la relation de récurrence est légèrement modifiée par les conditions terminales et aux bords pour les puts:

- Le vecteur initial $\tilde{C}_0$ duquel on part est désormais tel que:

$$\forall i \in \{1, ..., N-1\} \quad \tilde{C}_0^i = e^{r0} C(T, e^{\tilde{s}_i}) \overset{put}{=} \max\{0, K - Ke^{\tilde{s}_i}\} = K\left(1 - e^{\min\{\tilde{s}_i, 0\}}\right)$$

- La suite $\tilde{u}_m$ s'en trouve également modifiée:

$$\forall m \in \{0, ..., M\}$$

$$\tilde{u}_m = \begin{pmatrix} \mu K \\ 0 \\ 0 \\ \vdots \\ \vdots \\ 0 \\ 0 \end{pmatrix}$$

puisque:

$$\tilde{C}_m^N = e^{r\tilde{t}_m} C(t_{M-m}, L) \overset{put}{=} 0$$

$$\text{et} \qquad \tilde{C}_m^0 = e^{r\tilde{t}_m} C(t_{M-m}, 0) \overset{put}{=} e^{rm\Delta T} K e^{-r(T-(M-m)\Delta T)} = K$$

Par conséquent:

$$\boxed{\tilde{A}\tilde{C}_{m+1} = -\tilde{u}_{m+1} + \tilde{C}_m \qquad \forall m \in \{0, \ldots, M-1\}}$$

Une fois de plus, on peut utiliser la décomposition LU de $\tilde{A}$ pour simplifier la résolution du système à chaque étape, puis finalement repasser aux s via la relation de changement de variable $s/\tilde{s}$.

5.4 Algorithme et solution

Afin de résoudre (8), une fois de plus, nous devons trouver l'inverse de la matrice $\tilde{A}$ qui est de taille $(N-1)$. Cela est trop coûteux, c'est pourquoi nous utilisons l' **Algorithme de Crout** qui est un outil très puissant, utilisé pour résoudre le système de la forme $Ax = b$.

Algorithme de Crout

Cet algorithme fonctionne dans les deux étapes importantes suivantes :

1) Décomposition de la matrice $A = LU$ où L et U sont respectivement les matrices triangulaire inférieure et supérieure.

2) Résolution de $Ax = b$ en résolvant $Ly = b$ pour y puis résolvez $Ux = y$ pour x qui sont plus faciles à résoudre que le système original.

Dans le cas du EDP réduite, on se ramène finalement à nouveau à s, en utilisant la relation: $s = K \log\left(\frac{\tilde{s}}{K}\right) + \left(r - \frac{\sigma^2}{2}\right) \times T$ à l'instant initial $t = 0$.

Cependant, l'implémentation rigoureuse en C++ ne mène pas à une solution satisfaisante:

