Rapport - Programmation Avancée & Projet

Etude et Implémentation de l'équation de Black et Scholes

Lucas RODRIGUEZ

Janvier 2022

Introduction

Ce rapport a pour but de présenter notre travail sur le projet de l'UE Programmation Avancée & Projet. Nous avons choisi de réaliser ce projet sur le sujet n°2 concernant la **simulation de l'équation de Black et Scholes**. Le présent document présente nos pistes de travail, les difficultés techniques et mathématiques rencontrées, les solutions proposées puis implémentées pour aboutir à un programme informatique répondant aux attentes du sujet.

Objectif du sujet & Plan Dans un premier temps, nous réalisons un important travail bibliographique autour de l'équation de Black-Scholes, puis des méthodes des différences finies. Notre attention s'est plus particulièrement portée sur les discrétisations implicite et de Crank-Nicholson. Après avoir appliqué ces méthodes sur les EDP présentées, un travail sur la structure technique est réalisé au travers de la conception d'un diagramme de classes. Cette étape permet ainsi de prévoir au mieux les différentes imbrications de notre implémentation de la solution en C++.

Table des matières

1 Prospection & travaux théoriques

1.1 Equation de Black-Scholes

On note la fonction $C : [0,T]\times[0,L] \longrightarrow \mathbb{R}$. Il s'agit de la solution de l'EDP complète de Black-Scholes

L'équation de Black Scholes est intialement présentée sous la forme suivante :

$$\frac{\partial C}{\partial t} + rS\frac{\partial C}{\partial S} + \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 C}{\partial S^2} - rC = 0,\ \forall (t,S) \in [0,T]\times[0,L] \tag{1}$$

Elle est qualifiée de **complète**.

Après changement de variable sur C, nous pouvons « approximer » l'EDP complète par une EDP dite **réduite** :

$$\frac{\partial \widetilde{C}}{\partial \widetilde{t}} = \mu \frac{\partial^2 \widetilde{C}}{\partial \widetilde{S}^2},\ \forall (\widetilde{t},\widetilde{S}) \in [0,T]\times[0,L] \tag{2}$$

La solution de l'EDP réduite est alors $\widetilde{C}$, approximation de C et $\mu \in \mathbb{R}$.

Remarque 1 (Valeur de μ)**.** *La première remarque que l'on peut se faire est relative à la valeur numérique de μ. Comme cette dernière n'est pas donnée dans le sujet, il va falloir la déterminer.*

Remarque 2 (Analogie avec la thermodynamique)**.** *Nous pouvons également noter que l'EDP réduite (2) est semblable à l'équation de la chaleur, le nombre μ étant le coefficient de diffusivité thermique.*

1.2 Détermination du coefficient μ

Proposition 1 (Valeur du coefficient μ)**.**

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

Comme indiqué dans l'énoncé, la transformation de (1) à (2) implique un, voire plusieurs changements de variables.

On peut montrer que l'équation (1) est équivalente à une autre expression nous permettant plus facilement de déterminer μ.

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

$$\iff \frac{\partial C}{\partial t} + \frac{1}{2}\sigma^2 \Big(S\frac{\partial}{\partial S}\Big)^2 C + \Big(r - \frac{1}{2}\sigma^2\Big) S\frac{\partial C}{\partial S} - rC = 0 \tag{3}$$

Démonstration. En effet, en utilisant la règle du produit de différentielles sur l'opérateur $S\frac{\partial}{\partial S}$, on obtient :

$$\Big(S\frac{\partial}{\partial S}\Big)^2 C = S\frac{\partial}{\partial S}\Big(S\frac{\partial C}{\partial S}\Big) = S\Big(\frac{\partial S}{\partial S}\frac{\partial C}{\partial S} + S\frac{\partial^2 C}{\partial S^2}\Big) = S\frac{\partial C}{\partial S} + S^2\frac{\partial^2 C}{\partial C^2}$$

En injectant ce résultat dans (3), on a :

$$\frac{\partial C}{\partial t}+\frac{1}{2}\sigma^2\Big(S\frac{\partial C}{\partial S}+S^2\frac{\partial^2 C}{\partial S^2}\Big)+\Big(r-\frac{1}{2}\sigma^2\Big)S\frac{\partial C}{\partial S}-rC=0$$
$$\iff \frac{\partial C}{\partial t}+\frac{1}{2}\sigma^2S^2\frac{\partial^2 C}{\partial S^2}+rS\frac{\partial C}{\partial S}-rC=0$$

On retrouve l'équation (1).

On effectue un premier changement de variables : $\psi : (t,S) \longrightarrow (\tau, y)$ avec : $S=\exp y$ et $t=T-\tau$.

L'application ψ est de classe $\mathcal{C}^1$ donc différentiable sur $\mathbb{R}^2$.

Ainsi :

$$\boxed{S\frac{\partial \bullet}{\partial S}\longrightarrow\frac{\partial \bullet}{\partial y}} \text{ et } \boxed{\frac{\partial \bullet}{\partial t}\longrightarrow-\frac{\partial \bullet}{\partial \tau}}$$

En appliquant ψ à (3), on a :

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

Si on fixe $u=\exp(r\tau)C$, alors dans (3) :

$$u=\exp(r\tau)C \Longrightarrow \boxed{C=u\exp(-r\tau)\Longrightarrow\frac{\partial C}{\partial \tau}=-r\Big(\frac{\partial u}{\partial \tau}\Big)\exp(-r\tau)}$$

