MSc 203
University Paris Dauphine

C++ Project

Thibaut JEREBITZ
Florian PERIGNE
Valentin ROCHEREAU

C++ Project
MSc 203
University Paris Dauphine

Thibaut JEREBITZ
Florian PERIGNE
Valentin ROCHEREAU

January 25, 2020

Abstract

The goal of the project is to solve the Black Scholes PDE.
In this document, we develop the mathematics needed to implement a C++ code to solve the
PDE.

Outlines

1   Introduction

The price of a European option satitisfies the *Black Scholes equation*

$$0 = \frac{\partial f}{\partial t}(s,t) + \frac{1}{2}\sigma^2 s^2 \frac{\partial^2 f}{\partial s^2}(s,t) + rs\frac{\partial f}{\partial s}(s,t) - rf(s,t)$$

with the final condition $f(s,T) = V(s)$ where $V(s)$ is the payout.

Applying the variable change $x = log(s)$ yields :

$$\frac{\partial f}{\partial t}(x,t) = -\frac{1}{2}\sigma^2 \frac{\partial^2 f}{\partial x^2}(x,t) + (\frac{1}{2}\sigma^2 - r)\frac{\partial f}{\partial x}(x,t) + rf(x,t)$$

with the final condition $f(x,T) = V(x) = V(e^s)$.

A common method to solve this PDE is to use finite differences to approximate the derivatives and then solve the equation on a discrete mesh.

When developping the different formula, we will have to set boundaries conditions for the points on the border of the mesh. We will develop two possible solutions to set these conditions :

- Dirichlet method (part 2)

- Neumann method (part 3)

3

2 Dirichlet

$\forall n \in [\![1, N]\!], \forall i \in [\![0, x_{max}]\!],$

$$f_i^n = f(t_n, x_i) \tag{1}$$

$$\frac{f_i^{n+1} - f_i^n}{dt} = \theta L_i^n + (1 - \theta)L_i^{n+1} \tag{2}$$

Let $A_1^{(1-\theta)}, B_-^{(1-\theta)}, B_+^{(1-\theta)}, A_2^\theta, B_-^\theta$ and B_+^θ take the following values :

$$
\begin{aligned}
A_1^{(1-\theta)} &= \frac{1}{dt} - \frac{\sigma^2}{dx^2}(1 - \theta) - r(1 - \theta) \\
B_-^{(1-\theta)} &= \frac{\sigma^2}{2dx^2}(1 - \theta) - (\frac{\sigma^2}{2} - r)(1 - \theta)\frac{1}{2dx} \\
B_+^{(1-\theta)} &= \frac{\sigma^2}{2dx^2}(1 - \theta) + (\frac{\sigma^2}{2} - r)(1 - \theta)\frac{1}{2dx} \\
A_2^\theta &= \frac{1}{dt} + \frac{\theta\sigma^2}{dx^2} + r\theta \\
B_-^\theta &= -\theta\frac{\sigma^2}{2dx^2} + (\frac{\sigma^2}{2} - r)\theta\frac{1}{2dx} \\
B_+^\theta &= -\theta\frac{\sigma^2}{2dx^2} - (\frac{\sigma^2}{2} - r)\theta\frac{1}{2dx}
\end{aligned}
\tag{3}
$$

Then we obtain :

$$f_i^{n+1}A_1^{(1-\theta)} + f_{i+1}^{n+1}B_-^{(1-\theta)} + f_{i-1}^{n+1}B_+^{(1-\theta)} = f_i^n A_2^\theta - f_{i+1}^n B_-^\theta - f_{i-1}^n B_+^\theta \tag{4}$$

The previous formula gives us the following matrix system :

