Équation de Black et Scholes

Projet PAP - ENSIIE S3

MIELLE David, DESAN Nicolas

Décembre 2022

Table des matières

1 Équation complète

1.1 Discrétisation de l'espace-temps

1.1.1 Temps

L'intervalle $[0; T]$ est séparé en $M = 1000$ points. Ainsi, on note $\Delta t = \dfrac{T}{M}$, et on se déplace d'un point t_n à t_{n+1} par la formule

$$\forall n \in [\![0,\ M-1]\!], \quad t_{n+1} = t_n + \Delta t, \quad \text{avec } t_0 = 0. \tag{1}$$

1.1.2 Espace

L'intervalle spatial $[0; L]$ est séparé en $N = 1000$ points également. On note $\Delta s = \dfrac{L}{N}$, et on se déplace d'un point s_j à s_{j+1} par la formule

$$\forall j \in [\![0,\ N-1]\!], \quad s_{j+1} = s_j + \Delta s, \quad \text{avec } s_0 = 0. \tag{2}$$

1.2 Discrétisation de l'équation

1.2.1 Schéma explicite

La méthode explicite consiste à utiliser un schéma décentré à droite pour le calcul de la dérivée temporelle.
L'équation de Black et Scholes :

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

se réécrit :

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

On pose :

$$\begin{cases} a(S,\tau) & = \sigma^2 S^2/2 \\ b(S,\tau) & = rS \\ c(S,\tau) & = -r \end{cases}$$

avec :

$$\begin{cases} S_n & = n\Delta S \\ a_{n,j} & = a(S_n, \tau_j) \\ b_{n,j} & = b(S_n, \tau_j) \\ c_{n,j} & = c(S_n, \tau_j) \end{cases}$$

On a :

$$\begin{cases} \dfrac{\partial C}{\partial \tau} & = \dfrac{C_{n,j+1} - C_{n,j}}{\Delta t} + O(\Delta t) \\ \dfrac{\partial^2 C}{\partial S^2} & = \dfrac{C_{n+1,j} - 2C_{n,j} + C_{n-1,j}}{\Delta S^2} + O(\Delta S^2) \\ \dfrac{\partial C}{\partial S} & = \dfrac{C_{n+1,j} - C_{n-1,j}}{2\Delta S} + O(\Delta S^2) \end{cases}$$

Ce qui donne :

$$\begin{aligned} C_{n,j+1} = & \left(\nu_1 a_{n,j} - \frac{1}{2}\nu_2 b_{n,j}\right) C_{n-1,j} \\ & + \left(1 - 2\nu_1 a_{n,j} + \Delta t c_{n,j}\right) C_{n,j} \\ & + \left(\nu_1 a_{n,j} + \frac{1}{2}\nu_2 b_{n,j}\right) C_{n+1,j} \\ & + O(\Delta t^2, \Delta t \Delta S^2) \\ & \text{où } \nu_1 = \frac{\Delta t}{\Delta S^2} \text{ et } \nu_2 = \frac{\Delta t}{\Delta S} \end{aligned}$$

On obtient :

$$\boxed{\begin{aligned} C_{n,j+1} = & \frac{1}{2}(\sigma^2 n^2 - rn)\Delta t C_{n-1,j} + \left[1 - (\sigma^2 n^2 + r)\Delta t\right] C_{n,j} \\ & + \frac{1}{2}(\sigma^2 n^2 + rn)\Delta t C_{n+1,j} + O(\Delta t^2, \Delta t \Delta S^2) \end{aligned}}$$

Cette équation est définie pour $n = 1, ..., N-1$ car $C_{-1,j}$ et $C_{N+1,j}$ ne sont pas définis. Donc, il y a $N-1$ équations pour $N+1$ inconnues. Les deux équations restantes proviennent des conditions limites pour $n = 0$ et $n = N$ qui sont traitées séparément.

1.2.2 Schéma implicite

La méthode implicite consiste à utiliser un schéma décentré à gauche pour le calcul de la dérivée temporelle :

$$\begin{cases} \dfrac{\partial C}{\partial \tau} & = \dfrac{C_{n,j+1} - C_{n,j}}{\Delta t} + O(\Delta t) \\ \dfrac{\partial^2 C}{\partial S^2} & = \dfrac{C_{n+1,j+1} - 2C_{n,j+1} + C_{n-1,j+1}}{\Delta S^2} + O(\Delta S^2) \\ \dfrac{\partial C}{\partial S} & = \dfrac{C_{n+1,j+1} - C_{n-1,j+1}}{2\Delta S} + O(\Delta S^2) \end{cases}$$

Ce qui donne :

$$\boxed{\begin{aligned} C_{n,j} = \frac{1}{2}(rn - \sigma^2 n^2)\Delta t C_{n-1,j+1} + \left[1 + (\sigma^2 n^2 + r)\Delta t\right] C_{n,j+1} \\ - \frac{1}{2}(\sigma^2 n^2 + rn)\Delta t C_{n+1,j+1} + O(\Delta t^2, \Delta t \Delta S^2) \end{aligned}}$$

Cette équation est définie pour $n = 1, ..., N-1$ car $C_{-1,j+1}$ et $C_{N+1,j+1}$ ne sont pas définis. Donc, il y a $N-1$ équations pour $N+1$ inconnues. Les deux équations restantes proviennent des conditions limites pour $n = 0$ et $n = N$ qui sont traitées séparément.

1.2.3 Schéma de Crank-Nicholson

La méthode de Crank-Nicholson consiste à utiliser un schéma centré en espace. Elle revient à faire la moyenne du schéma explicite et du schéma implicite.

2 Code développé

2.1 Création des classes

2.1.1 Matrices

Nous avons décidé d'exploiter une classe de matrices `Matrix` pour résoudre l'équation de Black-Scholes. Cela sera fait via la méthode du pivot de Gauss.
Elle contient comme attributs privés un tableau `data_` qui contient les données, et deux tailles `width_` et `height_` qui représentent la largeur et la hauteur de la matrice respectivement.
Nous nous sommes rendus compte que la gestion de la mémoire était catastrophique. Nous gérions manuellement la mémoire, (allocation et libération), mais cela nous a pourtant posé beaucoup de soucis et d'erreurs, relevés soit par des `segmentation fault`, soit par Valgrind. Dans tous les cas, les résultats étaient inconsistants, aléatoires et ce fut assez pénible. Nous nous sommes alors tournés vers `std::vector`, qui est une librairie gérant automatiquement la mémoire, dans le but d'obtenir des résultats et un code compilant parfaitement.
On inclut la librairie `cstddef` qui fait bien partie de la librairie standard.
Au bout du compte, la classe `Matrix` contient les constructeurs suivants :

- `Matrix()` : le constructeur par défaut ;
- `Matrix(std::size_t width, std::size_t height)` : constructeur créant une matrice de `width*height` 0 ;
- `Matrix(double** data, std::size_t width, std::size_t height)` un constructeur qui initialise les éléments d'une matrice aux valeurs contenues dans `data_` ;
- `Matrix(const Matrix& other)` un constructeur de copie ;
- ∼`Matrix()` le destructeur.