On obtient :

$$-\Big(\frac{\partial u}{\partial \tau}e^{-r\tau}+(-r)e^{-r\tau}u\Big)+\frac{1}{2}\sigma^2e^{-r\tau}\frac{\partial^2 u}{\partial y^2}+\Big(r-\frac{1}{2}\sigma^2\Big)e^{-r\tau}\frac{\partial u}{\partial y}-rue^{-r\tau}=0$$
$$-\frac{\partial u}{\partial \tau}e^{-r\tau}+\frac{1}{2}\sigma^2e^{-r\tau}\frac{\partial^2 u}{\partial y^2}+\Big(r-\frac{1}{2}\sigma^2\Big)\frac{\partial u}{\partial y}e^{-r\tau}=0$$
$$\frac{\partial u}{\partial \tau}-\frac{1}{2}\sigma^2\frac{\partial^2 u}{\partial y^2}-\Big(r-\frac{1}{2}\sigma^2\Big)\frac{\partial u}{\partial y}=0$$

On réalise un autre changement de variable :

$$\boxed{\widetilde{S}=y+\Big(r-\frac{1}{2}\sigma^2\Big)\widetilde{t}} \text{ et } \boxed{\widetilde{t}=\tau}$$

On a :

$$\frac{\partial \bullet}{\partial \widetilde{S}}=\frac{\partial \bullet}{\partial \tau}\frac{\partial \tau}{\partial \widetilde{S}}+\frac{\partial \bullet}{\partial y}\frac{\partial y}{\partial \widetilde{S}}=\frac{\partial \bullet}{\partial y}$$

et

$$\frac{\partial \bullet}{\partial \widetilde{t}}=\frac{\partial \bullet}{\partial \tau}\frac{\partial \tau}{\partial \widetilde{t}}+\frac{\partial \bullet}{\partial y}\frac{\partial y}{\partial \widetilde{t}}=\frac{\partial \bullet}{\partial \tau}+\frac{\partial \bullet}{\partial y}(-1)\Big(r-\frac{1}{2}\sigma^2\Big)=\frac{\partial \bullet}{\partial \tau}-\frac{\partial \bullet}{\partial y}\Big(r-\frac{1}{2}\sigma^2\Big)$$

car $y=\widetilde{S}-(r-\frac{1}{2}\sigma^2)\widetilde{t}$

d'où :

$$\frac{\partial u}{\partial \widetilde{t}} + (r - \frac{1}{2}\sigma^2)\frac{\partial u}{\partial y} - \frac{1}{2}\sigma^2\frac{\partial^2 u}{\partial \widetilde{S}^2} - \Big(r - \frac{1}{2}\sigma^2\Big)\frac{\partial u}{\partial y} = 0$$

d'où en simplifiant :

$$\Longrightarrow \frac{\partial u}{\partial \widetilde{t}} = \frac{1}{2}\sigma^2\frac{\partial^2 u}{\partial \widetilde{S}^2}$$

D'où le résultat en identifiant u à $\widetilde{C}$. Par identification, on obtient $\boxed{\mu = \frac{1}{2}\sigma^2}$. ■

1.3 Présentation du domaine de résolution

On pose $\mathcal{D} = [0, T] \times [0, L]$.
Nous effectuons une discrétisation du domaine $\mathcal{D}$.
— $\forall m \in [\![0, M]\!]$, $\boxed{t_m = m\Delta t}$ avec : $\Delta t = \frac{T}{M}$
— $\forall j \in [\![0, N]\!]$, $\boxed{s_j = j\Delta s}$ avec : $\Delta s = \frac{L}{N}$

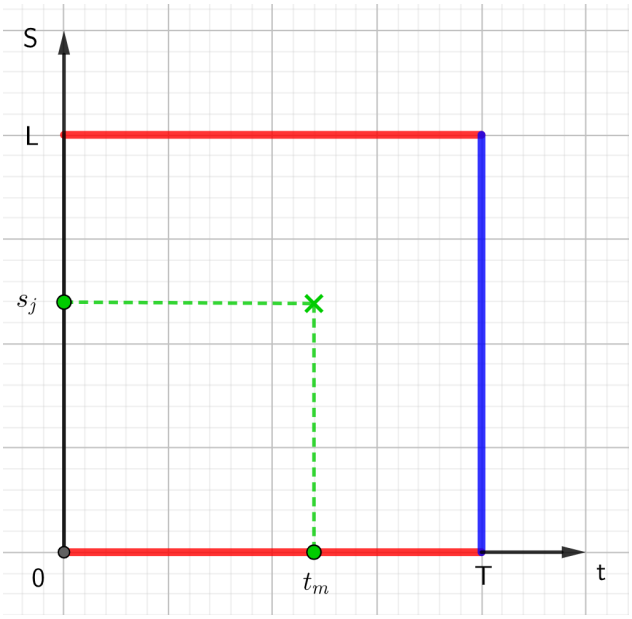


FIGURE 1 – Domaine de résolution sur $\mathcal{D}$

Sur 1, on peut noter :
— Les **segments rouges** représentent les conditions aux bords.
  — Celle en $S = 0$ est appelée condition aux bords basse
  — Celle en $S = L$ est appelée condition aux bords haute
— Le **segment bleu** représente la condition terminale.
L'objectif est de déterminer puis de tracer graphiquement les nuages de points

$$\{(s_j, C(0, s_j)), \forall j \in [\![0, N]\!]\} \text{ et } \{(s_j, \widetilde{C}(0, s_j)), \forall j \in [\![0, N]\!]\}$$

Par la suite, on notera $\mathbf{C}_j^m$, la quantité $C(t_m, s_j)$.

1.4 Etude de l'EDP complète (schéma de Crank-Nicholson)

Le schéma de Crank-Nicholson est un θ-schéma avec $\theta = \frac{1}{2}$. Il s'agit de la moyenne arithmétique entre la méthode explicite et la méthode implicite (décrite dans la section suivante).

