RAPPORT DE PAP

Black Scholes

12 janvier 2020

Abiola Trésor Djigui
Guillaume Shi de Milleville

Table des matières

0.1 Sujet

Le sujet consiste à résoudre

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

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

avec un temps terminal T donné, r le taux d'intérêt du marché et σ la volatilité de l'actif.
La fonction $C(s,t)$ est définie sur $[0,T] \times [0,L]$
Ceci afin de calculer $C(0,s)$ pour tout $s \in [0,L] C(0,s)$ avec les deux méthodes ci-dessus.

0.2 Théorie Mathématique

0.2.1 Différences finies

La résolution d'équations différencielles via les méthodes de différences finies consiste à discrétiser les espaces de variables en intervalles réguliers. Le problème de résolution revient donc à déterminer la fonction en chacun des points de la grille issue de la discrétisation.

Dans un second temps, on approxime les dérivées partielles dans l'équation par des différences. Ces approximations définissent le type de méthode de différence finie adopté pour la résolution.
Pour la résolution de l'EDP de Black-Scholes, nous utiliserons la méthode de Crank-Nicholson et la méthode des differences finies implicite.

0.2.2 Crank Nicholson

Pour résoudre l'équation différentielle avec la méthode Crank Nicholson :

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

On adopte les approximations suivantes :

$$\frac{\partial C_{i-1/2,j}}{\partial t} = \frac{C_{i,j} - C_{i-1,j}}{\delta t}$$

$$\frac{\partial C_{i-1/2,j}}{\partial S} = \frac{1}{2}[\frac{f_{i-1,j+1} - f_{i-1,j-1}}{2\delta S} + \frac{f_{i,j+1} - f_{i,j-1}}{2\delta S}]$$

$$\frac{\partial^2 C_{i-1/2,j}}{\partial S^2} = \frac{1}{2}[\frac{C_{i-1,j+1} - 2C_{i-1,j} + C_{i-1,j-1}}{\partial S^2} + \frac{C_{i,j+1} - 2C_{i,j} + C_{i,j-1}}{\partial S^2}]$$

où i et j représentent les indices sur la grille. En injectant ces approximations dans l'équation différentielle obtient :

$$-a_j C_{i-1,j-1} + (1 - b_j) C_{i-1,j} - c_j C_{i-1,j+1} = a_j C_{i,j-1} + (1 + b_j) C_{i,j} + c_j C_{i,j+1}$$

avec

$$a_j = \frac{\delta t}{4}(\sigma^2 j^2 - r j)$$

$$b_j = -\frac{\delta t}{2}(\sigma^2 j^2 + r)$$

$$c_j = \frac{\delta t}{4}(\sigma^2 j^2 + r j)$$

Cette équation revient en réalité à résoudre un système de M-1 équations à M-1 variables pour chaque i=N-1,. . .,1.

On passe donc à la forme matricielle suivante :

$$CF_{i-1} = DF_i + K_{i-1} + K_i \forall i \in (N, 1)$$

où

$$F_i = \begin{bmatrix} F_{i,1} \\ F_{i,2} \\ \vdots \\ F_{i,M-1} \end{bmatrix}$$

$$K_i = \begin{bmatrix} a_1 C_{i,0} \\ 0 \\ \vdots \\ 0 \\ c_{M-1} C_{i,M} \end{bmatrix}$$

$$C = \begin{bmatrix} 1-b_1 & -c_1 & 0 & \cdots & 0 & 0 \\ -a_2 & 1-b_2 & -c_2 & \cdots & 0 & 0 \\ 0 & -a_3 & 1-b_3 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & -a_{M-1} & 1-b_{M-1} \end{bmatrix}$$

$$D = \begin{bmatrix} 1+b_1 & c_1 & 0 & \cdots & 0 & 0 \\ a_2 & 1+b_2 & c_2 & \cdots & 0 & 0 \\ 0 & a_3 & 1+b_3 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & a_{M-1} & 1+b_{M-1} \end{bmatrix}$$

On initialise le système (calcul de F_n) à l'aide des conditions initiales puis par récurrence, on détermine Fi pour $i = N-1, ..., 0$.