De plus, elle contient aussi les accesseurs et méthodes suivants :

- `std::size_t getWidth()` : accesseur renvoyant la largeur de la matrice ;
- `std::size_t getHeight()` : accesseur renvoyant la hauteur de la matrice ;
- `double& operator()(const std::size_t i, const std::size_t j)` l'accesseur/modifieur renvoyant ou modifiant l'élément à l'emplacement `(i, j)` de la matrice courante ;
- `Matrix& operator=(const Matrix& other)` l'opérateur surchargeant = et copiant le contenu d'une matrice vers une autre ;
- `friend std::ostream& operator«(std::ostream& os, const Matrix& m)` l'opérateur affichant la matrice `m`.
- `int pivot(std::size_t a)` : méthode renvoyant l'indice de la meilleure ligne

pivot partant de la ligne indicée `a` ;
- `Matrix& line_switch(std::size_t l1, std::size_t l2)` : méthode échangeant les lignes `l1` et `l2` ;
- `Matrix& line_subtract(int l1, int l2, double lambda)` : méthode remplaçant la ligne `l1` par `l1` $-\lambda$`*l2`.

Finalement on crée les méthodes globales sur les matrices suivantes qui fonctionnent sous conditions de tailles adéquates :
- `Matrix operator+(Matrix& m1, Matrix& m2)` : renvoie `m1 + m2` ;
- `Matrix operator-(Matrix& m1, Matrix& m2)` : renvoie `m1 - m2` ;
- `Matrix operator*(Matrix& m1, Matrix& m2)` : renvoie `m1 * m2` ;
- `Matrix operator*(Matrix& m, const double k)` : renvoie `m * k` ;
- `void tri(Matrix& A, Matrix& Y)` : triangularise supérieurement `A` et `Y` ;
- `Matrix gaussJordan(Matrix& A, Matrix& Y)` : applique l'algorithme de Gauss-Jordan à `A` et `Y` ;
- `Matrix error(Matrix C, Matrix C_tilde)` : renvoie une matrice contenant, pour chacun de ses éléments, la différence en valeur absolue des éléments correspondants des deux matrices arguments.

2.1.2 Payoffs

Pour pouvoir résoudre l'équation de Black et Scholes, il est nécessaire de posséder une condition terminale et des conditions au bord. Il existe deux types de conditions terminales qui constituent les *payoffs*. Il nous fallait d'abord faire en sorte de pouvoir gérer ces deux *payoffs*, ce qui a été fait avec l'aide de plusieurs classes abstraites. On retrouve la structure utilisée sur le diagramme de classes suivant :

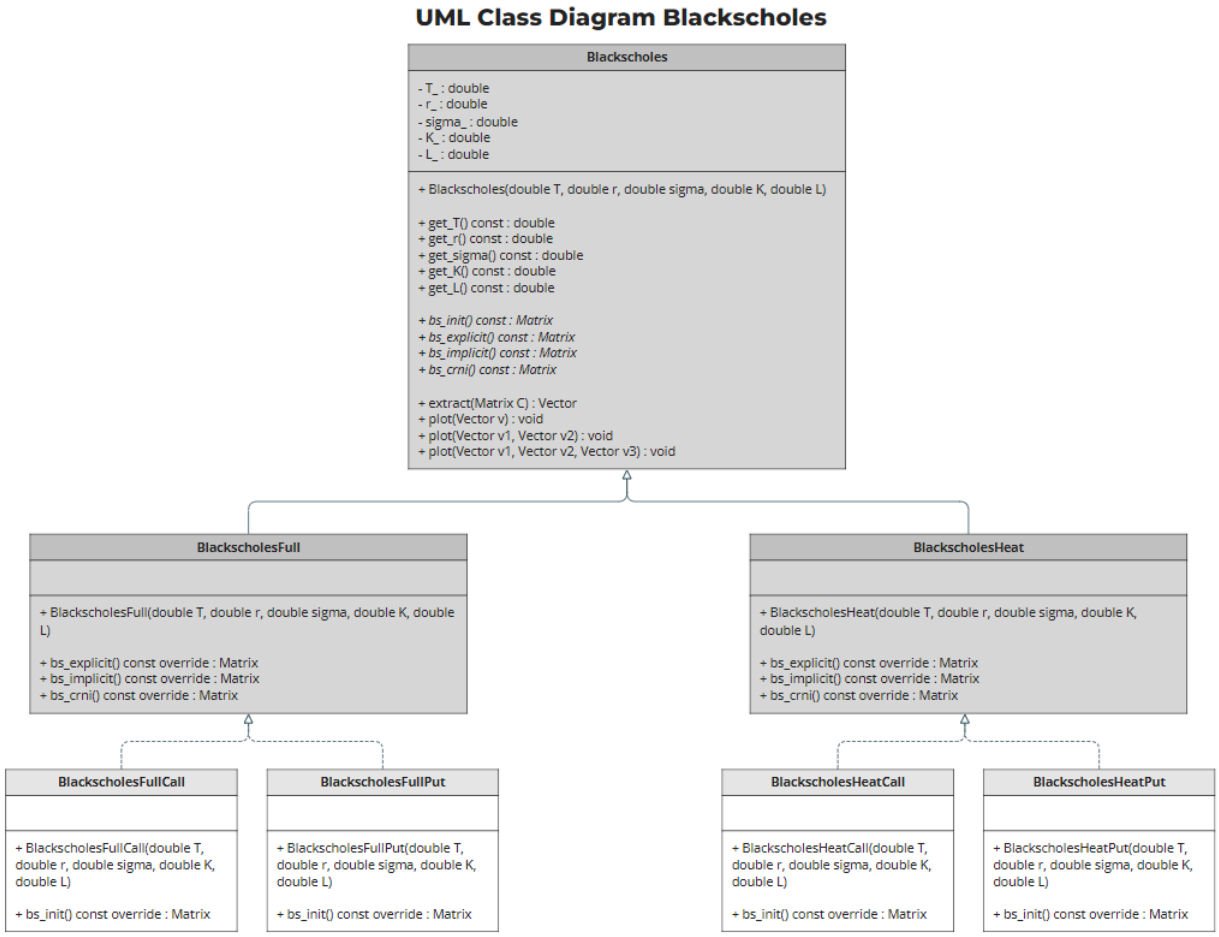

FIGURE 1 – *Architecture des classes utilisées*

On utilise trois classes abstraites pour structurer notre code (représentées en gris sur le diagramme). La première classe générale abstraite `Blackscholes` contient les attributs privés nécessaires à la résolution de l'équation. On rajoute donc des accesseurs qui retournent la valeur de la variable désirée. On rend la classe abstraite en ajoutant une fonction virtuelle pure *a minima*. Ici, il y en a en fait plusieurs, qui sont écrites en italique. `bs_init()` est une méthode qui vient initialiser la matrice résultat avec les conditions au bord. `bs_explicit()`, `bs_implicit()` ou encore `b_crni()` viennent résoudre l'équation discrétisée selon un point de vue explicite, implicite ou bien avec la méthode de Cranck-Nicholson. La méthode `extract()` renvoie théoriquement le vecteur colonne $C(0, j),\ j \in [\![0, M-1]\!]$. Cependant, lors de nos résolutions, nous avons rencontré des soucis pour extraire cette colonne, (ou même des colonnes proches) de

la matrice résultat `C`. Ainsi, nous avons décidé de renvoyer la colonne `N-1`, qui montre bien que l'algorithme a fonctionné. C'est une déviation par rapport au sujet, nous n'avons pas réussi à renvoyer la colonne 0.
Enfin, les différentes méthodes `plot()` renvoient un affichage `SDL` comprenant successivement une, deux ou trois courbes approchant $C(N-1,j),\ j \in [\![0, M-1]\!]$.