Nous commençons par discrétiser les différentes composantes intervenants au sein de l'équation (1) :

$$\begin{aligned}
C(t,s) &\simeq \mathbf{C}_j^m \\
\frac{\partial C}{\partial t} &\simeq \frac{1}{\Delta t}\Big(\mathbf{C}_j^{m+1} - \mathbf{C}_j^m\Big) \\
\frac{\partial C}{\partial S} &\simeq \frac{1}{4\Delta s}\Big(\mathbf{C}_{j+1}^{m+1} - \mathbf{C}_{j-1}^{m+1} + \mathbf{C}_{j+1}^m - \mathbf{C}_{j-1}^m\Big) \\
\frac{\partial^2 C}{\partial S^2} &\simeq \frac{1}{2(\Delta s)^2}\Big(\mathbf{C}_{j+1}^{m+1} - 2\mathbf{C}_j^{m+1} + \mathbf{C}_{j-1}^{m+1} + \mathbf{C}_{j+1}^m - 2\mathbf{C}_j^m + \mathbf{C}_{j-1}^m\Big)
\end{aligned}$$

Nous pouvons ensuite injecter ces discrétisations au sein de (1) : $\forall (m,j) \in [\![0, M-1]\!] \times [\![1, N-1]\!]$,

$$\begin{aligned}
\frac{1}{\Delta t}\Big(\mathbf{C}_j^{m+1} - \mathbf{C}_j^m\Big) + \frac{rj}{4}\Big(\mathbf{C}_{j+1}^{m+1} - \mathbf{C}_{j-1}^{m+1} + \mathbf{C}_{j+1}^m - \mathbf{C}_{j-1}^m\Big) + \\
\frac{1}{4}\sigma^2 j^2\Big(\mathbf{C}_{j+1}^{m+1} - 2\mathbf{C}_j^{m+1} + \mathbf{C}_{j-1}^{m+1} + \mathbf{C}_{j+1}^m - 2\mathbf{C}_j^m + \mathbf{C}_{j-1}^m\Big) - r\mathbf{C}_j^m = 0
\end{aligned}$$

On sépare les termes de temps t_{m+1} et t_m :

$$\begin{aligned}
\frac{1}{\Delta t}\mathbf{C}_j^{m+1} + \frac{rj}{4}\Big(\mathbf{C}_{j+1}^{m+1} - \mathbf{C}_{j-1}^{m+1}\Big) + \frac{\sigma^2 j^2}{4}\Big(\mathbf{C}_{j+1}^{m+1} - 2\mathbf{C}_j^{m+1} + \mathbf{C}_{j-1}^{m+1}\Big) = \\
\frac{1}{\Delta t}\mathbf{C}_j^m - \frac{rj}{4}\Big(\mathbf{C}_{j+1}^m - \mathbf{C}_{j-1}^m\Big) - \frac{\sigma^2 j^2}{4}\Big(\mathbf{C}_{j+1}^m - 2\mathbf{C}_j^m + \mathbf{C}_{j-1}^m\Big) + r\mathbf{C}_j^m
\end{aligned}$$

On multiplie par Δt pour simplifier puis on regroupe les différents termes :

$$\begin{aligned}
\mathbf{C}_j^{m+1}\Big[1 - \frac{\sigma^2 j^2 \Delta t}{2}\Big] + \mathbf{C}_{j+1}^{m+1}\Big[\frac{rj\Delta t}{4} + \frac{\sigma^2 j^2 \Delta t}{4}\Big] + \mathbf{C}_{j-1}^{m+1}\Big[-\frac{rj\Delta t}{4} + \frac{\sigma^2 j^2 \Delta t}{4}\Big] = \\
\mathbf{C}_j^m\Big[1 + \frac{\sigma^2 j^2 \Delta t}{2} + r\Delta t\Big] + \mathbf{C}_{j+1}^m\Big[-\frac{rj\Delta t}{4} - \frac{\sigma^2 j^2 \Delta t}{4}\Big] + \mathbf{C}_{j-1}^m\Big[\frac{rj\Delta t}{4} - \frac{\sigma^2 j^2 \Delta t}{4}\Big]
\end{aligned}$$

On met en place des coefficients pour faciliter l'écriture de l'expression :

$$a_j \mathbf{C}_{j-1}^{m+1} + b_j \mathbf{C}_j^{m+1} + c_j \mathbf{C}_{j+1}^{m+1} = e_j \mathbf{C}_{j-1}^m + d_j \mathbf{C}_j^m + f_j \mathbf{C}_{j+1}^m$$

avec : $\forall j \in [\![1, N-1]\!]$:

$$\begin{aligned}
a_j &= \frac{j\Delta t}{4}\Big(\sigma^2 j - r\Big) \\
b_j &= 1 - \frac{1}{2}\sigma^2 j^2 \Delta t \\
c_j &= \frac{j\Delta t}{4}\Big(\sigma^2 j + r\Big) \\
d_j &= 1 + \frac{1}{2}\sigma^2 j^2 \Delta t + r\Delta t \\
e_j &= -a_j \\
f_j &= -c_j
\end{aligned}$$

Afin de résoudre cette équation, on résoud le système matriciel suivant [1] :

$$\underbrace{\begin{pmatrix} b_1 & c_1 & 0 & \dots & 0 \\ a_2 & b_2 & \ddots & \ddots & \vdots \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & \ddots & \ddots & \ddots & c_{N-2} \\ 0 & \dots & 0 & a_{N-1} & b_{N-1} \end{pmatrix}}_{\in \mathcal{M}_{N-1}(\mathbb{R})} \underbrace{\begin{pmatrix} \mathbf{C}_1^{m+1} \\ \vdots \\ \vdots \\ \vdots \\ \mathbf{C}_{N-1}^{m+1} \end{pmatrix}}_{\in \mathbb{R}^{N-1}} + \underbrace{\begin{pmatrix} a_1(\mathbf{C}_0^{m+1} + \mathbf{C}_0^m) \\ 0 \\ \vdots \\ 0 \\ c_{N-1}(\mathbf{C}_N^{m+1} + \mathbf{C}_N^m) \end{pmatrix}}_{\in \mathbb{R}^{N-1}} =$$

$$\underbrace{\begin{pmatrix} d_1 & -c_1 & 0 & \dots & 0 \\ -a_2 & d_2 & \ddots & \ddots & \vdots \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & \ddots & \ddots & \ddots & -c_{N-2} \\ 0 & \dots & 0 & -a_{N-1} & d_{N-1} \end{pmatrix}}_{\in \mathcal{M}_{N-1}(\mathbb{R})} \underbrace{\begin{pmatrix} \mathbf{C}_1^m \\ \vdots \\ \vdots \\ \vdots \\ \mathbf{C}_{N-1}^m \end{pmatrix}}_{\in \mathbb{R}^{N-1}}$$