0.2.3 Via la méthode des différences finies implicite

0.2.3.1 Changements de variables

En faisant les changements de variables suivants

$$t = T - \frac{\tau}{\frac{1}{2}\sigma^2}$$

et

$$S = Ke^x$$

on peut écrire $C(t,S) = Kv(\tau, x)$ avec v vérifiant

$$\frac{\partial v}{\partial \tau} = \frac{\partial^2 v}{\partial x^2} + (k-1)\frac{\partial v}{\partial x} - kv$$

où

$$k = \frac{r}{\sigma^2/2}$$

Ensuite, on pose $v = exp(\alpha x + \beta\tau)\tilde{C}(\tau, x)$ où α et β sont à choisir avec un souci de simplification de l'équation à résoudre. En substituant v par sa nouvelle expression dans l'équation, on obtient :

$$\alpha = -\frac{k-1}{2}$$

$$\beta = -\frac{(k+1)^2}{4}$$

$$\frac{\partial \tilde{C}}{\partial \tau} = \frac{\partial^2 \tilde{C}}{\partial x^2}$$

Les nouveaux espaces des variables sont : $-\infty < x < log(L/K)$ et $0 \leq \tau \leq \frac{\sigma^2 T}{2}$
Il nous faut donc résoudre cette équation avec les conditions :

$$\tilde{C}(\tau, -\infty) \qquad \forall 0 \leq \tau \leq \frac{\sigma^2 T}{2}$$

$$\tilde{C}(\tau, log(L/K) \qquad \forall 0 \leq \tau \leq \frac{\sigma^2 T}{2}$$

$$\tilde{C}(0, x) \qquad \forall -\infty < x < log(L/K)$$

que nous déterminons facilement via la relation liant C et $\tilde{C}$.

0.2.3.2 Résolution

Le but est de résoudre l'équation réduite précédente via la méthode des différences finies implicite puis de repasser aux valeurs de C via la relation

$$C(S,t) = Ke^{\alpha x + \beta \tau} \tilde{C}(x, \tau)$$

Pour la discrétisation de l'espace de x, on devra fixer une borne inférieure. En effet, il est impossible de discrétiser de manière finie l'intervalle infini $[-\infty, log(L/K)]$. On prend X^- pour borne inférieure et l'espace des x devient en conséquence $[X^-, log(L/K)]$. Pour plus de simplicité, on pose $u = \tilde{C}$ pour la suite.
On adopte les approximations suivantes :

$$\frac{\partial u_{i,j}}{\partial \tau} = \frac{u_{i,j} - u_{i-1,j}}{\delta \tau}$$

$$\frac{\partial u_{i,j}}{\partial x^2} = \frac{u_{i+1,j-1} - 2u_{i+1,j} + u_{i+1,j+1}}{\delta x^2}$$

En injectant ces approximations dans l'équation de départ puis en posant $\gamma = \frac{\delta \tau}{\delta x^2}$, on

obtient :

$$-\gamma u_{i+1,j-1} + (1+2\gamma) u_{i+1,j} - \gamma u_{i+1,j+1} = u_{i,j}$$

ce qui donne sous forme matricielle

$$CU_{i+1} = U_i + K_{i+1}$$

où

$$C = \begin{bmatrix} 1+2\gamma & -\gamma & 0 & \cdots & 0 & 0 \\ -\gamma & 1+2\gamma & -\gamma & \cdots & 0 & 0 \\ 0 & -\gamma & 1+2\gamma & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & -\gamma & 1+2\gamma \end{bmatrix}$$

$$U_i = \begin{bmatrix} u_{i,x^-} \\ \vdots \\ u_{i,0} \\ \vdots \\ u_{i,x^+-1} \end{bmatrix}$$

$$K_i = \begin{bmatrix} \gamma u_{i,x^-} \\ 0 \\ \vdots \\ 0 \\ \gamma u_{i,x^+} \end{bmatrix}$$

On initialise le système avec les conditions initiales précisées plus haut puis par récurrence, on détermine les valeurs de $u = \tilde{C}$ en chaque point de la grille de discrétisation. Enfin, on déduit de cette résolution de l'équation réduite, les valeurs de C(t,S).