2.2 Opérations matricielles sur l'équation complète

2.2.1 Schéma explicite

On recherche, pour $1 \leq j < M$:

$$X = \begin{bmatrix} C_{0,j-1} \\ \vdots \\ C_{N-1,j-1} \end{bmatrix}$$

tel que $AX = Y$, avec :

$$Y = \begin{bmatrix} C_{0,j} \\ \vdots \\ C_{N-1,j} \end{bmatrix}$$

et :

$$A = \begin{bmatrix} L_0 & M_0 & 0 & & \dots & 0 \\ N_1 & L_1 & M_1 & 0 & \dots & 0 \\ 0 & N_2 & L_2 & M_2 & \ddots & 0 \\ \vdots & 0 & \ddots & \ddots & \ddots & 0 \\ & & \ddots & \ddots & \ddots & M_{M-1} \\ 0 & \dots & & 0 & N_{M-1} & L_{M-1} \end{bmatrix}$$

où :

$$L_n = 1 - (\sigma^2 n^2 + r)\Delta t$$
$$M_n = \frac{1}{2}(\sigma^2 n^2 + rn)\Delta t$$
$$N_n = \frac{1}{2}(\sigma^2 n^2 - rn)\Delta t$$

Ainsi, le but est de trouver l'ensemble des valeurs de $C(t,s)$ en réalisant la récurrence de $AX = Y$. Etant donné que les conditions initiales nous donnent la dernière colonne

de la matrice $(C_{i,j})_{(i,j)}$, on cherche à obtenir X en fonction de Y. Pour cela on a appliqué l'algorithme de Gauss-Jordan.

2.2.2 Schéma implicite

On recherche, pour $1 \leq j < M$:

$$X = \begin{bmatrix} C_{0,j} \\ \vdots \\ C_{N-1,j} \end{bmatrix}$$

tel que $AX = Y$, avec :

$$Y = \begin{bmatrix} C_{0,j-1} \\ \vdots \\ C_{N-1,j-1} \end{bmatrix}$$

et :

$$A = \begin{bmatrix} L_0 & M_0 & 0 & & \dots & 0 \\ N_1 & L_1 & M_1 & 0 & \dots & 0 \\ 0 & N_2 & L_2 & M_2 & \ddots & 0 \\ \vdots & 0 & \ddots & \ddots & \ddots & 0 \\ & & \ddots & \ddots & \ddots & M_{M-1} \\ 0 & \dots & & 0 & N_{M-1} & L_{M-1} \end{bmatrix}$$

où :

$$\begin{aligned} L_n &= 1 + (\sigma^2 n^2 + r)\Delta t \\ M_n &= -\frac{1}{2}(\sigma^2 n^2 + rn)\Delta t \\ N_n &= \frac{1}{2}(rn - \sigma^2 n^2)\Delta t \end{aligned}$$

Ainsi, le but est de trouver l'ensemble des valeurs de $C(t, s)$ en réalisant la récurrence de $AX = Y$. Etant donné que les conditions initiales nous donnent la dernière colonne de la matrice $(C_{i,j})_{(i,j)}$, on cherche à obtenir Y en fonction de X. Pour cela on a tout simplement appliqué le produit matriciel AX.

2.3 Opérations matricielles sur l'équation complète

Pour passer de l'équation complète de Black-Scholes à sa forme réduite dite "d'équation de la chaleur", on effectue les changements de variables suivants :

$$C(t,s) = \exp(\beta x + \gamma\tau)U(x,\tau),$$

avec $\beta = -\frac{1}{2}\big(2r/\sigma^2 - 1\big)$, $\gamma = -\frac{1}{4}\big(2r/\sigma^2 + 1\big)^2$, $s = e^x$ et $t = T - \frac{2\tau}{\sigma^2}$.
Après développement des calculs, on obtient bien que :

$$\boxed{\frac{\partial U}{\partial \tau} = \frac{\partial^2 U}{\partial x^2}}$$

On pose aussi les variations $\Delta x = \ln(L/M)$ et $\Delta\tau = -\frac{\sigma^2}{2}\Delta t$. A présent, on discrétise l'équation avec les schémas explicite et implicite.

2.3.1 Schéma explicite

Le schéma s'écrit comme suit :

$$\frac{U_{n,j} - U_{n,j-1}}{\Delta\tau} = \frac{U_{n+1,j-1} - 2U_{n,j-1} + U_{n-1,j-1}}{(\Delta x)^2}$$

Cela implique donc que :

$$\boxed{\mathrm{U}_{n,j} = \frac{\Delta\tau}{(\Delta x)^2}\big(U_{n+1,j-1} - 2U_{n,j-1} + U_{n-1,j-1}\big) + U_{n,j-1}}$$

Dans ce cas, on obtient le schéma de récurrence $AX = Y$ où, pour tout $j \in [\![1, N]\!]$:

$$A = \begin{bmatrix} 1-2\alpha & \alpha & 0 & & \dots & 0 \\ \alpha & 1-2\alpha & \alpha & 0 & \dots & \\ 0 & \alpha & 1-2\alpha & \alpha & \ddots & \vdots \\ \vdots & 0 & \ddots & \ddots & \ddots & 0 \\ & & \ddots & \ddots & \ddots & \alpha \\ 0 & \dots & & 0 & \alpha & 1-2\alpha \end{bmatrix},$$

$$X = \begin{bmatrix} U_{0,j-1} \\ \vdots \\ U_{N-1,j-1} \end{bmatrix}$$

et

$$Y = \begin{bmatrix} U_{0,j} \\ \vdots \\ U_{N-1,j} \end{bmatrix}.$$

Ainsi, le but est de trouver l'ensemble des valeurs de $C(t,s)$ en réalisant la récurrence de $AX = Y$. Etant donné que les conditions initiales nous donnent la première colonne de la matrice $(U_{i,j})_{(i,j)}$, on cherche à obtenir Y en fonction de X. Pour cela on a simplement appliqué le produit matriciel AX.

2.3.2 Schéma implicite

Le schéma s'écrit comme suit :

$$\frac{U_{n,j} - U_{n,j-1}}{\Delta\tau} = \frac{U_{n+1,j} - 2U_{n,j} + U_{n-1,j}}{(\Delta x)^2}$$

Cela implique donc que :

$$\boxed{\mathrm{U}_{n,j-1} = \frac{\Delta\tau}{(\Delta x)^2}\big(- U_{n+1,j} + 2U_{n,j} - U_{n-1,j}\big) + U_{n,j}}$$

Dans ce cas, on obtient le schéma de récurrence $AX = Y$ où, pour tout $j \in [\![1, N]\!]$:

$$A = \begin{bmatrix} 1+2\alpha & -\alpha & 0 & & \dots & 0 \\ -\alpha & 1+2\alpha & -\alpha & 0 & \dots & \\ 0 & -\alpha & 1+2\alpha & -\alpha & \ddots & \vdots \\ \vdots & 0 & \ddots & \ddots & \ddots & 0 \\ & & \ddots & \ddots & \ddots & -\alpha \\ 0 & \dots & & 0 & -\alpha & 1+2\alpha \end{bmatrix},$$

$$X = \begin{bmatrix} U_{0,j} \\ \vdots \\ U_{N-1,j} \end{bmatrix}$$

et

$$Y = \begin{bmatrix} U_{0,j-1} \\ \vdots \\ U_{N-1,j-1} \end{bmatrix}.$$

Ainsi, le but est de trouver l'ensemble des valeurs de $C(t,s)$ en réalisant la récurrence de $AX = Y$. Etant donné que les conditions initiales nous donnent la première colonne de la matrice $(U_{i,j})_{(i,j)}$, on cherche à obtenir X en fonction de Y. Pour cela on a appliqué l'algorithme de Gauss-Jordan.