$$
\begin{pmatrix}
1 & 0 & 0 & 0 & 0 & \cdots & 0 & 0 \\
B_+^{(1-\theta)} & A_1^{(1-\theta)} & B_-^{(1-\theta)} & 0 & 0 & \cdots & 0 & 0 \\
0 & B_+^{(1-\theta)} & A_1^{(1-\theta)} & B_-^{(1-\theta)} & 0 & \cdots & 0 & 0 \\
\vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\
0 & 0 & 0 & 0 & 0 & \cdots & 0 & 1
\end{pmatrix}
\begin{pmatrix}
f_0^{n+1} \\
f_1^{n+1} \\
\vdots \\
f_{x_{max}-1}^{n+1} \\
f_{x_{max}}^{n+1}
\end{pmatrix}
$$
$$
=
\begin{pmatrix}
e^{rdt} & 0 & 0 & 0 & 0 & \cdots & 0 & 0 \\
-B_+^\theta & A_2^\theta & -B_-^\theta & 0 & 0 & \cdots & 0 & 0 \\
0 & -B_+^\theta & A_2^\theta & -B_-^\theta & 0 & \cdots & 0 & 0 \\
\vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\
0 & 0 & 0 & 0 & 0 & \cdots & 0 & e^{rdt}
\end{pmatrix}
\begin{pmatrix}
f_0^n \\
f_1^n \\
\vdots \\
f_{x_{max}-1}^n \\
f_{x_{max}}^n
\end{pmatrix}
\tag{5}
$$

The previous system (equation 5) is equivalent to :

$$Af^{n+1} = Bf^n \tag{6}$$

4

To solve this matrix system, we use Thomas algorithm. This algorithm is used to solve matrix systems with tridiagonal matrix. For such systems, the solution can be obtained in $O(n)$ operations instead of $O(n^3)$ required by Gaussian elimination. To see the complete method : Wikipedia Thomas Algorithm.

3 Neumann

$\forall n \in [\![1, N]\!], \forall i \in [\![0, x_{max}]\!]$,

$$\frac{df}{dx}(t_n, x_0) = k_1 \tag{7}$$

$$\frac{df}{dx}(t_n, x_0) = \frac{f_1^n - f_0^n}{dx} \Leftrightarrow f_0^n = f_1^n - k_1 dx \tag{8}$$

$$\frac{df}{dx}(t_n, x_{x_{max}}) = k_2 \tag{9}$$

$$\frac{df}{dx}(t_n, x_{x_{max}}) = \frac{f_{x_{max}}^n - f_{x_{max}-1}^n}{dx} \Leftrightarrow f_{x_{max}}^n = f_{x_{max}-1}^n + k_2 dx \tag{10}$$

Using the same formula (equation 4):

$$f_i^{n+1} A_1^{(1-\theta)} + f_{i+1}^{n+1} B_-^{(1-\theta)} + f_{i-1}^{n+1} B_+^{(1-\theta)} = f_i^n A_2^\theta - f_{i+1}^n B_-^\theta - f_{i-1}^n B_+^\theta \tag{11}$$

Taking the left part of the equation 11, and using equation 8 we have :

$$f_i^{n+1} A_1^{(1-\theta)} + f_{i+1}^{n+1} B_-^{(1-\theta)} + f_{i-1}^{n+1} B_+^{(1-\theta)} = f_1^n A_2^\theta - f_2^n B_-^\theta - (f_1^n - k_1 dx) B_+^\theta \tag{12}$$

$$= f_1^n (A_2^\theta - B_+^\theta) - f_2^n B_-^\theta + k_1 B_+^\theta dx \tag{13}$$

Taking the right part of the equation 11, and using equation 10 we have :

$$f_i^n A_2^\theta - f_{i+1}^n B_-^\theta - f_{i-1}^n B_+^\theta = f_{x_{max}-1}^n A_2^\theta - f_{x_{max}}^n B_-^\theta - f_{x_{max}-2}^n B_+^\theta \tag{14}$$

$$= f_{x_{max}-1}^n (A_2^\theta - B_-^\theta) - k_2 B_-^\theta dx - f_{x_{max}-2}^n B_+^\theta \tag{15}$$

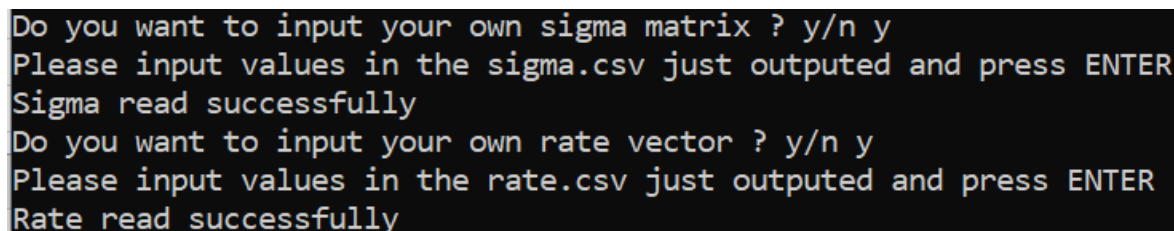The previous formulas give us the following matrix system :