0.3 Implémentation en C++ : problèmes, solutions et structures

0.3.1 Problèmes rencontrés et solutions adoptées

0.3.1.1 Problème de complexité

Pour la résolution du problème de l'EDP de Black-Scholes, nous constatons qu'il y a des systèmes matriciels à résoudre. Celà nous pousse à utiliser des matrices carrées avec des opérations de multiplications et d'additions matricielles.
Deux problèmes se posent ici :

— la complexité spatiale de la représentation en mémoire des matrices de tailles importantes.

— la complexité temporelle de création de ces matrices ainsi que des opérations sur les matrices.

En effet, pour une discrétisation spatiale en M intervalles, on a une complexité en $O(M^2)$ pour la création ou le stockage et une compexité en $O(M^3)$ pour les opérations de produits.

$$AB = \begin{pmatrix} \sum_{k=1}^{n} a_{1,k}b_{k,1} & \sum_{k=1}^{n} a_{1,k}b_{k,2} & \cdots & \sum_{k=1}^{n} a_{1,k}b_{k,p} \\ \sum_{k=1}^{n} a_{2,k}b_{k,1} & \sum_{k=1}^{n} a_{2,k}b_{k,2} & \cdots & \sum_{k=1}^{n} a_{2,k}b_{k,p} \\ \vdots & \vdots & \ddots & \vdots \\ \sum_{k=1}^{n} a_{m,k}b_{k,1} & \sum_{k=1}^{n} a_{m,k}b_{k,2} & \cdots & \sum_{k=1}^{n} a_{m,k}b_{k,p} \end{pmatrix}$$

Heureusement, nous pouvons contourner ce problème de complexité puisque les matrices aux nous sommes confrontées sont tridiagonale. Nous allons donc adapter le code de manière à ne stocker en mémoire que les coefficients non nuls, i.e ceux sur les diagonales inférieure, principale et supérieure. Aussi, l'opération *produit matriciel* est adapté de manière à ne s'intéresser qu'aux coefficients non nuls des matrices.

Cette solution permet de diminuer significativement la complexité de notre algorithme. En effet, nous passons de complexités quadratique et cubique à une complexité linéaire : création, stockage et produit en $O(M)$.

0.3.1.2 Problème de discrétisation

L'espace des x pour la résolution de l'équation différentielle est $[-\infty, H]$, pour H>0. Il n'est pas possible de discrétiser un tel intervalle. Lors de l'implémentation, on a donc dû

fixer une borne inférieure dans l'espace transformé (espace des x). En fonction de cette borne inf, le passage dans l'espace initial (espace avant transformation) donne lieu à des prix plus ou moins concentrés vers 0 (ou vers L). La borne inf dans l'espace transformé a donc dû être choisie judicieusement pour avoir des valeurs de C bien étalées après transformation inverse.

0.3.1.3 Problème de complexité

Lors du calcul des erreurs, on a rencontré une difficulté : les prix sous-jacents pour les quels on connait le prix de l'option ne coincident pas pour les deux méthodes. Ceci est dû au fait que lors de la transformation inverse des abscisses pour passer aux prix réels, perd l'écart régulier créé lors de la discrétisation à l'échelle logarithmique. Alors si on fait la différences des prix en balayant simplement le tableau [prix sous jacent, prix option], on a pas vraiment les bonnes correspondances et donc l'évaluation d'erreur serait fausse.

Alors, pour résoudre ce problème, on va certes balayer le tableau [prix sous jacent, prix option] mais on ne calculera l'erreur associé que si la différence entre les prix (abscisses) des deux méthodes est inférieure à une constante ϵ choisie. Pour notre implémentation, on a choisi $\epsilon = 4$.