2.3.3 Détermination des conditions au bord

Les conditions au bord pour l'équation complète s'écrivent

$$C(T, s) = \max(0, K - s),$$

si le *payoff* est un *Call*, et

$$C(T, s) = \max(0, s - K)$$

si c'est un *Put*.
Or, on a que

$$t = T \iff \tau = 0$$

et

$$x = \ln(S).$$

Ainsi, avec

$$U(x, \tau) = \exp(-\beta x - \gamma\tau)C(t, s) = \exp(-\beta x - \gamma\tau)C\big(T - \frac{2\tau}{\sigma^2}, e^x\big),$$

on obtient les conditions au bord suivantes :

$$\boxed{\text{U(0, x)} = \max(0, \exp(-\beta x)(K - s))}$$

pour un *Call*, et

$$\boxed{\text{C(T, s)} = \max(0, \exp(-\beta x)(s - K))}$$

pour un *Put*.
On a fait le choix de ne pas considérer les autres conditions au bord étant donné que puisque $x = \ln(s) \xrightarrow[s\to 0]{} -\infty$.

3 Tracé des courbes

3.1 Méthode de tracé et SDL

Pour tracer les courbes approchées représentatives des fonctions $C(M-1,j),\ j \in [\![0, N-1]\!]$, nous traçons de petits segments reliant des points appartenant à la courbe au moyen d'un vecteur contenant les valeurs de $C(M-1,j),\ j \in [\![0, N-1]\!]$. Le fichier `plot.h` contient notamment des fonctions permettant de dessiner un repère orthonormé et d'y visualiser des courbes à partir d'un vecteur dans une fenêtre SDL par utilisation de `renderer`. En utilisant une des fonctions `plot()` présentes dans `Blackscholes.h`, l'utilisateur peut afficher une, deux ou trois courbes superposées. La méthode `extract(Matrix C)` renvoie la colonne `N - 1`de la matrice `C`.