On tombe sur le système matriciel :

$$\boxed{M_1 \mathbf{C}^{m+1} + k_{m+1} = M_2 \mathbf{C}^m}$$

Remarque 3 (Observation sur la structure des matricies M_1 et M_2)**.** *On constate que les matrices M_1 et M_2 sont des matrices creuses. Par conséquent, leur stockage en mémoire requiert initialement $(N-1)^2$. Comme $N \sim 10^3$, il est* ***non pertinent*** *de stocker 10^6 coefficients.*

Solution *: Par conséquent, il est cohérent de ne stocker que les bandes diagonales de coefficients :*

$$A = (a_1, \dots, a_{N-1}) \in \mathbb{R}^{N-1}$$
$$B = (b_1, \dots, b_{N-1}) \in \mathbb{R}^{N-1}$$
$$C = (c_1, \dots, c_{N-1}) \in \mathbb{R}^{N-1}$$
$$D = (d_1, \dots, d_{N-1}) \in \mathbb{R}^{N-1}$$

Remarque 4 (Calcul de a_j, b_j, c_j, d_j)**.** *Ces coefficients dépendent de la valeur de j et en C++, les indices vont de 0 à $N-2$ tandis que nous avons besoin des indices de 1 à $N-1$. Il faut donc penser à translater la situation lors du calcul des coefficients des matrices M_1 et M_2.*

La condition terminale s'applique une seule et unique fois [2] sur le vecteur $\mathbf{C}^m$.

Les conditions aux bords exigent le calcul du vecteur $k_{m+1}, \forall m \in [\![0, M]\!]$. Ce calcul est donc dépendant du temps ; il faut donc actualiser le vecteur à chaque itération.

Remarque 5. *M_1 et M_2 sont tridiagonales* ***et*** *inversibles.*

1. Une transformation matricielle préliminaire a été réalisée afin de se trouver avec des matrices carrées propices à la résolution d'un système linéaire
2. lors de l'initialisation de l'algorithme.

Protocole de résolution numérique Afin de résoudre les systèmes linéaires $\forall m \in [\![0, M]\!]$, plusieurs méthodes de résolution peuvent être adoptées :

Problème : Résoudre

$$M_1 \mathbf{C}^{m+1} + k_{m+1} = M_2 \mathbf{C}^m, \forall m \in [\![0, M]\!]$$

où $\mathbf{C}^m$ est le vecteur d'inconnues.

— **Solution 1 - Simple inversion de matrice** : On peut implémenter tout d'abord la méthode du pivot de Gauss afin d'inverser tout simplement la matrice $M_2 \in \mathcal{M}_{N-1}$.
Ainsi :

$$\forall m \in [\![0, M]\!],\ \mathbf{C}^m = M_2^{-1}\Big(M_1 \mathbf{C}^{m+1} + k_{m+1}\Big)$$

Néanmoins, pour chaque étape, le coût de cette méthode est en $\mathcal{O}(N^3)$ ($\frac{2}{3}N^3 +$ **toi**[3] opérations). Nous avons cherché à affiner le coût de la méthode.

— **Solution 2 - Décomposition LU pour matrice tridiagonale** : Afin de limiter le coût des opérations, on peut réaliser une décomposition LU de M_2 **une seule fois** lors de l'initialisation.

$$M_2 = LU$$

avec L matrice triangulaire inférieure à diagonale unitaire et U matrice triangulaire supérieure.
Pour ainsi résoudre $M_2 x = b$ et qu'on a $M_2 = LU$, on résoud $Ly = b$ puis $Ux = y$.
Pour chaque étape, le coût de cette méthode est en $\mathcal{O}(N^2)$ ($2N^2+$**toi** opérations).

Remarque 6 (Solutions algorithmiques)**.** *On peut utiliser l'algorithme de Doolittle ou de Crout afin de calculer la décomposition LU souhaitée.*

— **Solution 3 - Algorithme de Thomas (résolution de système tridiagonal** : L'algorithme de Thomas permet de résoudre les systèmes de la forme $Ax = b$ avec A est tridiagonale.

Remarque 7 (Avantage au niveau du stockage)**.** *Cette méthode ne requiert pas d'avoir rempli un objet de type* `double**` *mais simplement le remplissage des bandes A, B, C et D.*

Cette méthode possède un côut linéaire : $\mathcal{O}(N)$.

Après plusieurs tentatives d'implémentations, nous avons finalement décidé de mettre en place l'algorithme de Thomas pour des considérations d'optimisation du programme final.

1.5 Etude de l'EDP réduite (schéma implicite)

Au point (t_m, s_j), on discrétise les différentes composantes différentielles par :

$$\frac{\partial \widetilde{C}}{\partial \widetilde{t}} \simeq \frac{1}{\Delta t}\Big(\mathbf{C}_j^{m+1} - \mathbf{C}_j^m\Big)$$
$$\frac{\partial^2 \widetilde{C}}{\partial \widetilde{S}^2} \simeq \frac{1}{(\Delta s)^2}\Big(\mathbf{C}_{j+1}^{m+1} - 2\mathbf{C}_j^{m+1} + \mathbf{C}_{j-1}^{m+1}\Big)$$