0.3.2 Description des classes

Nous avons dû créer différentes classes afin de factoriser le mieux possible le code. Les différentes classes que nous avons utilisé sont les suivantes :

- **Payoff** : représente la fonctionnalité du payoff.
- **PayoffCall** : hérite de **Payoff** et représente le payoff d'un Call.
- **PayoffPut** : hérite de **Payoff** et représente les payoff d'un Put.
  (Figure **4**)
- **EurOption** : composé d'un **Payoff** et représente une option de type européene.
  (Figure **5**)
- **BlackScholesPDE** : représente l'équation de black-scholes complète associée à une **EurOption**.
  (Figure **6**)
- **ReducedPDE** : représente l'équation réduite assocé à une **EurOption**.
  (Figure **7**)
- **CrankNicholson** : représente la résolution par la méthode de crank-Nicholson d'une **BlackScholesPDE**.
  (Figure **8**)
- **FdmImplicite** : représente la résolution par la méthode des différences finies d'une **ReducedPDE**.
  (Figure **9**)
- **Error** : qui permet de gérer les erreus.
- **BaseMatrix** : classe abstraite représentant une colonne.
- **ColMatrix** : hérite de **BaseMatrix** et représente les matrices colonnes.
- **TridiagoMatrix** : hérite de **BaseMatrix** et représente les matrices tridiagonales.
  (Figure **10**)
- **TidiagoLinearSolver** : représente la résolution d'un système linéire tridiagonal.
  (Figure **11**)

0.4 Conclusion

Pour conclure, la principal problème rencontré dans ce projet est celui de la complexité. Au début, nous avons eu beaucoup de mal à faire fonctionner notre programme en temps restreint en raison des tailles de discrétisations imposées pour l'application numérique. Mais très vite, ce problème a été contourné en définissant une classe qui gère de manière intelligente les matrices de grandes tailles.

Les courbes obtenues par les deux méthodes sont très proches, aussi bien dans le cas du *Call* que celui du *Put*. Nous observons une allure reconnaissable pour les *Put*. Contrairement à ce à quoi on s'attendait, nous n'observons pas de parité *Call-Put*. Mais celà est certainement dû à la condition de bord à droite pour le *Call*, qui elle même ne satisfait pas la parité *C-P* avec la condition de bord à droite pour le *Put*.
Les graphes suivants représentent ceux que nous avons obtenus à l'issus de ce projet.