3.2 Affichage des résultats, équation complète

3.2.1 Call, méthode explicite

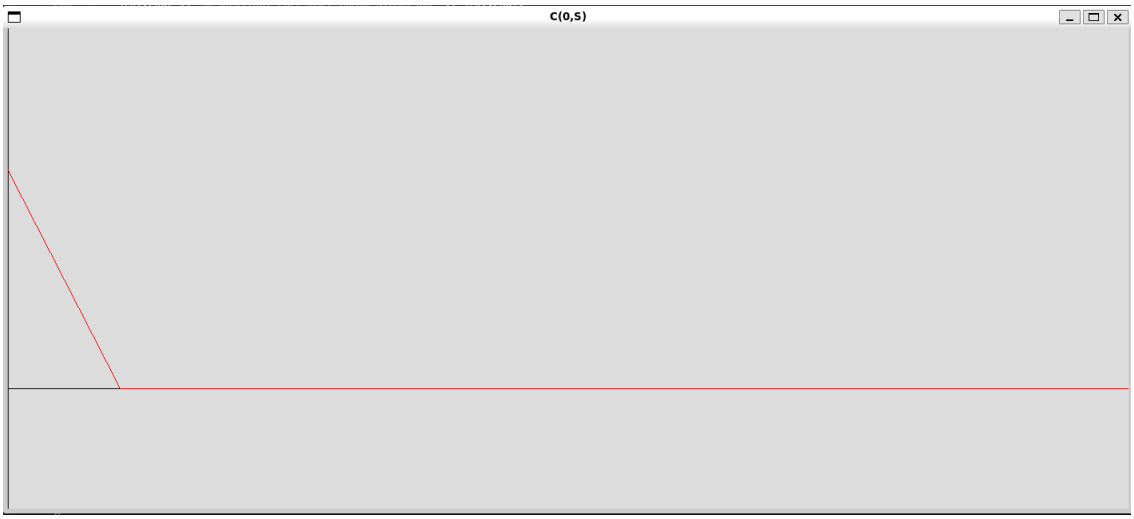

FIGURE 2 – Call schéma explicite

3.2.2 Call, méthode implicite

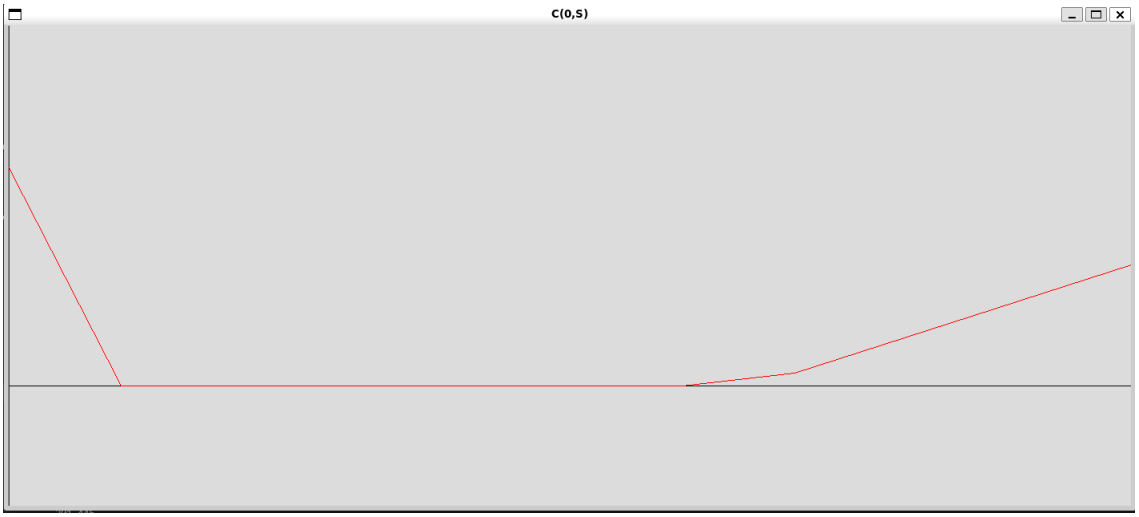

FIGURE 3 – Call schéma implicite

3.2.3 Call, méthode de Cranck-Nicholson

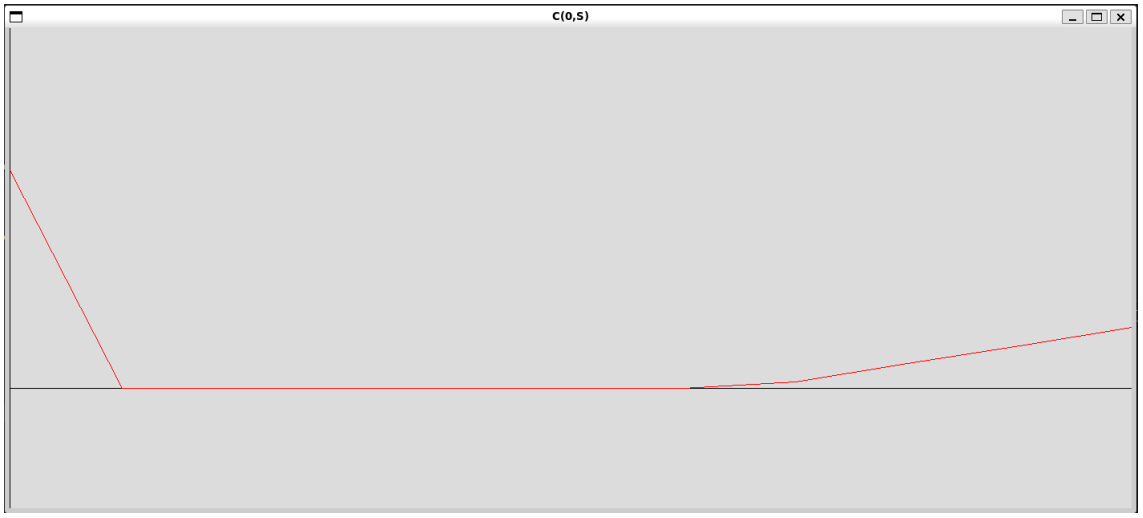

FIGURE 4 – Call schéma de Crank-Nicholson

3.2.4 Call, trois méthodes

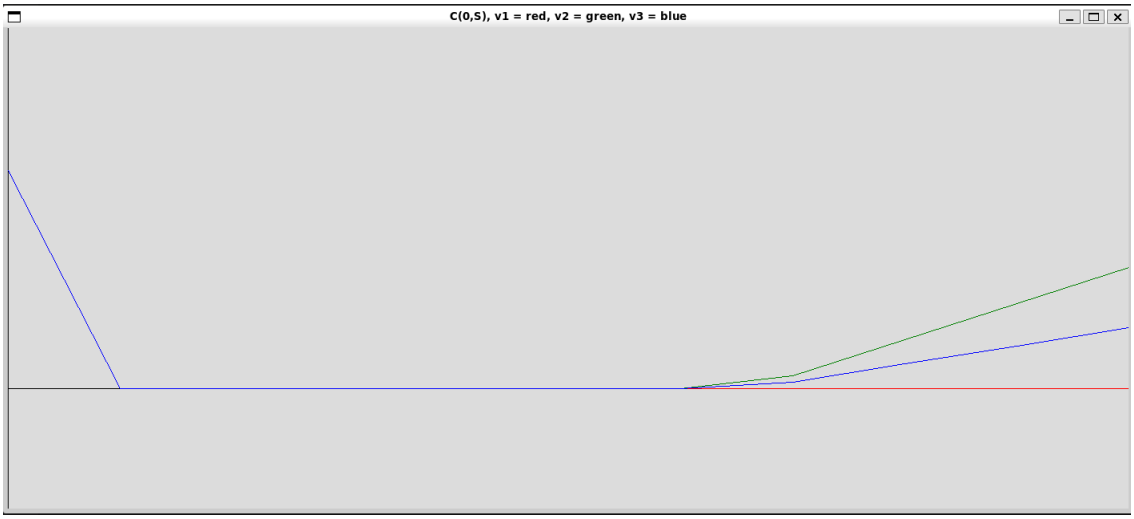

FIGURE 5 – Call, comparaison trois méthodes

3.2.5 Put, méthode explicite

FIGURE 6 – Put schéma explicite

3.2.6 Put, méthode implicite

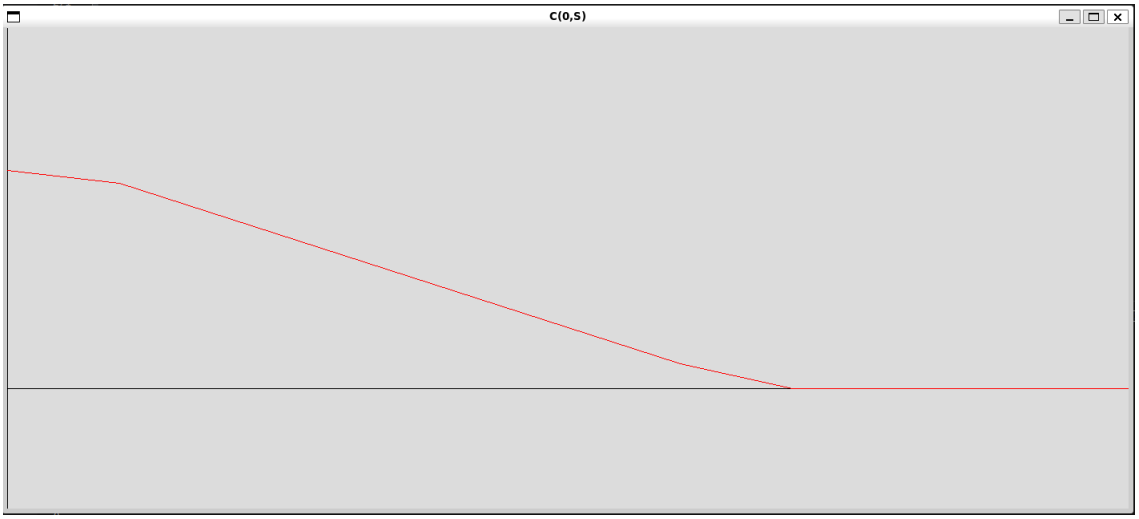

FIGURE 7 – Put schéma implicite

3.2.7 Put, méthode de Cranck-Nicholson

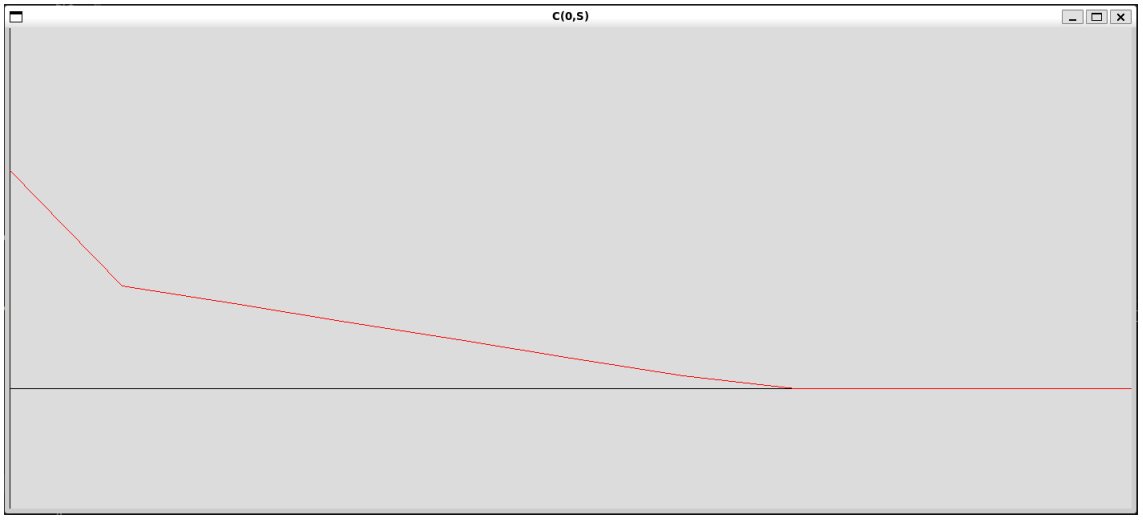

FIGURE 8 – Put schéma de Crank-Nicholson

3.2.8 Put, trois méthodes

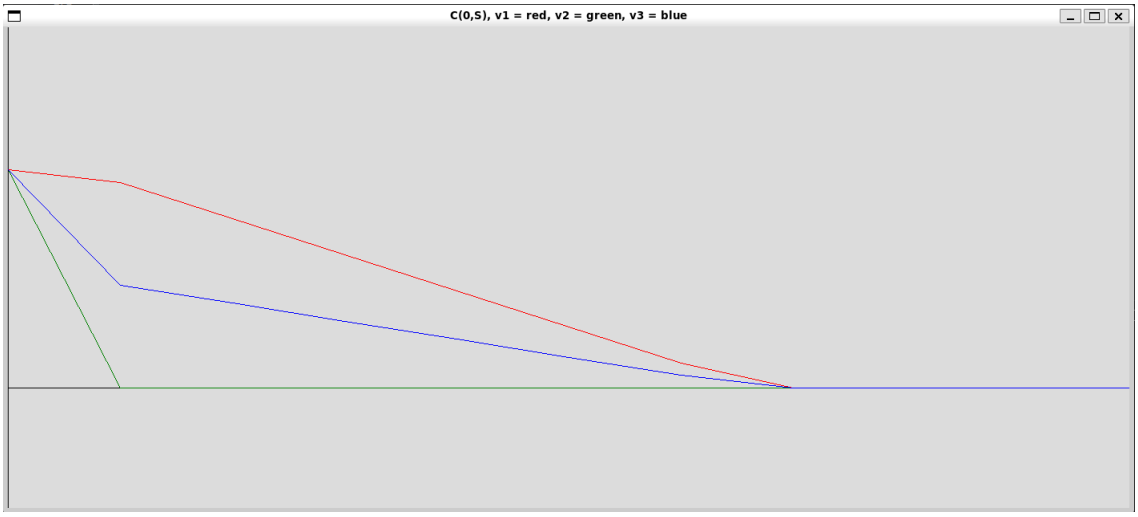

FIGURE 9 – Put, comparaison trois méthodes

3.3 Affichage des résultats, équation réduite

3.3.1 Call, méthode explicite

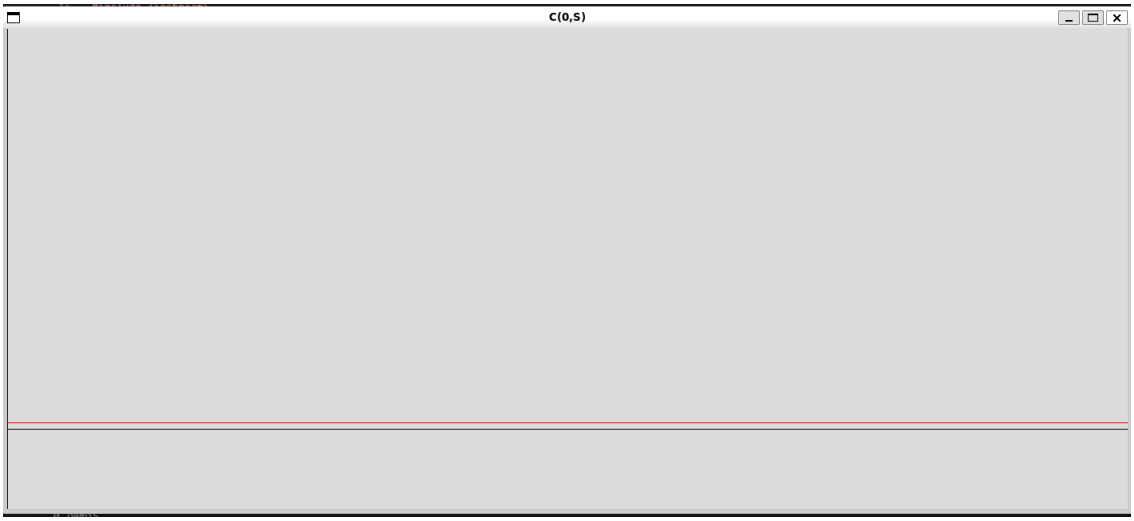

FIGURE 10 – Call schéma explicite

3.3.2 Call, méthode implicite

```
on a finalement la matrice C d'un HEAT CALL IMPLICITE :

99 0 0 0 0 0 0 0 0 0