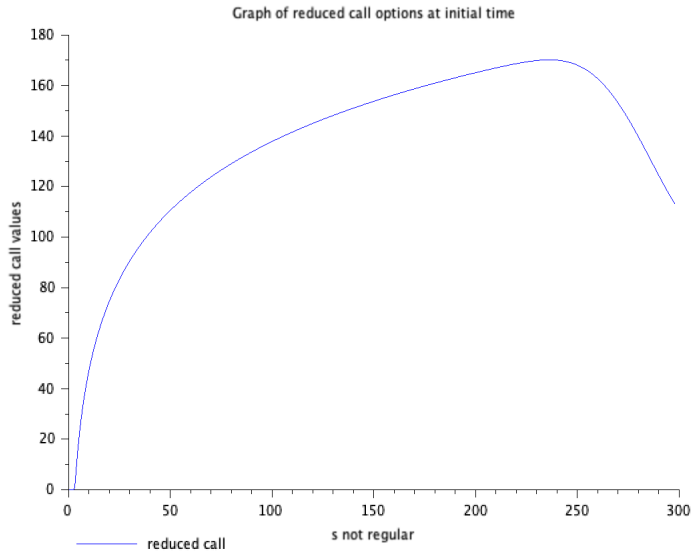

Figure 3: Graphe d'un call solution de (5), avec discrétisation exponentielle

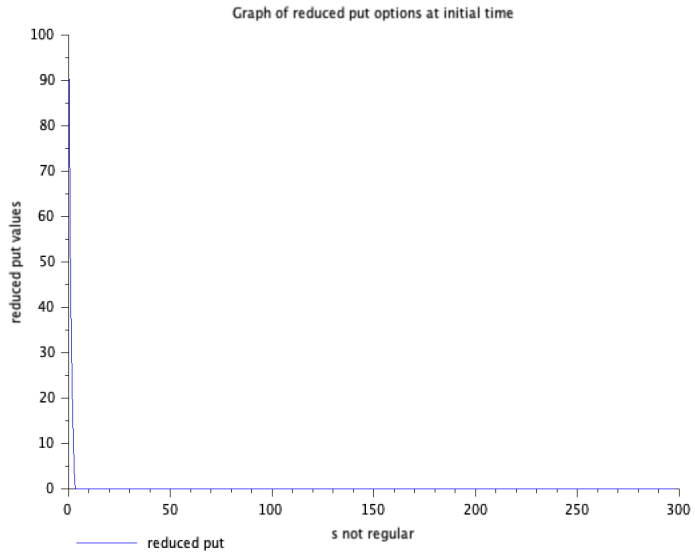

Figure 4: Graphe d'un put solution de (5), avec discrétisation exponentielle

En effet, du fait du passage des $\tilde{s}$ aux s, l'exponentielle "compresse" les premiers s, si bien que les résultats obtenus sont des courbes qui, très rapidement, prennent des valeurs de l'ordre de la centaine, ou au contraire, s'annulent très vite.

C'est ce qu'on peut observer sur le graphe suivant, avec une densité importante des points verts (pas non constant) autour de 0:

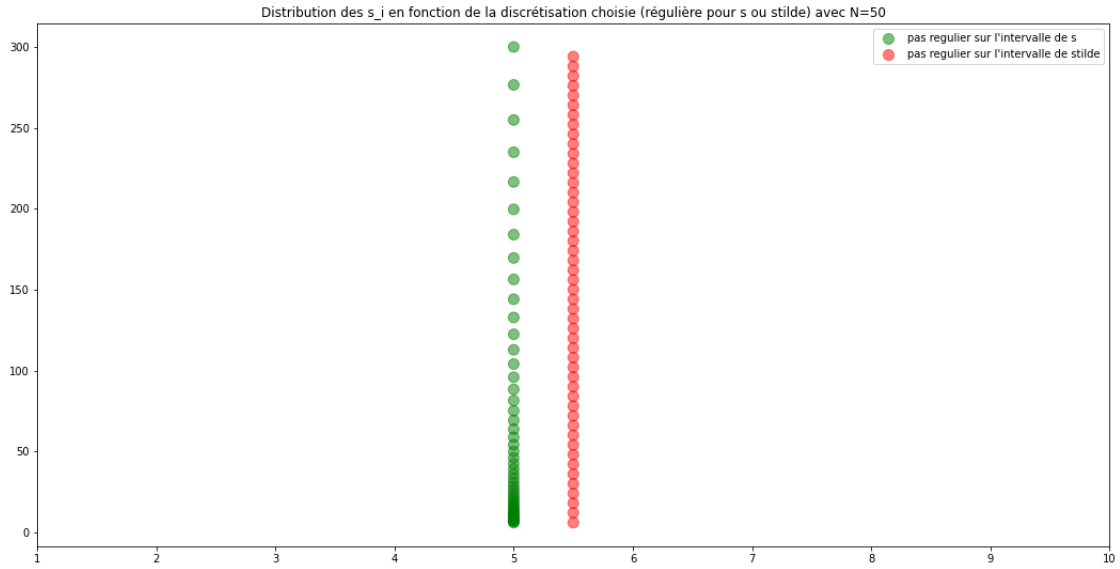

Figure 5: Comparaison des discrétisations de l'intervalle des s pour (1)(rouge) et (5) (vert)

En revanche, en retournant aux s d'origine avec une discrétisation à pas constant $s_i = i\Delta S$, on obtient des résultats bien plus proches de ceux attendus:

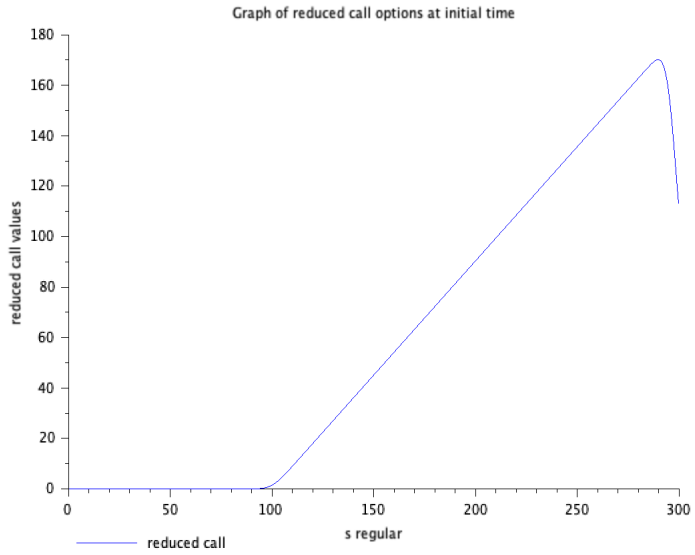

Figure 6: Graphe d'un call solution de (5), avec discrétisation régulière des s

Cette fois-ci, le call augmente bien à partir de 100 environ, comme c'était déjà le cas avec le call obtenu pour (1).