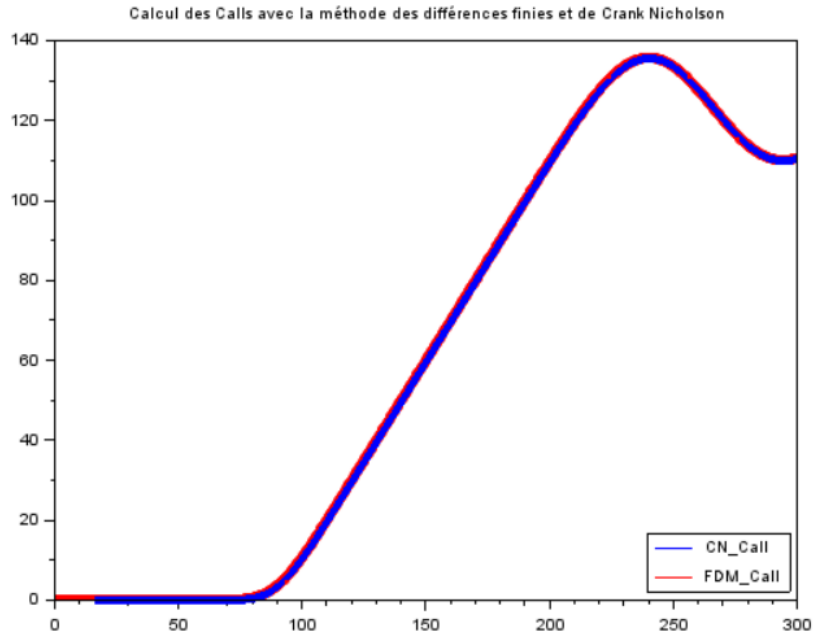


Figure 1: Prix du Call pour les deux méthodes en fonction du prix sous jacent

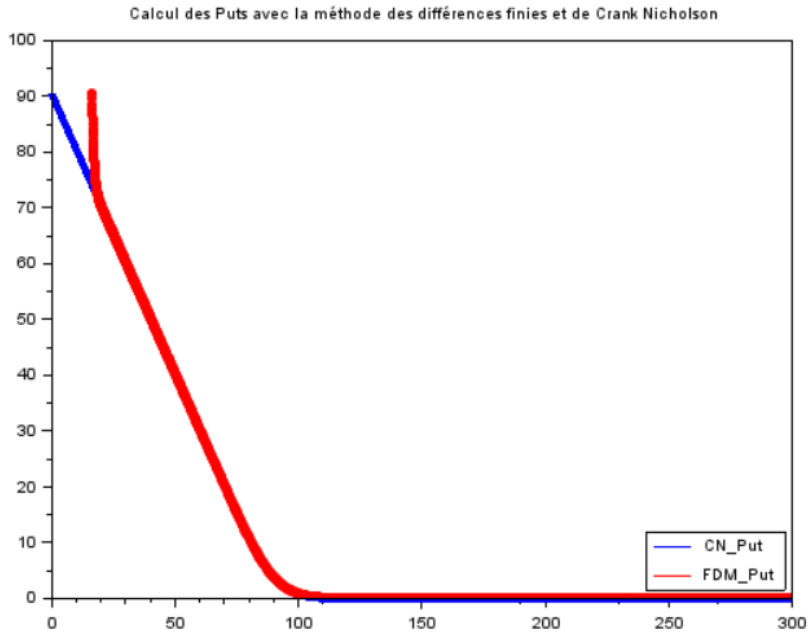
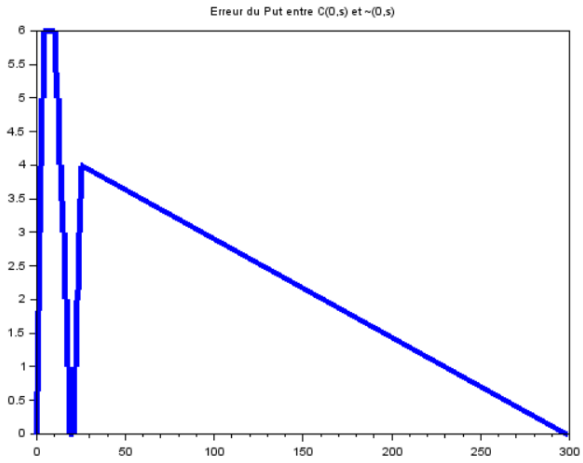