7.54658e+15 7.54723e+15 7.54723e+15 7.54723e+15 7.54723e+15 7.54723e+15 7.54723e+15 7.54723e+15 7.54723e+15 7.54723e+15
0 -3.26236e+11 -3.26236e+11 -3.26236e+11 -3.26236e+11 -3.26236e+11 -3.26236e+11 -3.26236e+11 -3.26236e+11 -3.26236e+11
0 1.41018e+07 1.41018e+07 1.41018e+07 1.41018e+07 1.41018e+07 1.41018e+07 1.41018e+07 1.41018e+07 1.41018e+07
0 -609.564 -609.564 -609.564 -609.564 -609.564 -609.564 -609.564 -609.564 -609.564
0 0.0263489 0.0263489 0.0263489 0.0263489 0.0263489 0.0263489 0.0263489 0.0263489 0.0263489
0 -1.13896e-06 -1.13896e-06 -1.13896e-06 -1.13896e-06 -1.13896e-06 -1.13896e-06 -1.13896e-06 -1.13896e-06 -1.13896e-06
0 4.92324e-11 4.92324e-11 4.92324e-11 4.92324e-11 4.92324e-11 4.92324e-11 4.92324e-11 4.92324e-11 4.92324e-11
0 0 0 0 0 0 0 0 0 0
0 -4.92324e-11 -4.92324e-11 -4.92324e-11 -4.92324e-11 -4.92324e-11 -4.92324e-11 -4.92324e-11 -4.92324e-11 -4.92324e-11
0 0 0 0 0 0 0 0 0 0
```

FIGURE 11 – Matrice C en petite dimension (20*20) renvoyée par la méthode implicite

Ici on obtient des valeurs de $U(\tau, x)$ sans les valeurs initiales pour $x = 0$ car elles ne sont pas atteignables en $-\infty$. Ainsi, par défaut la première ligne contient des 0.