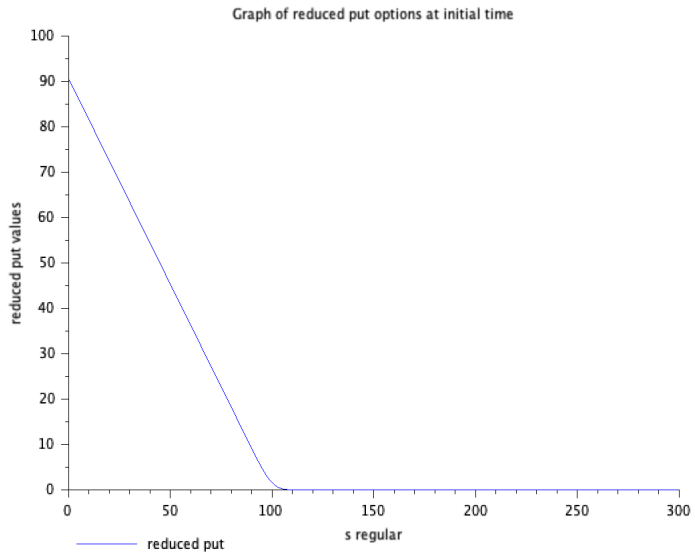

Figure 7: Graphe d'un put solution de (5), avec discrétisation régulière des s

De même, le put s'annule bien à partir de $s \approx 100$, comme c'était déjà le cas pour le put obtenu pour (5).

Bien que nous pensons qu'il y a une erreur de conversion des $\tilde{s}$ vers les s, nous n'en avons malheureusement pas trouvé la raison.
Pensant que les courbes obtenues avec la discrétisation régulière des s sont plus probables d'être correctes, nous avons continué avec celles-ci dans la section suivante:

6 Implémentation en C++

6.1 Problèmes techniques et décisions à prendre

- Il était possible de gérer l'héritage de la classe `Payoff` de plusieurs manières, par exemple:

  1. Définir jusqu'à 4 classes filles différentes `Put_complete`, `Put_reduced`, `Call_complete` et `Call_reduced`.
  2. Faire appel à des **namespaces** "`Reduced`" et "`Complete`" au sein même de 2 classes `Put` et `Call` afin de distinguer les fonctions implémentant les solutions de (1) de celles de (5).

- De même, la nécessité de résoudre des systèmes matriciels nous a conduit à implémenter une classe **Matrix**. Les dilemnes rencontrés furent les suivants:

  1. Comment gérer la décomposition LU, qui retourne un couple de matrices L et U
  2. Certaines méthodes s'appliquant à des vecteurs colonnes (ie des matrices à une colonne) sont nécessaires, telle que l'affichage de leur contenu. Nous nous sommes demandé s'il était plus judicieux de les inclure dans un fichier autre que `Matrix.cpp`

- Pour ce qui est de la résolution des systèmes matriciels, il nous est venu l'idée de définir une classe `Linear_System` à part entière, un peu à la manière de la classe Newton implémentant la méthode de Newton-Raphson en cours.

- Nous avons également eu des problèmes avec la fonction log de C++, qui retournait constamment des "nan" lors de l'implémentation des solution de (5).

6.2 Solutions trouvées

- On a finalement choisi d'utiliser des namespaces dans les classes `Put` et `Call`. En revanche, la classe `Payoff` implémente quelques méthodes qui permettent à la fois d'obtenir les solutions de (1) (comme `get_A()` et `delta_T()`) et de (5) (tel que `get_C()` et `delta_stilde()`).

- Pour ce qui est de la décomposition LU, nous n'avons pas implémenté de méthodes retournant explicitement les matrices triangulaires de la factorisation.

  Les décompositions ne servant que lors de la résolution des sytèmes linéaires, nous avons directement inclus les décompositions dans une fonction **solve** qui retourne le vecteur solution d'un système.

  En parallèle, pour effectuer des tests, nous avons implémenté une méthode **is_lu_decomp** permettant de savoir si la matrice courante admet une décomposition LU ou non.

- Nous n'avons finalement pas implémenté de classe `Linear_System`, mais seulement la fonction **solve** mentionnée plus haut, qui prend en paramètres une matrice M et un vecteur b et qui retourne le vecteur x (s'il existe) solution de $Mx = b$.