FIGURE 2: Prix du Put pour les deux méthodes en fonction du prix sous jacent



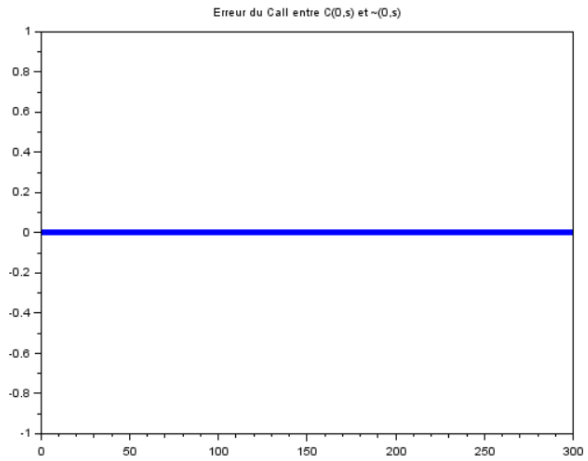


Figure 3: Erreur Put et call

0.5 Annexe

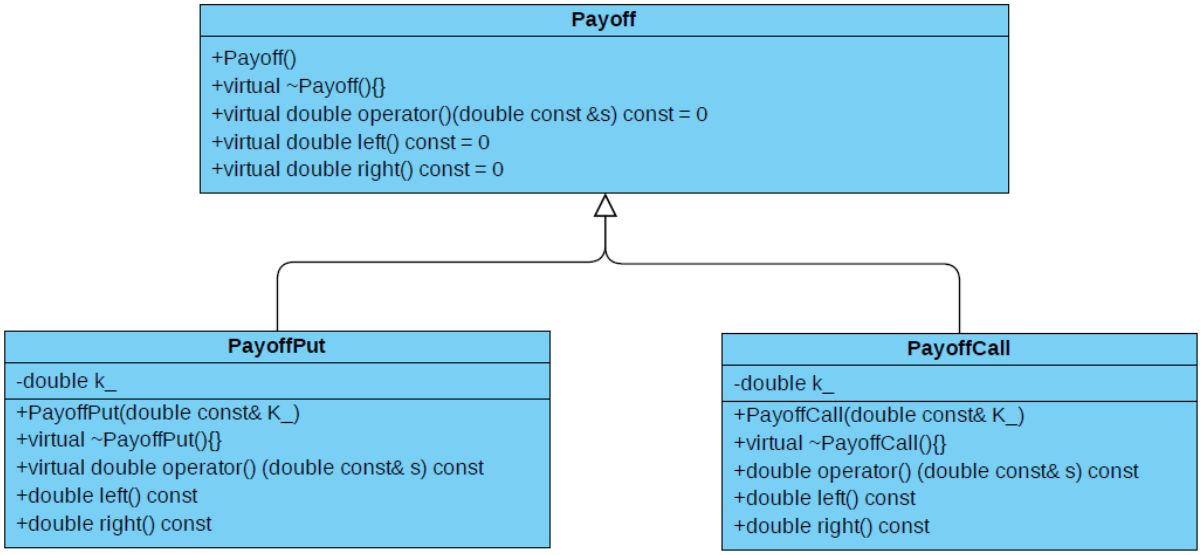


Figure 4: Payoff

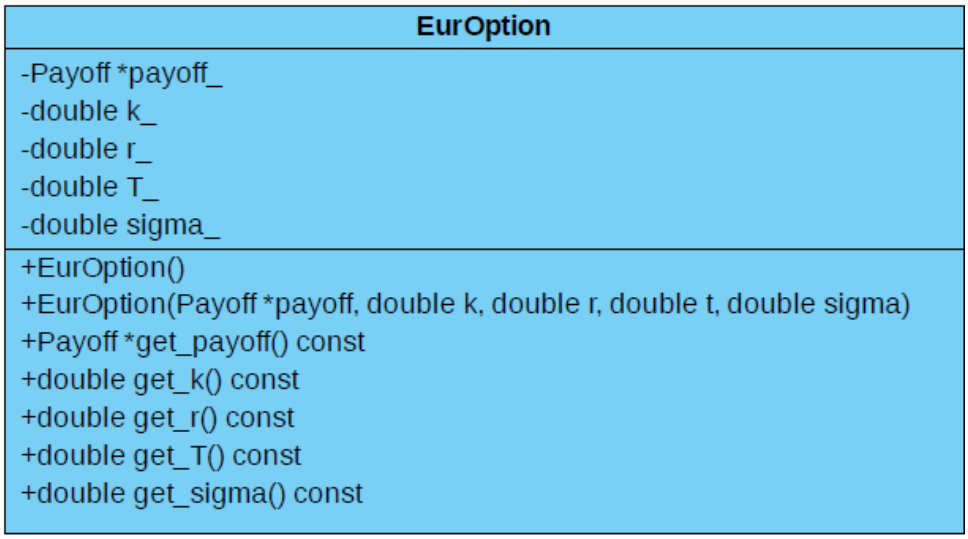