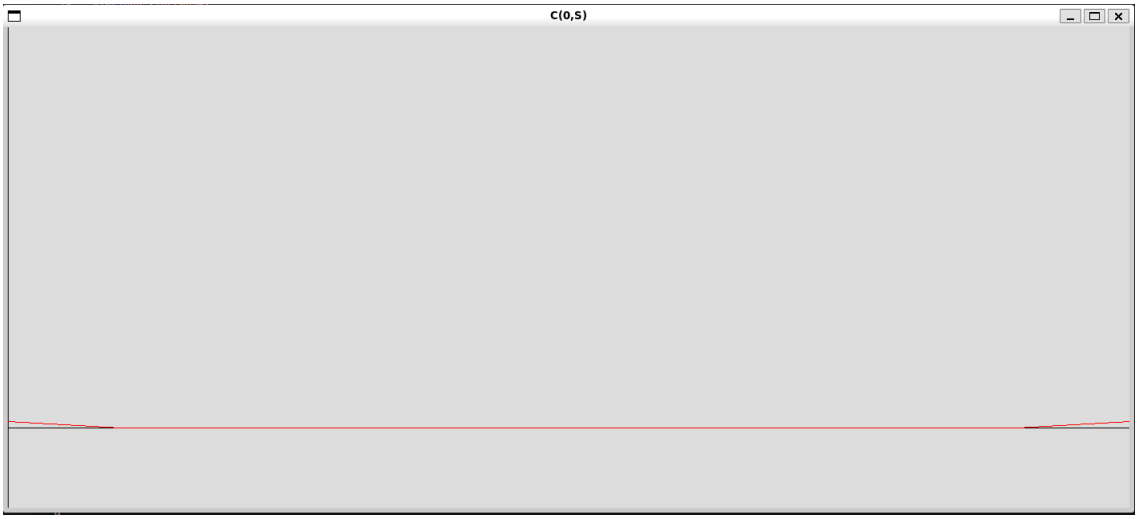

FIGURE 12 – Call schéma implicite

3.3.3 Call, méthode de Cranck-Nicholson

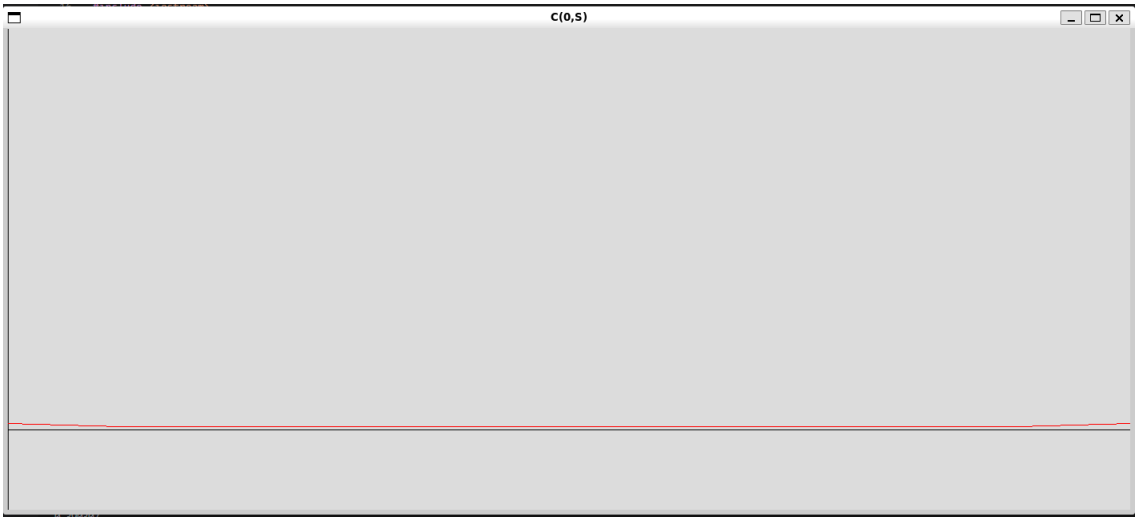

FIGURE 13 – Call schéma de Crank-Nicholson

3.3.4 Put, méthode explicite

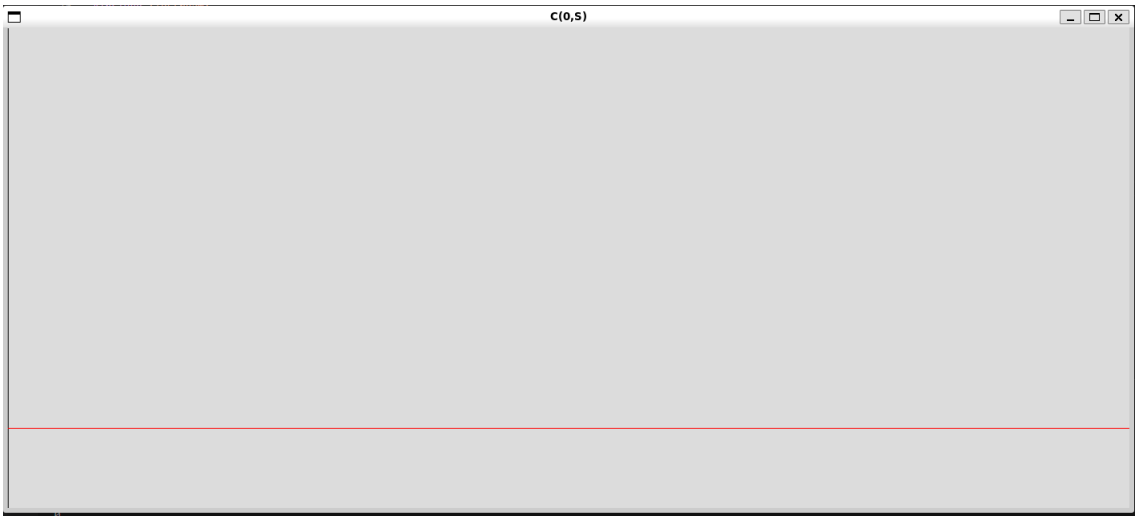

FIGURE 14 – Put schéma explicite

3.3.5 Put, méthode implicite

```
on a finalement la matrice C d'un HEAT PUT IMPLICITE :

0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
9.29809e+30 0 0 0 0 0 0 0 0 0 0
3.37061e+46 8.18348e+149 8.18348e+149 8.18348e+149 8.18348e+149 8.18348e+149 8.18348e+149 8.18348e+149 8.18348e+149 8.18348e+149 8.18348e+149
1.09405e+62 -4.9948e+145 -4.9948e+145 -4.9948e+145 -4.9948e+145 -4.9948e+145 -4.9948e+145 -4.9948e+145 -4.9948e+145 -4.9948e+145 -4.9948e+145
3.53887e+77 1.52429e+141 1.52429e+141 1.52429e+141 1.52429e+141 1.52429e+141 1.52429e+141 1.52429e+141 1.52429e+141 1.52429e+141 1.52429e+141
1.14456e+93 -9.30354e+136 -9.30354e+136 -9.30354e+136 -9.30354e+136 -9.30354e+136 -9.30354e+136 -9.30354e+136 -9.30354e+136 -9.30354e+136 -9.30354e+136
3.70179e+108 -3.87254e+139 -3.87254e+139 -3.87254e+139 -3.87254e+139 -3.87254e+139 -3.87254e+139 -3.87254e+139 -3.87254e+139 -3.87254e+139 -3.87254e+139
1.19725e+124 8.95885e+143 8.95885e+143 8.95885e+143 8.95885e+143 8.95885e+143 8.95885e+143 8.95885e+143 8.95885e+143 8.95885e+143 8.95885e+143
3.87221e+139 -2.07257e+148 -2.07257e+148 -2.07257e+148 -2.07257e+148 -2.07257e+148 -2.07257e+148 -2.07257e+148 -2.07257e+148 -2.07257e+148 -2.07257e+148
1.25237e+155 0 0 0 0 0 0 0 0 0 0
```

FIGURE 15 – Put schéma implicite

Ici on obtient des valeurs de $U(\tau, x)$ sans les valeurs initiales pour $x = 0$ car elles ne sont pas atteignables en $-\infty$. Ainsi, par défaut la première ligne contient des 0.