- Les choix effectués ont été résumés dans le diagramme UML ci-dessous. Pour ne pas surcharger le diagramme, les descriptions des méthodes ont été placées juste avant texte informatif suivant:

Description des méthodes

- **Classe Payoff:**
  - **Payoff(*double K, double r, double sigma, double T, double L, int N, int M*)** et **Payoff(*const Payoff P*)**: respectivement les constructeurs de copie et avec paramètres)
  - ***double delta_T()***: retourne L/N le pas temporel utilisé dans le schéma numérique de Cranck Nicholson pour résoudre (1)
  - ***double delta_stilde() const***: retourne $\frac{1}{N-1}\log(N)$ le pas temporel utilisé dans le schéma numérique des différences finies implicites pour résoudre (5)
  - ***Matrix get_A() const, get_B() const et get_C() const***: voir les calculs lors de la résolution de (1) et (5) pour plus de détails à propos de A et B et C=$\tilde{A}$.
  - ***std::vector<double> s_regular_values() const*** et ***std::vector<double> s_not_regular_values() const***: retournent des vecteurs contenant les points en lesquels l'intervalle $[0, L]$ est discrétisé. Le premier implémente la discrétisation à pas constant utilisée lors de la résolution de (1), la seconde implémente celle de (5) (pas non constant).
  - ***std::vector<double> C_t0() const=0***: retourne le vecteur résultat contenant les C(0,s)
- **Classe Put:**
  - **Namespace "Complete":**
    - ***Put()*** et ***Put(double K, double r, double sigma, double T, double L, int N, int M)***: constructeurs par défaut (paramètres initialisés avec les valeurs de l'énoncé) et avec paramètres respectivement
    - ***double C_T(double s) const***: implémente $C(T, s) = \max(K - s, 0)$ pour une option put
    - ***double C_L(double t) const***: implémente $C(t, L) = 0$ pour une option put
    - ***double C_s0(double t) const***: implémente $C(t, 0) = K\exp(-r(T - t))$ pour une option put)
    - ***std::vector<double> C_t0() const***: implémente $C(0, s\_i)$ pour i dans $\{1, ..., N - 1\}$)
  - **Namespace "Reduced"**
    - ***Put()*** et ***Put(double K, double r, double sigma, double T, double L, int N, int M)***: Description identique au namespace "Complete"
    - ***double stilde(double s, double t)***: retourne $\tilde{s} = \log\left(\frac{s}{K}\right) + \left(r - \frac{\sigma^2}{2}\right)(T - t)$ à l'instant t et au point s.
    - ***double ttilde(int m)***: retourne $m \times \frac{T}{M}$ le $m^{ième}$ point de l'intervalle $[0, T]$ discrétisé avec M pas constant.
    - ***double stilde(int i)***: retourne la valeur de $\tilde{s}_i$ à l'état initial, lorsque $\tilde{t} = T$
    - ***double C_stildeN(double ttilde) const***: retourne $\tilde{C}\left(\tilde{t}, \tilde{s} = \tilde{s}_{\max}\right)$ la valeur du payoff à l'instant $\tilde{t}$ lorsque $\tilde{s}$ est maximum, ie lorsque $s = L$
    - ***double C_stilde0(double ttilde) const***: retourne la valeur $\tilde{C}(\tilde{t}, \tilde{s} = -\infty)$ du payoff associé à $\tilde{t}$ lorsque $\tilde{s} = -\infty$.

· ***double C_ttilde0(double ttilde) const***: retourne $\tilde{C}(\tilde{t} = 0, \tilde{s})$ la valeur du payoff associé à stilde lorsque $\tilde{t} = 0$

· ***std::vector<double> C_t0() const***: implémente $C(0, s\_i)$ pour i dans $\{1, ..., N-1\}$)

– **Classe Call:**
Les méthodes de la classe Call sont identiques à celles de la classe Put, en adaptant bien sûr les valeurs retournées par les fonctions au type de l'option.