Figure 5: Option Européenne

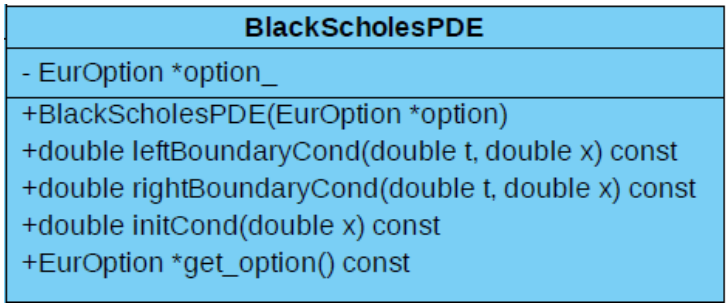


Figure 6: Equation de Black Scholes

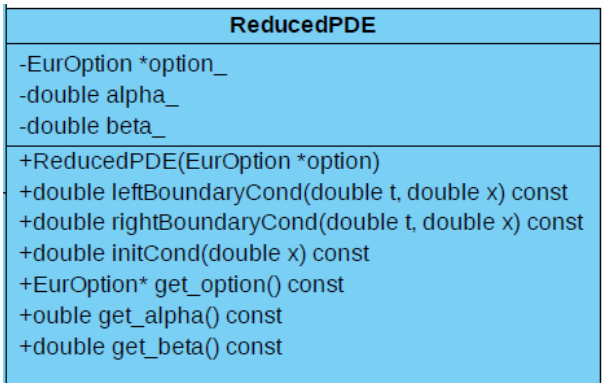


Figure 7: Equation Réduite

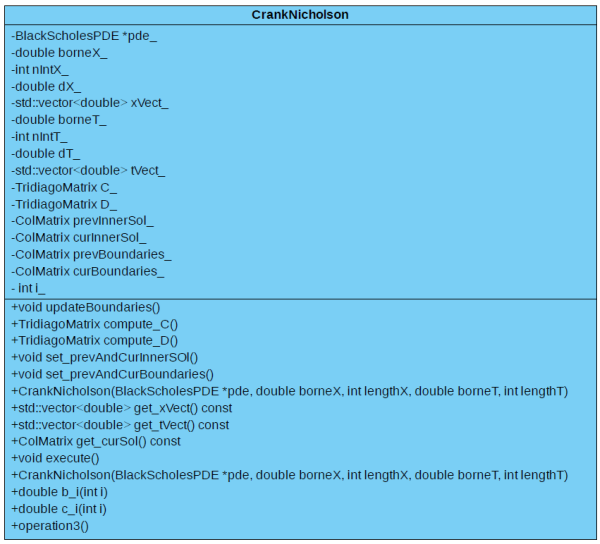


Figure 8: Crank Nicholson

