Florian Schirrer, Sami Sdiqui, Quentin Lam

User Guide : C++ with Finite difference, the Crank-Nicholson scheme

Introduction:

First of all, we assume the European option satisfies the Black and Scholes equation.

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

In order to solve the equation, we used the finite differences method to approximate the derivatives based on a discrete mesh.

In our particular case, we used the Crank Nicolson Scheme, defined as the averaged scheme by adding the fully explicit and fully implicit schemes together.

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

Such that and Θ equals to ½

$$L_i^n=-\frac{1}{2}\sigma^2\frac{f_{i+1}^n-2f_i^n+f_{i-1}^n}{dx^2}+\left(\frac{1}{2}\sigma^2-r\right)\frac{f_{i+1}^n-f_{i-1}^n}{2dx}+rf_i^n$$

By separating the terms in i+1, i and i-1, we can find the below equation:

$$\forall i \in \{1, \cdots, N-1\}, \quad L_i^n = \left(\frac{\frac{\sigma^2}{2}-r}{2dx} - \frac{\sigma^2}{2dx^2}\right) f_{i+1}^n + \left(r + \frac{\sigma^2}{dx^2}\right) f_i^n - \left(\frac{\frac{\sigma^2}{2}-r}{2dx} + \frac{\sigma^2}{2dx^2}\right) f$$

$$With, \quad a_i = \frac{\sigma^2/2 - r}{2dx} - \frac{\sigma^2}{2dx^2}, \quad b_i = r + \frac{\sigma^2}{dx^2}, \quad c_i = -\left(\frac{\frac{\sigma^2}{2} - r}{2dx} + \frac{\sigma^2}{2dx^2}\right)$$

Under Dirichlet boundaries, we have : $f_0^n = 0$ and $f_N^n = (S-K) \times e^{-r(T-t)}$

$$For \quad i = 1, \quad L_1^n = \left(\frac{\frac{\sigma^2}{2}-r}{2dx} - \frac{\sigma^2}{2dx^2}\right) f_2^n + \left(r + \frac{\sigma^2}{dx^2}\right) f_1^n$$

$$For \quad i = N-1, \quad L_{N-1}^n = \left(r + \frac{\sigma^2}{dx^2}\right) f_{N-1}^n - \left(\frac{\frac{\sigma^2}{2}-r}{2dx} + \frac{\sigma^2}{2dx^2}\right) f_{N-2}^n$$

With,

$$a_1 = \frac{\sigma^2/2 - r}{2dx} - \frac{\sigma^2}{2dx^2}, \quad b_1 = r + \frac{\sigma^2}{dx^2}$$

$$b_{N-1} = r + \frac{\sigma^2}{dx^2}, \quad c_{N-1} = -\left(\frac{\frac{\sigma^2}{2} - r}{2dx} + \frac{\sigma^2}{2dx^2}\right)$$

Under Neumann boundaries, we have : $f_1^n = f_0^n$ and $f_N^n = f_{N-1}^n + dx$

$$For \quad i = 1, \quad L_1^n = \left(\frac{\frac{\sigma^2}{2}-r}{2dx} - \frac{\sigma^2}{2dx^2}\right) f_2^n + \left(r + \frac{\sigma^2}{dx^2} - \left(\frac{\frac{\sigma^2}{2}-r}{2dx} + \frac{\sigma^2}{2dx^2}\right)\right) f_1^n$$

$$For \quad i = N-1, \quad L_{N-1}^n = \left(\frac{\sigma^2/2-r}{2dx} - \frac{\sigma^2}{2dx^2} + r + \frac{\sigma^2}{dx^2}\right) f_{N-1}^n - \left(\frac{\frac{\sigma^2}{2}-r}{2dx} + \frac{\sigma^2}{2dx^2}\right) f_{N-2}^n$$

