université PARIS-SACLAY

Option Pricing using Finite Difference Method

ENSIIE / Université Paris Saclay

C++/VBA Project

Students:
Gadaleta Pietro
Stampini Francesco

Under guidance of:
Pedro Ferreira
Vincent Torri

January 5, 2022

1 Introduction

The aim of this project is to develop an implicit Finite Differences method to price different types of option; more precisely, we will deal with European and American Puts and Calls.
To do that, we will solve the Black-Scholes PDE. After that, we will proceed with the computation of the Greeks of the option, and we will report the result on Excel. The functions used to solve the PDE are written on C++, you can find all the documentation of the files here, a .html file created using Doxygen.
Subsequently, the C++ files were used to create a `.dylib` file, a dynamic library, (the MacOS equivalent of a Windows .dll), since the both of us are working on this operative system; for security reasons, the `.dylib` file must be put in the folder *Application Support/Microsoft*.
Finally, we have created an Interface in Excel, where you can insert the inputs that will be used in a VBA module to recall the C++ function in the dylib. Moreover, in VBA, we will plot the graphs of the option and of the Greeks. It must be noted that the plots of the Delta and Gamma over the underlying price as are always shown, whereas to plot the other Greeks a box must be checked, this due to the fact that such computation significantly slows down the computational speed and the time to obtain results.

2 Finite Difference Methods

Finite Difference Methods value a derivative by solving the differential equation that the derivative satisfies. In order to do that, a discretization value of the derivative is used, but additional complexity can be due to the geometry of the domain, for instance when it is necessary to use an unstructured, adaptive discretization mesh. Luckily, finance problems are typically posed on rectangular domains.
The solution is a four step process [1]:

1. **Transformation** of the problem, such as a change of variable;
2. **Localization**, replacing the unbounded problem domain (in case is unbounded of course) by bounded domain, denoted $\mathcal{O}$, introducing a boundary condition for the function f outside $[0, T)$x$\mathcal{O}$, such that $f = \Gamma$ at T;
3. **Discretizing** the problem using a mesh over $[0, T)$x$\mathcal{O}$;

4. **Resolution** of the problem using linear algebra in the node of the mesh.

In our case, the first equation to solve is the well-known Black-Scholes PDE:

$$\begin{cases} v(T,S) = \phi(S), \quad S \in (0,+\infty) \\ \partial_t v + (r-q)S\partial_S v + \frac{1}{2}\sigma^2 S^2 \partial^2_{S^2} v - rv = 0 \quad \text{in } [0,T) \times (0,+\infty) \end{cases} \tag{1}$$

where S represents the underlying stock's price, $v(t,S)$ is the price process of a European Vanilla Option with integrable payoff $\phi(S_T)$, r is the constant interest rate, σ is the volatility and q the dividend yield. In our case, we have sligthly simplified the problem considering $d = 0$, the resulting equation is so:

$$\begin{cases} v(T,S) = \phi(S), \quad S \in (0,+\infty) \\ \partial_t v + rS\partial_S v + \frac{1}{2}\sigma^2 S^2 \partial^2_{S^2} v - rv = 0 \quad \text{in } [0,T) \times (0,+\infty). \end{cases} \tag{2}$$

Another important variable in pricing the option is T, the time to maturity of the contract. The time mesh is an equally spaced grid of length $N\Delta t$ where each interval is of length Δt.
As explained before, the first step of the Finite Differences method is the **transformation of the variable**. This is not strictly necessary, but in this case it is computationally more efficient to use the log-price $\log S(t)$ rather than $S(t)$. Defining $Z(t) = \log S(t)$, we obtain from Eqn. (2):

$$\begin{cases} f(T,z) = \psi(z), \quad z \in R \\ \partial_t f + (r - \frac{\sigma^2}{2})\partial_z f + \frac{1}{2}\sigma^2 \partial^2_{z^2} f - rf = 0 \quad \text{in } [0,T) \times (-\infty,+\infty), \end{cases} \tag{3}$$

where $\psi(z) = \phi(e^z)$, and the price process is given by $f(t, Z_t)$. The second step is the **localization of the domain**; usually, without the change of variable, we would have to define $S_{\max}$, a stock price high enough to assume that:

- when dealing with **puts**, the option has virtually zero value;
- when dealing with **calls**, the price is always constant and equal to $S_{\max} - K$.