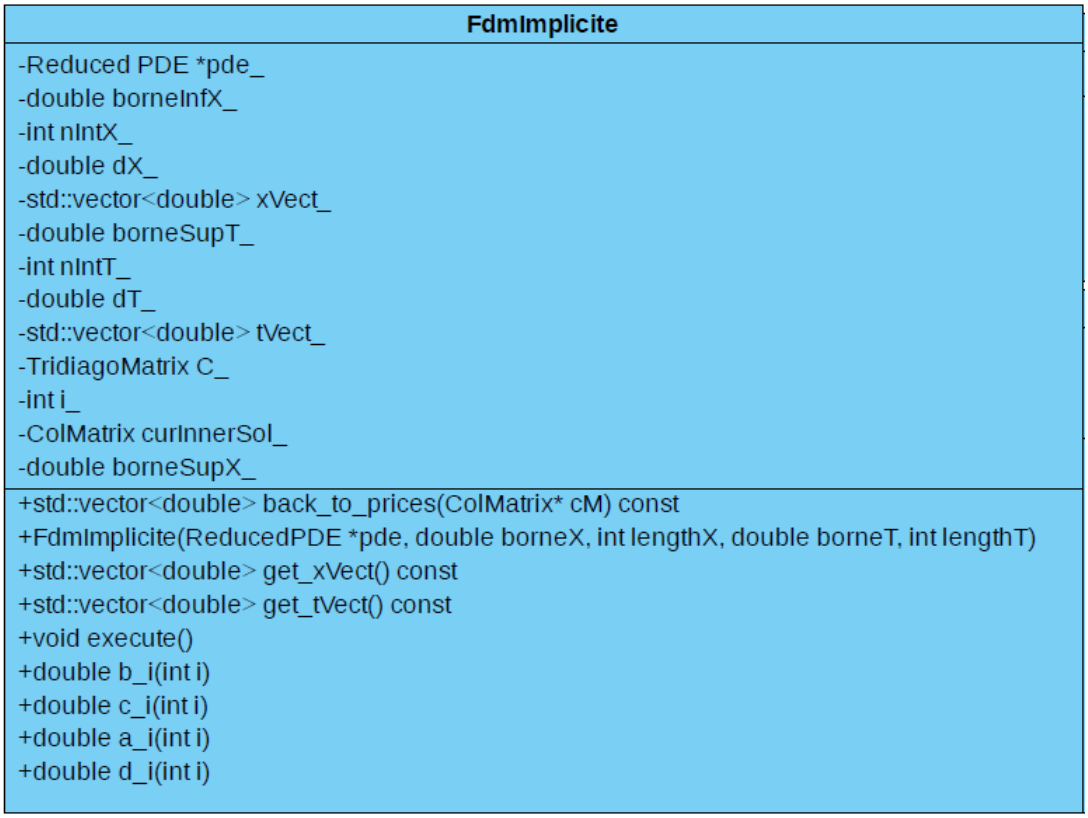


Figure 9: Méthode des différences finies implicite

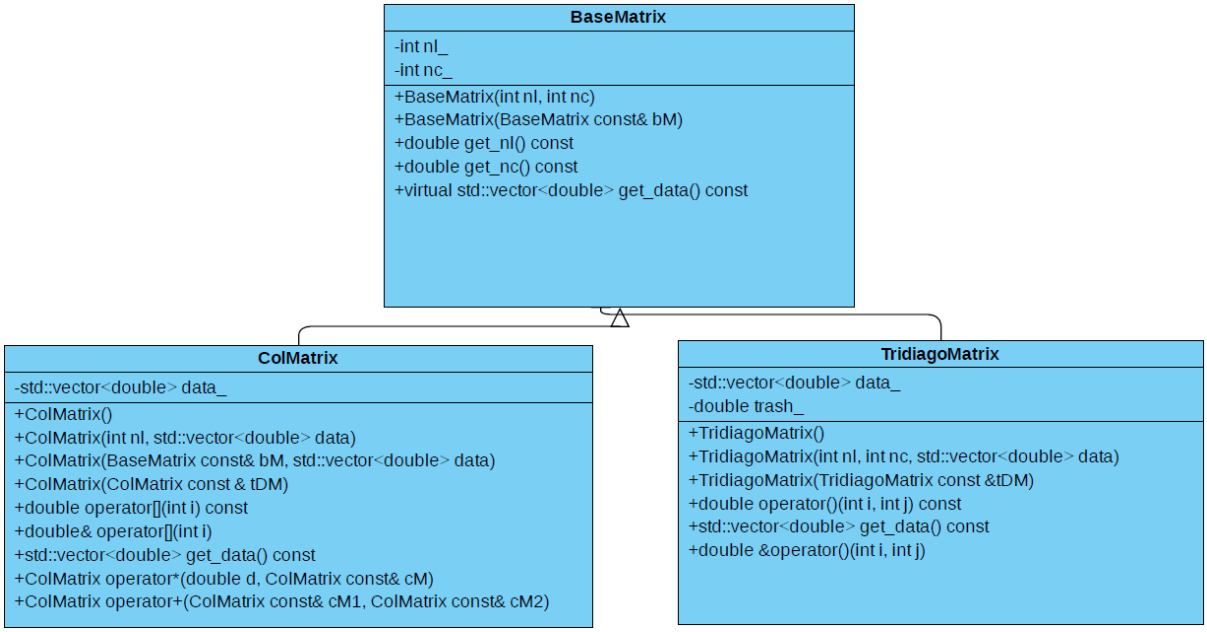


Figure 10: Matrices

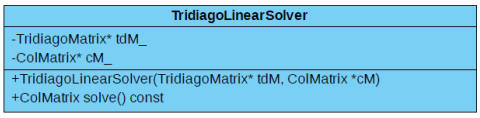


Figure 11: Solveur pour matrices sytèmes linéaires tridiagonaux