With,

$$a_1 = \frac{\frac{\sigma^2}{2} - r}{2dx} - \frac{\sigma^2}{2dx^2}, \quad b_1 = r + \frac{\sigma^2}{dx^2} - \left(\frac{\frac{\sigma^2}{2} - r}{2dx} + \frac{\sigma^2}{2dx^2}\right)$$

$$b_{N-1} = \frac{\sigma^2/2 - r}{2dx} - \frac{\sigma^2}{2dx^2} + r + \frac{\sigma^2}{dx^2}, \quad c_{N-1} = -\left(\frac{\frac{\sigma^2}{2} - r}{2dx} + \frac{\sigma^2}{2dx^2}\right)$$

Then, we have : $f_i^n + \theta dt L_i^n = f_i^{n+1} - (1-\theta) dt L_i^{n+1}$

For better use, we transformed the above equations into a matrix problem, such that:

$$\begin{pmatrix} b_1 & a_1 & 0 & \cdots & \cdots & \cdots & 0 \\ c_2 & b_2 & a_2 & \ddots & & & \vdots \\ 0 & \ddots & \ddots & \ddots & \ddots & & \vdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\ \vdots & & \ddots & \ddots & \ddots & \ddots & 0 \\ \vdots & & & \ddots & c_{N-2} & b_{N-2} & a_{N-2} \\ 0 & \cdots & \cdots & \cdots & 0 & c_{N-1} & b_{N-1} \end{pmatrix} * \begin{pmatrix} f_1^n \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ f_{N-1}^n \end{pmatrix} =$$

$$\begin{pmatrix} b_1 & a_1 & 0 & \cdots & \cdots & \cdots & 0 \\ c_2 & b_2 & a_2 & \ddots & & & \vdots \\ 0 & \ddots & \ddots & \ddots & \ddots & & \vdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\ \vdots & & \ddots & \ddots & \ddots & \ddots & 0 \\ \vdots & & & \ddots & c_{N-2} & b_{N-2} & a_{N-2} \\ 0 & \cdots & \cdots & \cdots & 0 & c_{N-1} & b_{N-1} \end{pmatrix} * \begin{pmatrix} f_1^{n+1} \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ f_{N-1}^{n+1} \end{pmatrix} + \begin{pmatrix} 0 \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ 0 \\ B_R \end{pmatrix}$$

If under Neumann, we can define the boundaries increment vector such as:

$$B_R = dx \times dt \times \left(-\frac{r - \sigma^2/2}{2dx} + \frac{\sigma^2}{2dx^2} \right)$$

If under Dirichlet, we can define the boundaries increment vector such as:

$$B_R = -(S - K)e^{-r(T-t)} \times dt \times \left(-\frac{r - \sigma^2/2}{2dx} + \frac{\sigma^2}{2dx^2} \right)$$

Main approach:

- 1) Outline the mathematical formalism using the Black and Scholes equation and the boundary conditions
- 2) Solve the mathematics and establish a tri-diagonal matrix equation. Use of the Crout Algorithm for the resolution.
- 3) Discretize the space and time paths into a mesh grid
- 4) Create class, objects and methods in order to facilitate the implementation.

Boundary conditions:

- Dirichlet. For an European option, boundaries will be from 0 to Smax.
- Neumann

The user has the choice between these two types on each side of the grid. Only the type would be chosen, not the boundary values within each type.

Pool of classes:

- matrix:
  - Used to facilitate matrix operations such as addition or multiplication
- matrix_pde_case1:
  - calculate_parameters. Used to define the space/time mesh grids
  - set_initial_conditions
  - forward_coefficient. Compute the coefficient at position n+1
  - present_coefficient. Compute the coefficient at position n
  - backward_coefficient. Compute the coefficient at position n-1
  - Crout_Algo_Resolution: Algorithm of Crout to resolve the PDE for a given time
  - LU_compute: LU decomposition of the coefficients matrix used in the Crout algorithm
  - Get_price: Function to print the price the derivative at a given time and underlying level
  - Compute_delta: computation of the delta of the derivative
  - Compute_gamma: computation of the delta of the derivative
  - Compute_theta: computation of the delta of the derivative
- Pde:
  - Compute the different boundary values according to the boundary type (Neumann or Dirichlet).
- Pde_solver: Used to create the space/time mesh grid