3. **toi** : termes d'ordre inférieur

On peut alors injecter les discrétisations de ces deux composantes différentielles dans (2) :

$$\frac{1}{\Delta t}\Big(\mathbf{C}_j^{m+1} - \mathbf{C}_j^m\Big) - \frac{\mu}{(\Delta s)^2}\Big(\mathbf{C}_{j+1}^{m+1} - 2\mathbf{C}_j^{m+1} + \mathbf{C}_{j-1}^{m+1}\Big) = 0$$

$$\frac{1}{\Delta t}\mathbf{C}_j^{m+1} - \frac{\mu}{(\Delta s)^2}\Big(\mathbf{C}_{j+1}^{m+1} - 2\mathbf{C}_j^{m+1} + \mathbf{C}_{j-1}^{m+1}\Big) = \frac{1}{\Delta t}\mathbf{C}_j^m$$

$$\mathbf{C}_j^{m+1}\Big(1 + 2\frac{\mu\Delta t}{(\Delta s)^2}\Big) + \mathbf{C}_{j+1}^{m+1}\Big(-\frac{\mu\Delta t}{(\Delta s)^2}\Big) + \mathbf{C}_{j-1}^{m+1}\Big(-\frac{\mu\Delta t}{(\Delta s)^2}\Big) = \mathbf{C}_j^m$$

Au final, on trouve $\forall (m, j) \in [\![0, M]\!] \times [\![0, N]\!]$:

$$\boxed{\mathbf{C}_j^m = \mathbf{C}_j^{m+1}\Big(1 + 2\frac{\mu\Delta t}{(\Delta s)^2}\Big) + \mathbf{C}_{j+1}^{m+1}\Big(-\frac{\mu\Delta t}{(\Delta s)^2}\Big) + \mathbf{C}_{j-1}^{m+1}\Big(-\frac{\mu\Delta t}{(\Delta s)^2}\Big)}$$

Pour plus de clarté, on peut noter $\alpha = \frac{\mu\Delta t}{(\Delta s)^2}$ d'où :

$$\boxed{\mathbf{C}_j^m = -\alpha\mathbf{C}_{j-1}^{m+1} + \Big(1 + 2\alpha\Big)\mathbf{C}_j^{m+1} - \alpha\mathbf{C}_{j+1}^{m+1}}$$

En notation matricielle, on obtient :

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

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

$$\mathbf{C}^m = (\mathbf{C}_1^m, \dots, \mathbf{C}_{N-1}^m)^{\mathbf{T}} \in \mathbb{R}^{N-1} \text{ et } k_m = (a_1\mathbf{C}_0^m, 0, \dots, 0, c_{N-1}\mathbf{C}_N^m)^{\mathbf{T}} \in \mathbb{R}^{N-1}$$

On possède le système suivant :

$$\boxed{M_1\mathbf{C}^{m+1} + k_{m+1} = \mathbf{C}^m}$$

Protocole de résolution numérique Comme détaillé précédemment dans le cadre de l'EDP complète, nous allons utiliser l'algorithme de Thomas afin de résoudre les M systèmes matriciels linéaires.

Cela nous permet d'obtenir un **coût opérationel linéaire** pour chacune des étapes de l'algorithme ; ce qui est très avantageux comparé au côut cubique d'une implémentation du pivot de Gauss.

2 Implémentation de la solution

La composition structurelle détaillée du projet est visible à la fois sur le diagramme de classes UML disponible à la fin du rapport ou dans le répertoire `docs/img/` du projet.
De plus, le projet a été entièrement commenté en utilisant la syntaxe de Doxygen. Cette dernière est trouvable au format HTML et TEX dans le répertoire `documentation/`.

2.1 Gestion des structures de données

Afin de gérer au mieux les différentes structures de données utilisées au sein de l'implémentation, nous avons créer plusieurs classes et structures en C++ afin de regrouper les informations entre elles.

Nous avons implémenté les classes suivantes afin

— **Matrix** Il s'agit d'un alias du type `double**`. En effet, nous avons jugé pertinent de ne pas implémenter une véritable classe `class Matrix` car nous avons estimé que les opérations à réaliser sur le tableau de données étaient élémentaires et que nous garantissions dès le début qu'aucune exception n'allait être levée (ex : indice incorrect, ...).
Par conséquent, nous avons symplement : `typedef double** Matrix;`

— **Mesh** : Cette classe permet de gérer la création et l'accès aux éléments d'une discrétisation temporelle ou spatiale (selon les besoins de l'opérateur).

— **Payoff** : Il s'agit d'une classe abstraite permettant d'implémenter le type de payoff à utiliser pour la simulation de l'EDP de Black-Scholes.
2 autres classes `class Call` et `class Put` sont dérivées de la classe `class Payoff` et implémenter la fonction f de payoff respectif.

— **Option** : Cette classe est très basique et ne possède qu'un seul constructeur. Elle permet juste de stocker au sein de ses membres de données les différents paramètres de l'option.

— **PDE** : Il s'agit d'une classe abstraite implémentant les différentes conditions aux bords, terminale ainsi que le calcul des coefficients pour les deux types d'EDP : réduite & compllète.
`class ReducedPDE` et `class CompletePDE` héritent de `class PDE`.

— **FiniteDifference** : Cette classe abstraite déclare les différentes méthodes afin de résoudre l'EDP via la méthodes des différences finies avec un schéma implicite via `class IMFD` et un schéma de Crank Nicholson via `class CrankNicholsonFD`.

Structures de données conventionnelles En plus des classes présentées ci-dessous, nous avons décidé de principalement utiliser des `std::vector<double>` afin de stocker les coefficients des vecteurs utilisés. Cette utilisation nous permet également d'avoir accès à toute une collection de méthodes pré-implémentées par la STL : insertion, suppression, test de vacuité, ...

2.2 Difficultées rencontrées

La principale source des difficultées rencontrées a été la mise en place des formules mathématiques. En effet, « jongler » entre les indices de discrétisation et des éléments des vecteurs entre les formules théoriques et l'implémentation en C++ a été très difficile et nous a fait perdre énormément de temps. Par exemple : une translation de seulement un indice pouvait provoquer au mieux un *segmentation fault*, au pire un remplissage d'une matrice ou d'un vecteur par des valeurs non souhaitées, ce qui avait quasiment pour conséquence de produire des `nan`[4], inexploitables par la suite.