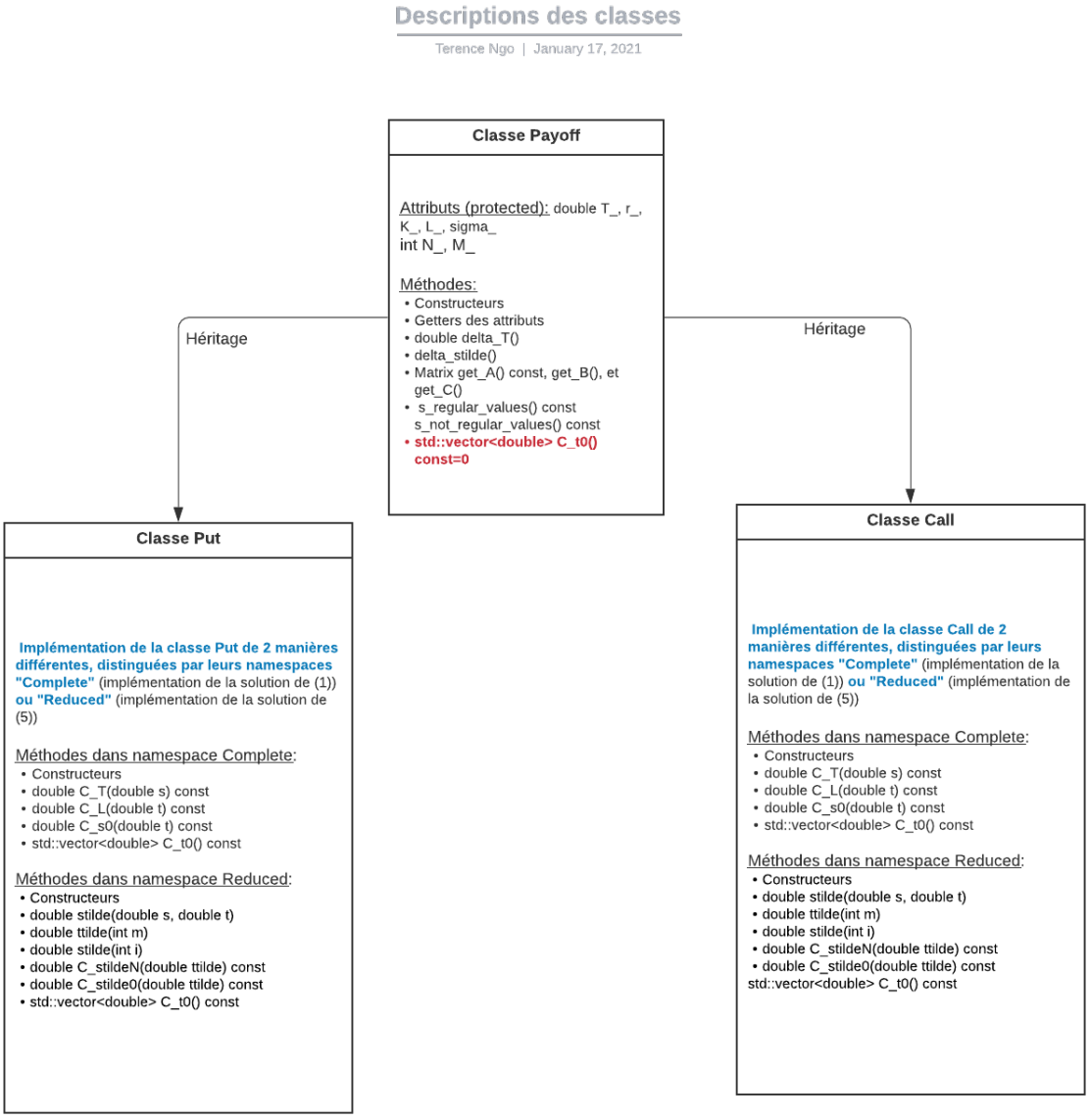

Figure 8: Structure de l'héritage de la classe Payoff

– **Classe Matrix:**

∗ **Méthodes:**

· ***Matrix(int n), Matrix(const Matrix& m)*** et ***Matrix()***: respectivement les constructeurs avec paramètre (seulement la taille de la matrice), de copie et par défaut.

- ***~Matrix()***: le destructeur
- ***int get_n() const*** et ***double operator()(int i, int j) const***: les getters respectivement de la taille de la matrice et de l'élément en position (i, j) de celle-ci
- ***double*** & ***operator()(int i, int j)***: version setter de operator() permettant de modifier l'élément à la position (i, j).
- ***bool is_invertible() const***: retourne True si la matrice courante est inversible, False sinon
- ***double det() const***: retourne le déterminant de la matrice courante
- ***bool is_triangular_sup() const*** et ***bool is_triangular_inf() const***:
- ***bool is_lu_decomp() const***: retourne True si la matrice courante admet une décomposition LU, False sinon. Attention, cette méthode ne s'applique qu'aux matrices tridiagonales

∗ **Fonctions extérieures à la classe:**

- ***std::ostream*** & ***operator<<(std::ostream*** & ***os, const Matrix*** & ***m)***: permet l'affichage du contenu d'une matrice m
- ***std::vector<double> solve(const Matrix*** & A***, const std::vector<double>*** & ***b)***: implémente la méthode de Crout; retourne le vecteur solution X du système matriciel $AX = b$ si A est LU décomposable et tridiaonale
- ***std::vector<double> operator+(const std::vector<double>*** & ***v1, const std::vector<double>*** & ***v2)***: somme les éléments de $v1$ et $v2$ un à un et retourne le vecteur résultant
- ***std::ostream*** & ***operator<<(std::ostream*** & ***os, const Matrix*** & ***v)***: permet l'affichage du contenu d'un vecteur v.