$$
\begin{pmatrix}
-\frac{1}{dx} & \frac{1}{dx} & 0 & 0 & 0 & \cdots & 0 & 0 & 0 & 0 \\
B_+^{(1-\theta)} & A_1^{(1-\theta)} & B_-^{(1-\theta)} & 0 & 0 & \cdots & 0 & 0 & 0 & 0 \\
0 & B_+^{(1-\theta)} & A_1^{(1-\theta)} & B_-^{(1-\theta)} & 0 & \cdots & 0 & 0 & 0 & 0 \\
\vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \vdots \\
0 & 0 & 0 & 0 & 0 & \cdots & 0 & B_+^{(1-\theta)} & A_1^{(1-\theta)} & B_-^{(1-\theta)} \\
0 & 0 & 0 & 0 & 0 & \cdots & 0 & 0 & -\frac{1}{dx} & \frac{1}{dx}
\end{pmatrix}
\begin{pmatrix}
f_0^{n+1} \\
f_1^{n+1} \\
\vdots \\
f_{x_{max}-1}^{n+1} \\
f_{x_{max}}^{n+1}
\end{pmatrix}
$$

$$
=
\begin{pmatrix}
1 & 0 & 0 & 0 & 0 & \cdots & 0 & 0 & 0 & 0 & 0 \\
B_+^\theta dx & A_2^\theta - B_+^\theta & -B_-^\theta & 0 & 0 & \cdots & 0 & 0 & 0 & 0 & 0 \\
0 & -B_+^\theta & A_2^\theta & -B_-^\theta & 0 & \cdots & 0 & 0 & 0 & 0 & 0 \\
\vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \vdots & \vdots \\
0 & 0 & 0 & 0 & 0 & \cdots & 0 & -B_+^\theta & A_2^\theta & -B_-^\theta & 0 \\
0 & 0 & 0 & 0 & 0 & \cdots & 0 & 0 & -B_+^\theta & A_2^\theta - B_-^\theta & -B_-^\theta dx \\
0 & 0 & 0 & 0 & 0 & \cdots & 0 & 0 & 0 & 0 & 1
\end{pmatrix}
\begin{pmatrix}
k_1 \\
f_1^n \\
\vdots \\
f_{x_{max}-1}^n \\
k_2
\end{pmatrix}
$$

$$\tag{16}$$

4 Userguide

4.1 Volatility and Rates inputs

Our program offers the possibility to the user to input the volatility matrix and the rates vector in an excel (csv) file. To do so, the program will ask the user if he wants to use this feature (see figure 1). If the user answers "yes" (y) then the program creates two .csv files to be completed by the user. One for the volatility and the other one for the rates. To input its parameters, the user can open the .csv file in excel, and convert the column A "from text to columns". User will find an empty matrix with the X in lines (all spot prices) and Y in columns (all the times considered). Once saved the user continues to run the program by pressing "ENTER".
User can choose to change the separators in the *.csv* files.

```
Do you want to input your own sigma matrix ? y/n y
Please input values in the sigma.csv just outputed and press ENTER
Sigma read successfully
Do you want to input your own rate vector ? y/n y
Please input values in the rate.csv just outputed and press ENTER
Rate read successfully
```

Figure 1: Options available

4.2 Payoff function

The function payoff takes three arguments : a name for the chosen payoff, a vector (strikes) and a function. All these parameters are optional. If the user doesn't enter anything, the payoff will be $g = S$. The user can enter a name of payoff recognised by the program. The registered payoffs are : "Call", "Put", "Call Spread", "Put Spread", "Straddle", "Strangle". Once the payoff's name chosen, the user has to enter a parameter for the strike(s) of the payoff chosen. If several ones are entered, the program will only take into account the first one for a single strike strategy ("Call", "Put"). For multiple strike strategies ("Call Spread", "Put Spread", "Straddle", "Strangle") the user has to enter the strikes from the smallest to the biggest.
If the user wants to generate a customised payoff, he has to input a different name (one not recognised by the program). Then enter the different strikes and finally the function of the payoff.