Une autre difficulté a été lors de la gestion de l'affichage graphique. En effet, il était nécessaire de normaliser les coordonnées afin de les définir dans le nouveau système de coordonnées.

4. Not a number

2.3 Algorithme principal

Afin de réaliser le solveur par différences finies, l'algorithme à exécuter consiste en la succession ordonnée des étapes suivantes :

Initialisation

1. Récupération des paramètres nécessaires à la simulation
2. Discrétisation des domaines spatial et temporel
3. Initialisation à vide des vecteurs de données
4. Calcul des bandes de coefficients pour les matrices (M_1 pour la méthode implicite, M_1 et M_2 pour la méthode de Crank-Nicholson)
5. Ajout des conditions terminales

Traitement Pour chaque instant t_m, $\forall m \in [\![0, M]\!]$:

1. Calcul du vecteur k_m (via le calcul des conditions aux bords)
2. Calcul du membre de droite de l'équation
3. Application de l'algorithme de Thomas (résolution de systèmes matriciels tridiagonaux)

Terminaison

1. Ajout au vecteur solution en début et fin des conditions aux bords
2. Sauvegarde des données en format CSV pour traitements potentiels ultérieurs
3. Création & Paramétrage de la fenêtre graphique
4. Affichage & Fin de l'exécution du programme

2.4 Gestion de l'affichage graphique

Après avoir calculé les vecteurs solutions pour chacune des 3 méthodes, il est nécessaire de réaliser un affichage graphique propre **et** soigné des données afin de les comparer : cette étape consiste donc à réaliser un « wrapper » des différentes méthodes existantes de la SDL.

Remarque 8 (Manque de fonctionnalités de la SDL)**.** *Il est primordial de noter que la SDL ne contient pas de méthodes permettant de réaliser l'affichage de nuages de points. Elle permet juste l'affichage et la gestion d'une fenêtre.*

Il est donc nécessaire d'implémenter des méthodes pour l'affichage d'un cadre au graphique délimitant les données du reste de la fenêtre.

Nous faisons également le choix de représenter graphiquement un point par un cercle[5] de rayon prédéfini et fixe. En effet, la SDL possède une méthode nommée `SDL_RenderDrawPoint(SDL_Renderer renderer, int x, int y);` permettant d'afficher un point. Néanmoins, elle colorie uniquement un pixel, ce qui est très peu clair lors de l'affichage.

5. La création et l'affichage du cercle se feront en utilisant l'algorithme de tracé d'arc de cercle de Bresenham. Voir https://en.wikipedia.org/wiki/Midpoint_circle_algorithm

Normalisation des coordonnées Un problème critique à cette application graphique a été le suivant : l'application du solveur `FiniteDifference.compute_solution()` renvoie les couples $(s_j, C(0, s_j))$ et $(s, \widetilde{C}(0, s_j))$, $\forall j \in [\![0, N]\!]$. Or, le cadre du graphique présente un système de coordonnées cartésiennes $(x, y) \in \mathbb{N}^2$ [6]

Il faut donc, à partir des coordonnées des points dans le domaine réel, réaliser une normalisation afin de lui attribuer un unique couple de coordonnées dans le cadre du graphique.

Nous utilisons pour cela la fonction `double map(...);` implémentée pour l'occasion.

Structures & Classes utilisées pour l'IHM Afin de faciliter la gestion de l'affichage, nous avons découpé notre implémentation en 2 classes : `class Window` et `class Sdl`.

Pour rappel, nous devons au total afficher 4 fenêtres différentes :

1. Une fenêtre affichant la superposition des 2 solutions pour un Put européen
2. L'erreur relative entre ces 2 courbes
3. Une fenêtre affichant la superposition des 2 solutions pour un Call européen
4. L'erreur relative entre ces 2 courbes

Remarque 9. *Nous avons choisi d'afficher les 4 fenêtres en même temps. La première piste d'implémentation a été d'utiliser les threads. Néanmoins, jugeant cette solution trop compliquée, nous avons décidé d'adopter la stratégie suivante.*

La classe `class Window` permet d'ajouter les vecteurs de coordonnées, de les normaliser puis de préparer l'affichage de chacune des courbes qui sera réalisé ultérieurement par la classe `class Sdl`

Remarque 10 (Ajout de textes sur les graphiques)**.** *Nous avons également décidé de rajouter des textes dans le graphique. Nous utilisons la bibliothèque* `sdl_ttf` *afin de les afficher.*

Pour faciliter leur ajout, nous avons mis en place un `struct Label` *qui sont stockés dans un* `std::vector<Label>` *et ajoutés grâce à un setter* `Window.set_text(...)`

La classe `class Sdl` déclare 4 variables de type `Window*` afin de gérer les 4 fenêtres respectives depuis le seul fichier `main.cpp`. Elle permet de déclarer, afficher, écouter les évènements [7] et fermer les fenêtres graphiques.

Résultats des solveurs par différences finies Voici 2 exemples de simulation pour un Put puis un Call européen avec les paramètres spécifiés dans l'énoncé.

6. Les coordonnées sont obligatoirement des entiers car nous travaillons avec des pixels : nous ne pouvons pas éclairer des ≪ portions de pixels ≫ seulement

7. L'écoute d'évènements permet par exemple de mettre en place un ≪ waiter ≫ pour laisser les fenêtres ouvertes tant que l'utilisateur n'appuie pas sur le bouton de fermeture.