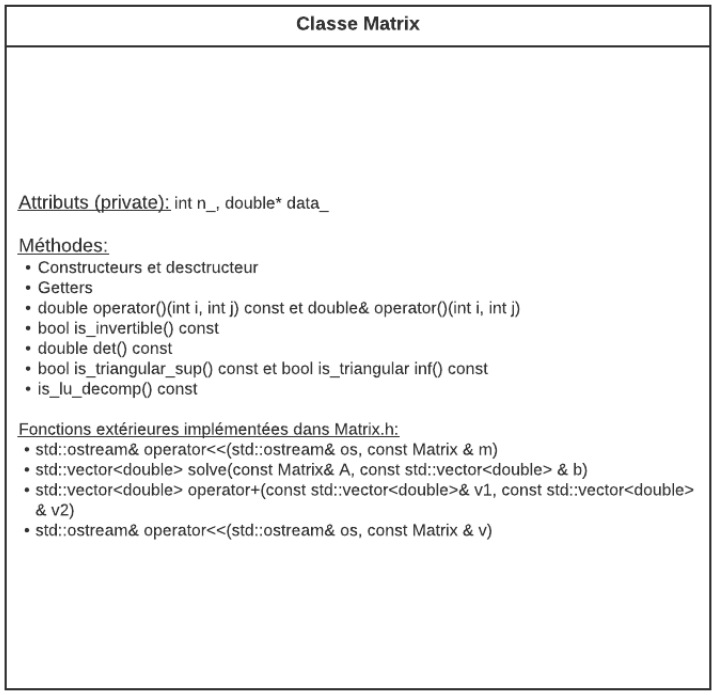

Figure 9: Structure de la classe Matrix

Ci-dessous se trouvent les graphes avec les options superposées en fonction de leur nature,

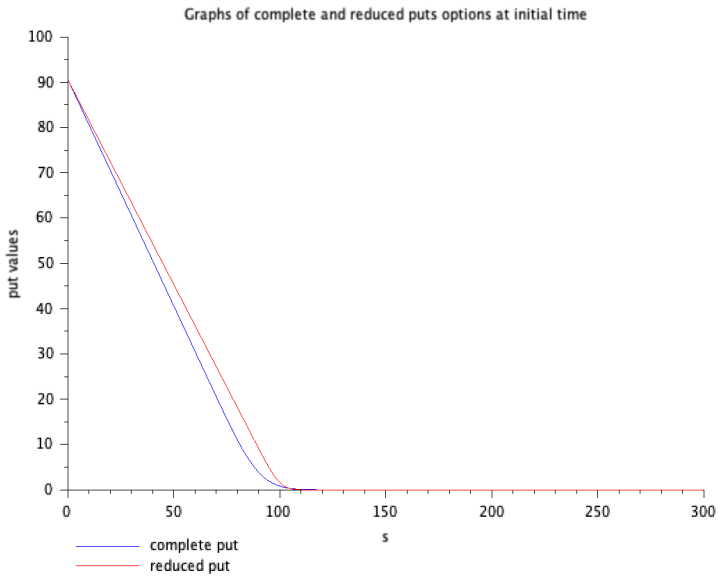

Figure 10: Comparaison des puts solutions de (1) et (5)

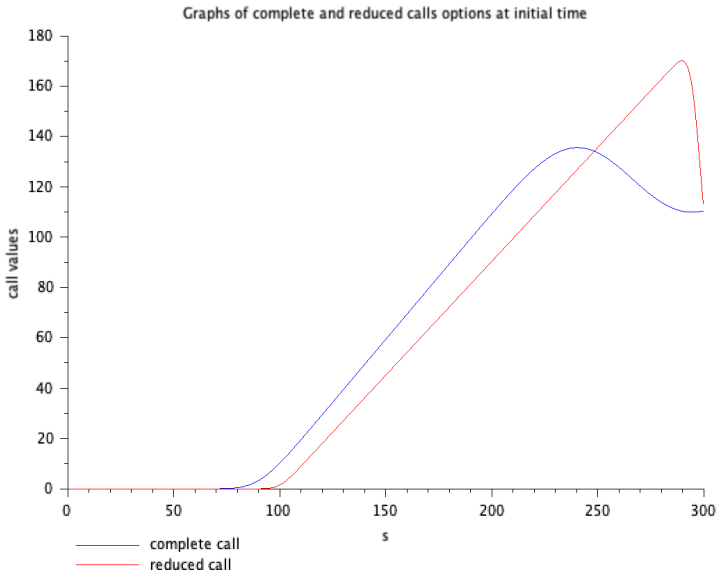

Figure 11: Comparaison des calls solutions de (1) et (5)

ainsi que les courbes d'erreurs résultants:

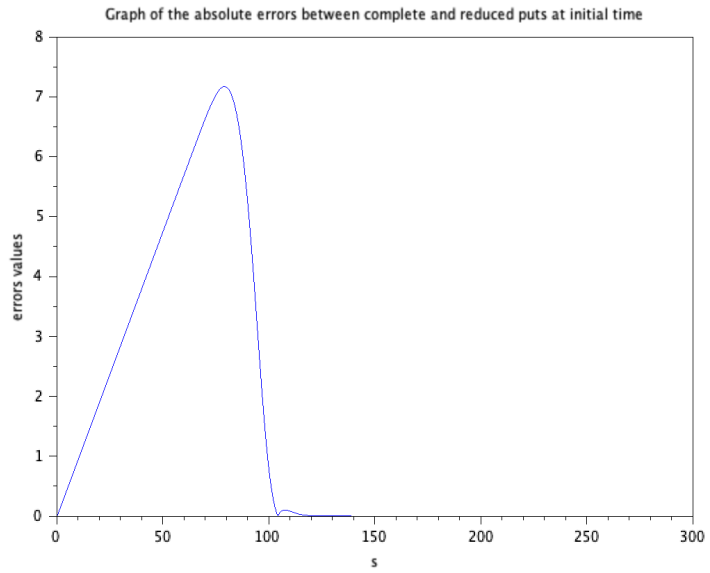

Figure 12: Erreurs absolues entre les puts solutions de (1) et de (5)

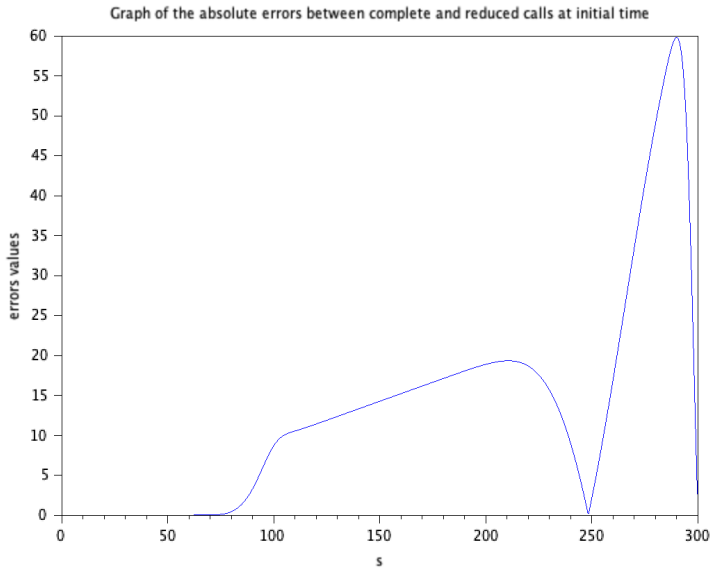

Figure 13: Erreurs absolues entre les calls solutions de (1) et de (5)

On remarque que les puts sont bien plus proches entre eux que ne le sont les calls (cf la différence d'échelle sur l'axe des ordonnées).

6.3 Optimisation

Nous avons utilisé l'outil `std::thread` afin d'accélérer le temps d'exécution du code.
Pour tester son efficacité, nous avons fait varier N et M (en les prenant toujours égaux) les nombres de points utilisés pour les discrétisations respectivement de $[0, L]$ et $[0, T]$.

Ci-dessous se trouve le graphe de comparaison des temps d'exécution avec et sans utilisation de thread:

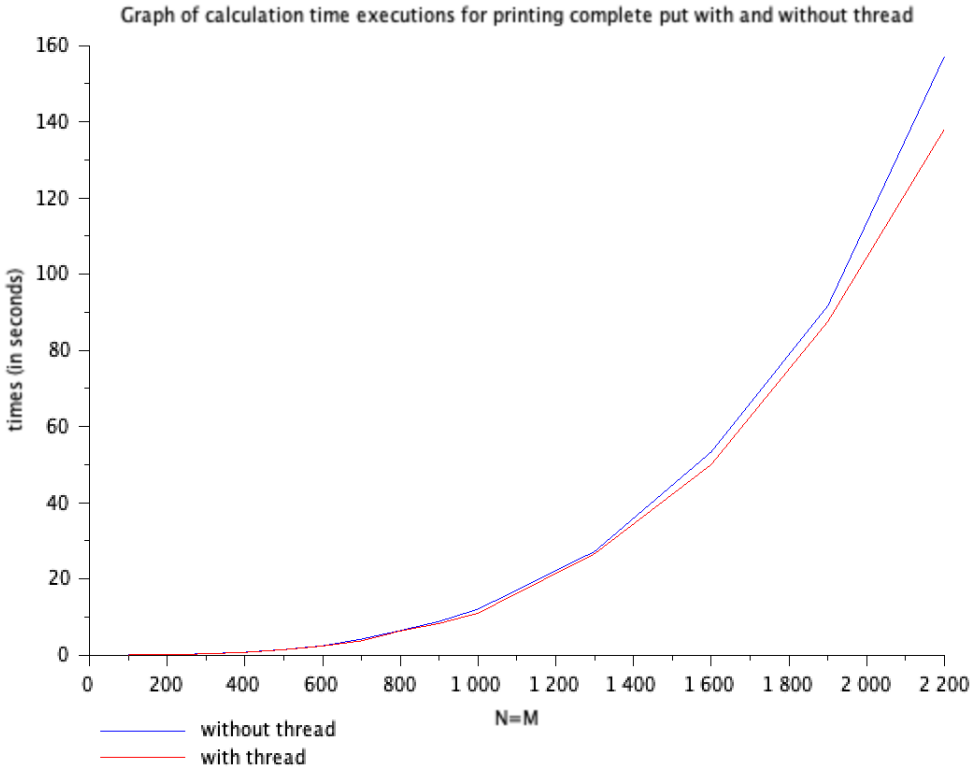

Figure 14: Comparaison des temps de calculs pour afficher le put complet avec/sans thread

Effectivement, on constate que `std::thread` accélère le temps de calcul. Plus N (et M) sont importants, ie plus les calculs sont conséquents, et plus l'écart se creuse entre les 2 courbes. Par exemple, lorsque $N = M = 2200$, C++ met 157.17s à effectuer les calculs sans `std::thread`. Avec l'utilisation de celui, ce temps descend à 138.055s seulement, soit un gain de temps de presque 14%.
`std::thread` est donc particulièrement intéressant lorsque de gros calculs sont mis en jeu.

Conclusion

Nous avons déterminé le prix d'options financiers à travers la résolution de l'équation de Black-Scholes. Nous avons mis en oeuvre le schéma numérique de Crank-Nicholson pour approximer l'équation version complète de Black-Scholes (1) dans un premier temps, puis utilisé le schéma numérique des différences finies implicites pour approximer la version équation de la chaleur" (5) dans un second temps.

Le changement de variable pour passer de la première à la seconde équation a posé des problèmes de discrétisation que nous ne sommes malheureusement pas parvenus à résoudre.
Nous avons alors admis qu'il est possible de discrétiser de manière régulière l'intervalle $[0, L]$ pour (5), sans quoi il n'aurait d'ailleurs pas été possible d'afficher simplement les courbes des erreurs.

Finalement, il semblerait que la version réduite (5) de l'équation de Black-Scholes soit plus facile à résoudre que sa version complète; Toutefois, le retour à l'espace de départ avec s et t peut s'avérer problématique.

Références

[1] Résolution d'une Système Linéaire à Matrice Tridiagonale (*Décomposition LU*) https://www.ljll.math.upmc.fr/hecht/ftp/InfoSci/doc-pdf/Master_book_LU.pdf

[2] Solving Black-Scholes PDE by Crank-Nicolson and Hopscotch Methods http://janroman.dhis.org/stud/I2010/CNhop/CNhop.pdf

[3] Computational Finance with C++ https://sellersgaard.files.wordpress.com/2013/09/c.pdf

[4] Numerical Methods For Option Pricing (*Master Thesis 2012*) http://oa.upm.es/21942/1/TESIS_MASTER_IGOR_VIDIC.pdf

[5] Comparative Error Analysis of the Black-Scholes Equation https://aquila.usm.edu/cgi/viewcontent.cgi?article=1628&context=honors_theses

[6] Modèle de Black-Scholes (*Rapport de Stage à ENS Cachan*) http://dev.ipol.im/~morel/M%E9moires_Stage_Licence_2011/Romain%20WARLOP%20rapport.pdf

[7] Transformation of Black-Scholes PDE to Heat Equation https://en.wikipedia.org/wiki/Black\OT1\textendashScholes_equation