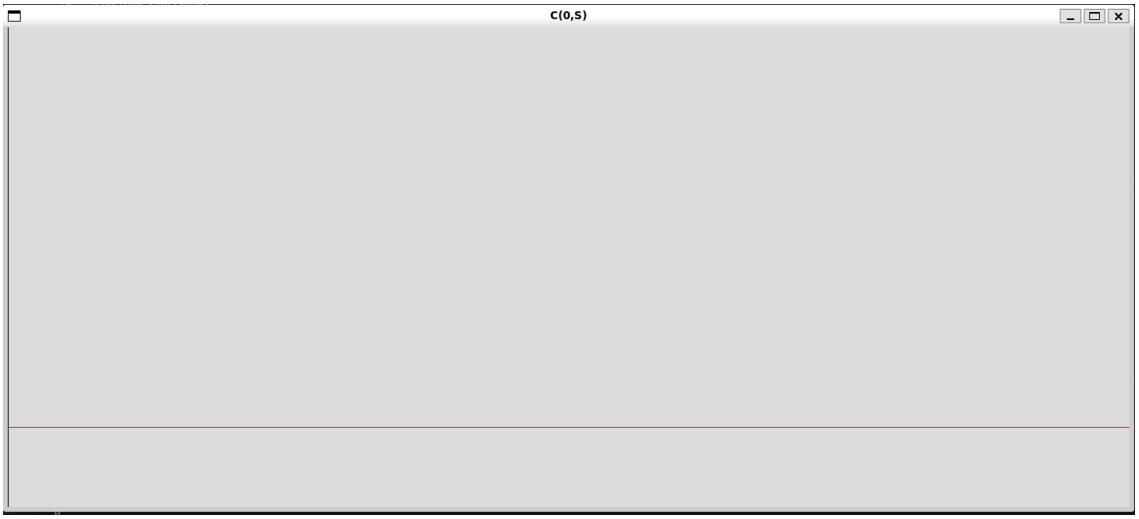

FIGURE 16 – Matrice C en petite dimension (20*20) renvoyée par la méthode implicite

3.3.6 Put, méthode de Cranck-Nicholson

La méthode de Cranck-Nicholson s'avérant être une moyenne des schémas explicite et implicite, au vu du caractère erroné de nos résultats, il est légitime de ne pas obtenir de courbe correcte ici non plus.

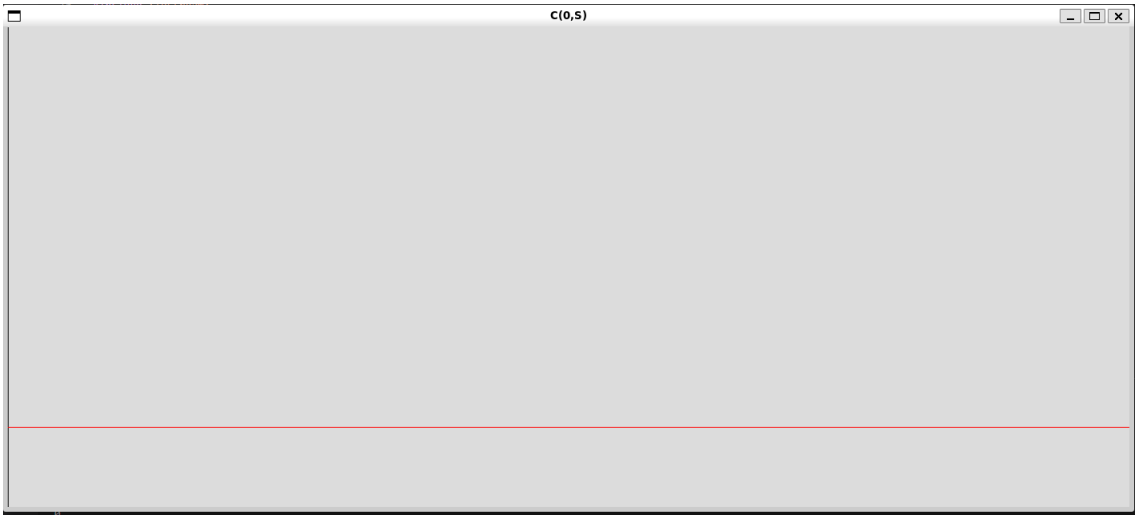

FIGURE 17 – Call schéma de Crank-Nicholson

3.4 Comparaison des résultats des deux équations

On devait afficher dans une fenêtre les deux courbes approchant les solutions des deux équations. Au vu de nos résultats faux pour la résolution de l'équation réduite notamment à cause des conditions au bord, nous n'avons pas jugé utile d'afficher cela. Nous aurions évidemment utilisé la méthode `plot(Matrix v1, Matrix v2)` pour faire cela. De plus, il fallait afficher dans une autre fenêtre l'erreur entre les deux méthodes. Là encore, cela n'a que peu d'intérêt puisqu'une des deux matrices contient des valeurs aberrantes. On aurait pu, au choix, afficher la différence de chacune des valeurs des matrices, ou bien leur module. Chaque usage est différent, mais afficher leur module a pour avantage de comprendre plus facilement à quel point les valeurs entre les matrices résultats diffèrent, nous trouvons cela plus simple à visualiser, mais ce n'est qu'une histoire de signes.

4 Perspectives d'optimisation

Le code compile avec succès sans erreurs ni warnings (bien que l'exécution soit un peu longue) [1] et chaque fonction développée a été testée.

Les résultats du programme pour des résolutions de l'EDP (schémas explicite, implicite et de Crank-Nicholson) pour des conditions initiales de type Put sont satisfaisants. Cependant, les résultats du programme pour des résolutions de l'EDP pour des conditions initiales de type Call et de l'équation réduite méritent de faire l'objet de nos études et investigations approfondies ultérieurement.
À titre indicatif, nous ajoutons ici certains points qui peuvent être optimisés :

- Exploiter des décompositions matricielles (LU, QR, Cholesky...) pour améliorer les complexités spatiale et temporelle, plutôt que la méthode de Gauss-Jordan ;
- Afficher la graduation des axes du repère lors de l'affichage d'une courbe dans la fenêtre SDL ;
- Permettre à l'utilisateur de zoomer et se déplacer sur la courbe lors de l'affichage dans la fenêtre SDL ;
- Linéariser les matrices (c'est-à-dire que `tableau[i][j]` serait en fait représenté par `tableau[i*height + j]`) pour ne pas à avoir à utiliser des tableaux de tableaux. On économiserait en appels de méthodes, en temps d'accès, en mémoire, en bref tout serait mieux.

1. Le code a été compilé sur Windows avec VScode et WSL, avec `g++` de version `11.2`.

5 Manuel d'utilisation

Ce manuel est plus précisément axé sur une utilisation sur `Linux` avec `g++`.

5.1 Installation

Après avoir récupéré , vous pouvez suivre les étapes d'installation suivantes :

1. Récupérer l'archive nommée `MIELLE_DESAN_projet_bs.tar.xz` ;
2. Ouvrir un terminal et y créer un dossier avec la commande `mkdir nomDossier` ;
3. Se placer dans `nomDossier` et extraire le contenu de l'archive avec la commande :
   `> tar -xzvf MIELLE_DESAN_projet_bs.tar.xz`

Pour l'affichage de courbes, il est nécessaire d'avoir installé la bibliothèque SDL `libsdl2-dev` en entrant par exemple la commande suivante dans un terminal :
`> sudo apt-get install libsdl2-dev`

Ainsi, on pourra bénéficier de l'affichage des résultats.

5.2 Lancement du programme

Nous vous laissons éditer le fichier `main_BS.cpp` pour afficher les courbes de votre choix. Par défaut, la résolution de l'équation complète par Cranck-Nicholson (et donc implicite et explicite) est utilisée pour les deux *payoffs*, tandis que seule la méthode implicite est décommentée pour l'équation réduite.

1. Ouvrir un terminal et se placer dans le dossier `Dossier`
2. Dans le terminal, entrer la commande de compilation suivante :
   `> g++ -g -Wall -Wextra -o projet *.cpp ‘pkg-config -cflags -libs sdl2‘`
3. Dans le terminal, entrer ensuite la commande suivante pour lancer l'exécution :
   `> ./projet`