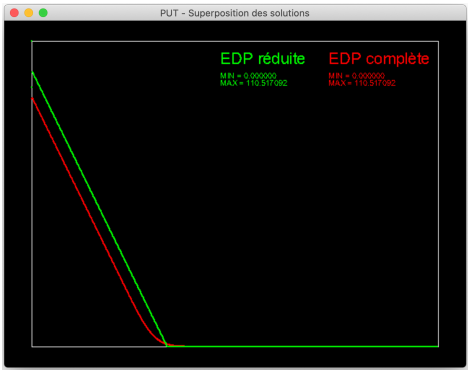


FIGURE 2 – Courbes des 2 solutions pour un Put européen

FIGURE 3 – Courbes des 2 solutions pour un Call européen

3 Conclusion

3.1 Commentaires sur les résultats

Au travers des 2 graphiques présentés en Annexes, on observe que l'erreur relative entre les 2 méthodes ne dépasse pas 10 unités dans chacun des cas. On peut donc conclure que les 2 pricers présentent des résultats relativement similaires. De plus, les résultats trouvés sont très représentatifs de ceux présents dans la littérature en mathématiques financières.

En exportant les données trouvées sur des fichiers CSV puis en les exploitant avec un petit script en Python, nous trouvons que l'erreur moyenne est de 2.4 sur chacune des situations, ce qui nous confortent dans notre conclusion.

3.2 Commentaires personnels & Synthèse générale des travaux

Ce projet nous a permis de réaliser une vaste étude bibliographique puis d'étudier, de concevoir la structure, d'implémenter la solution puis de présenter les résultats sous forme graphique, en l'espace de moins de trois semaines.

L'erreur relative entre les 2 courbes ne dépassant pas 10 dans chacun des cas, nous pouvons conclure qu'il s'agit d'un excellent résulat et que la superposition des 2 courbes solutions, bien qu'imparfaite, témoigne d'une bonne implémentation. En restituant les solutions trouvées dans le contexte de l'étude réalisée, on s'aperçoit qu'il s'agit d'un prix ; or, un écart de prix trouvés de 10 euros semble difficilement tolérable dans des

systèmes de pricing d'options, qui obéissent à des contraintes de justesse et de conformité extrêmement fortes.

Afin de corriger ces erreurs, il faudra sans aucun doute considérer l'étude d'un θ-schéma avec $\theta \neq \frac{1}{2}$, en trouvant une valeur optimale du paramètre de la méthode.

4 Annexes techniques

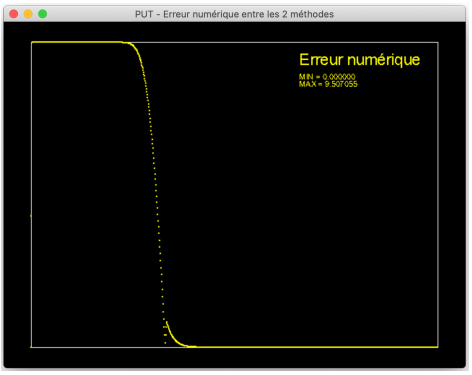


FIGURE 4 – Erreur absolue entre les 2 solutions trouvées pour un Put européen

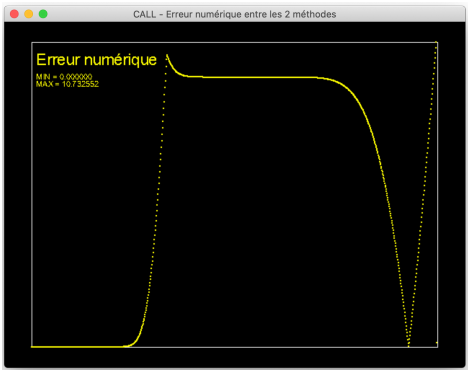
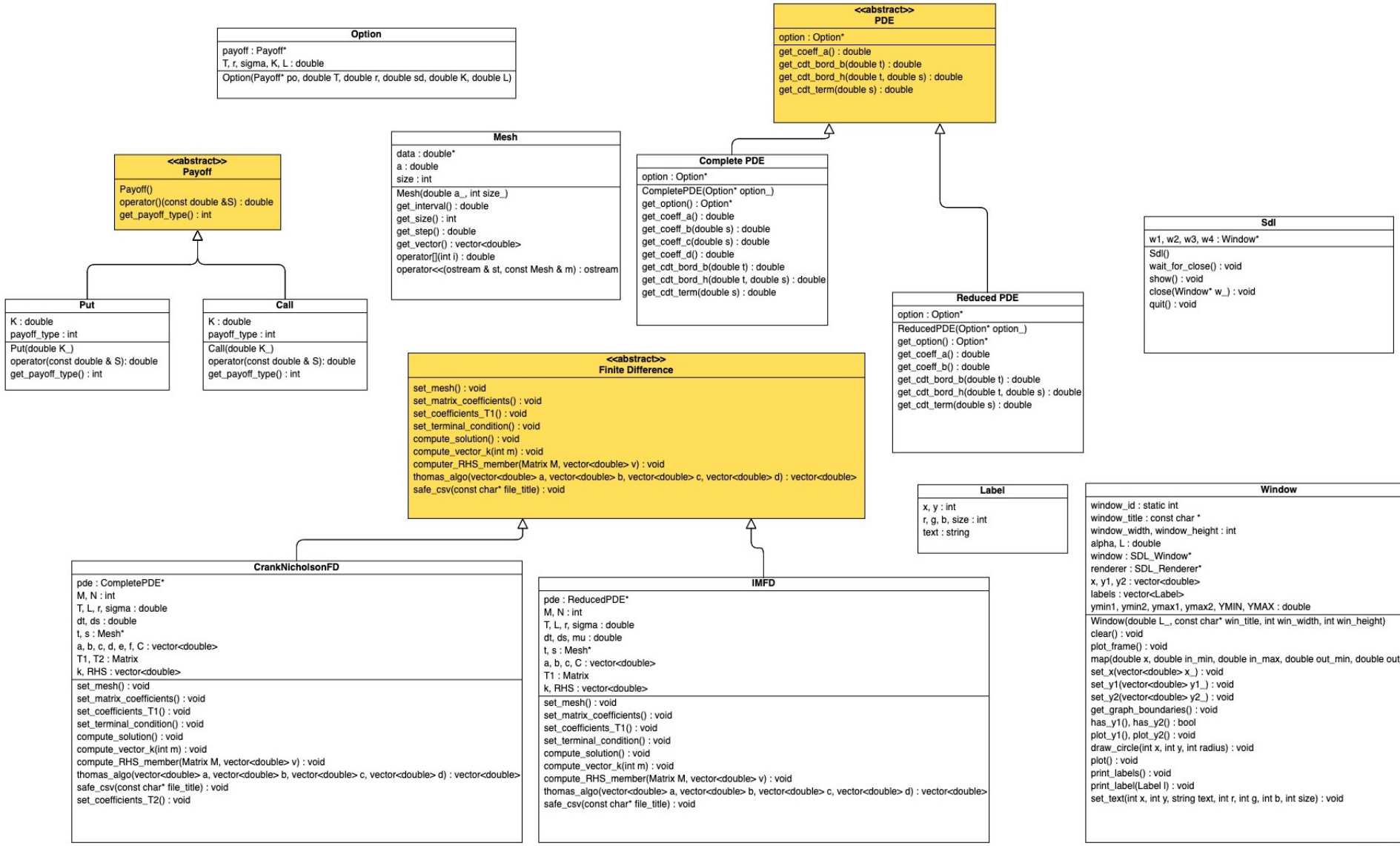


FIGURE 5 – Erreur absolue entre les 2 solutions trouvées pour un Call européen

Le diagramme de classes UML est disponible ci-dessous.

Remarque 11. *Les classes abstraites figurent en couleur orange.*


Option
payoff : Payoff*
T, r, sigma, K, L : double
Option(Payoff* po, double T, double r, double sd, double K, double L)
<<abstract>>
PDE
option : Option*
get_coeff_a() : double
get_cdt_bord_b(double t) : double
get_cdt_bord_h(double t, double s) : double
get_cdt_term(double s) : double
Mesh
data : double*
a : double
size : int
Mesh(double a_, int size_)
get_interval() : double
get_size() : int
get_step() : double
get_vector() : vector<double>
operator[](int i) : double
operator<<(ostream & st, const Mesh & m) : ostream
Complete PDE
option : Option*
CompletePDE(Option* option_)
get_option() : Option*
get_coeff_a() : double
get_coeff_b(double s) : double
get_coeff_c(double s) : double
get_coeff_d() : double
get_cdt_bord_b(double t) : double
get_cdt_bord_h(double t, double s) : double
get_cdt_term(double s) : double
<<abstract>>
Payoff
Payoff()
operator()(const double &S) : double
get_payoff_type() : int
Put
K : double
payoff_type : int
Put(double K_)
operator(const double & S): double
get_payoff_type() : int
Call
K : double
payoff_type : int
Call(double K_)
operator(const double & S): double
get_payoff_type() : int
Reduced PDE
option : Option*
ReducedPDE(Option* option_)
get_option() : Option*
get_coeff_a() : double
get_coeff_b() : double
get_cdt_bord_b(double t) : double
get_cdt_bord_h(double t, double s) : double
get_cdt_term(double s) : double
Sdl
w1, w2, w3, w4 : Window*
Sdl()
wait_for_close() : void
show() : void
close(Window* w_) : void
quit() : void
<<abstract>>
Finite Difference
set_mesh() : void
set_matrix_coefficients() : void
set_coefficients_T1() : void
set_terminal_condition() : void
compute_solution() : void
compute_vector_k(int m) : void
computer_RHS_member(Matrix M, vector<double> v) : void
thomas_algo(vector<double> a, vector<double> b, vector<double> c, vector<double> d) : vector<double>
safe_csv(const char* file_title) : void
Label
x, y : int
r, g, b, size : int
text : string
Window
window_id : static int
window_title : const char *
window_width, window_height : int
alpha, L : double
window : SDL_Window*
renderer : SDL_Renderer*
x, y1, y2 : vector<double>
labels : vector<Label>
ymin1, ymin2, ymax1, ymax2, YMIN, YMAX : double
Window(double L_, const char* win_title, int win_width, int win_height)
clear() : void
plot_frame() : void
map(double x, double in_min, double in_max, double out_min, double out_max) : double
set_x(vector<double> x_) : void
set_y1(vector<double> y1_) : void
set_y2(vector<double> y2_) : void
get_graph_boundaries() : void
has_y1(), has_y2() : bool
plot_y1(), plot_y2() : void
draw_circle(int x, int y, int radius) : void
plot() : void
print_labels() : void
print_label(Label l) : void
set_text(int x, int y, string text, int r, int g, int b, int size) : void
CrankNicholsonFD
pde : CompletePDE*
M, N : int
T, L, r, sigma : double
dt, ds : double
t, s : Mesh*
a, b, c, d, e, f, C : vector<double>
T1, T2 : Matrix
k, RHS : vector<double>
set_mesh() : void
set_matrix_coefficients() : void
set_coefficients_T1() : void
set_terminal_condition() : void
compute_solution() : void
compute_vector_k(int m) : void
compute_RHS_member(Matrix M, vector<double> v) : void
thomas_algo(vector<double> a, vector<double> b, vector<double> c, vector<double> d) : vector<double>
safe_csv(const char* file_title) : void
set_coefficients_T2() : void
IMFD
pde : ReducedPDE*
M, N : int
T, L, r, sigma : double
dt, ds, mu : double
t, s : Mesh*
a, b, c, C : vector<double>
T1 : Matrix
k, RHS : vector<double>
set_mesh() : void
set_matrix_coefficients() : void
set_coefficients_T1() : void
set_terminal_condition() : void
compute_solution() : void
compute_vector_k(int m) : void
compute_RHS_member(Matrix M, vector<double> v) : void
thomas_algo(vector<double> a, vector<double> b, vector<double> c, vector<double> d) : vector<double>
safe_csv(const char* file_title) : void