However, after the change of variable, we can define the new space domain to $\mathcal{O} = [z-l, z+l]$, where $z_0 = \log S_0$ and l is chosen using the quantile, such that

$$P\left(|Z_t - z_0| \leq \ell, t \in [0,T]\right) \geq 1 - \alpha \tag{4}$$

for a $\alpha > 0$ sufficiently small. Since the stock's price follows a log-normal distribution, $\log S_t$ follows a Normal distribution and, for this reason, the formula above can be resolved setting:

$$\ell = |r - \frac{\sigma^2}{2}|T + g\sigma\sqrt{T} \tag{5}$$

where g is a "sufficiently high" quantile of the Gaussian distribution.
Equation (3) can be trasformed into:

$$\begin{cases} f(T,z) = \psi(z), \quad z \in R \\ \partial_t f + (r - \frac{\sigma^2}{2})\partial_z f + \frac{1}{2}\sigma^2 \partial^2_{z^2} f - rf = 0 \quad \text{in } [0,T) \times (z_0 - l, z_0 + l). \end{cases} \tag{6}$$

The next step is the **discretization**, where we will briefly introduce the approximation used. Considering a smooth function f, it is possible to compute its Taylor Expansion around x. Indeed, given a small increment δ, the expansion is:

$$\begin{aligned} f(x+\delta) &= f(x) + \delta \cdot f'(x) + \frac{0^2}{2} f''(x) \\ &+ \frac{\delta^3}{6} f'''(x) + \frac{\delta^4}{24} f^{(4)}(x) + \frac{\delta^5}{120} f^{(5)}(x) + O\left(\delta^6\right), \end{aligned}$$

which implies:

$$\begin{aligned} f'(x) &= \frac{1}{\delta}[f(x+\delta) - f(x)] - \frac{\delta}{2} f''(x) \\ &- \frac{\delta^2}{6} f'''(x) - \frac{\delta^3}{24} f^{(4)}(x) - \frac{\delta^4}{120} f^{(5)}(x) + O\left(\delta^5\right), \end{aligned}$$

Stopping at the first order term, we can write:

$$f'(x) = \frac{1}{\delta}[f(x+\delta) - f(x)] + O\left(\delta\right), \tag{7}$$

which is known as the *forward difference* approximation. We can repeat the same procedure with $-\delta$ and sum the two equations, obtaining

$$f'(x) = \frac{f(x+\delta) - f(x-\delta)}{2\delta} - \frac{\delta^2}{6} f'''(x) + O\left(\delta^4\right),$$

which, when stopping at the second term, becomes:

$$f'(x) = \frac{f(x+\delta) - f(x-\delta)}{2\delta} + O\left(\delta^2\right), \tag{8}$$

the so called *central-difference* second order approximation of f'.
On the other hand, if instead of sum we subtract the equation, we will obtain:

$$f''(x) = \frac{f(x+\delta) - 2f(x) + f(x-\delta)}{\delta^2} - \frac{\delta^2}{12} f^{(4)}(x) + O\left(\delta^4\right),$$

that, when truncating, becomes:

$$f''(x) = \frac{f(x+\delta) - 2f(x) + f(x-\delta)}{\delta^2} + f^{(4)}(x) + O\left(\delta^2\right), \tag{9}$$

the *central-difference* second order approximation of f''.
We can use Equation (8) as in [2] for an interior point of the grid to approximate the first order derivative:

$$\frac{\partial f}{\partial Z} = \frac{f_{i,j+1} - f_{i,j-1}}{2\Delta Z}. \tag{10}$$

while, for the time derivative, we will use a forward difference approximation since the value at time $i + \Delta t$ is only related with the value at time $(i+1)\Delta t$ and not $(i-1)\Delta t$, obtaining

$$\frac{\partial f}{\partial t} = \frac{f_{i+1,j} - f_{i,j}}{\Delta t}. \tag{11}$$

For the second derivative we will use equation (9)

$$\frac{\partial^2 f}{\partial Z^2} = \frac{f_{i,j+1} + f_{i,j-1} - 2f_{i,j}}{\Delta Z^2}. \tag{12}$$

The difference equation becomes

$$\frac{f_{i+1,j} - f_{i,j}}{\Delta t} + \left(r - \sigma^2/2\right) \frac{f_{i,j+1} - f_{i,j-1}}{2\Delta Z} + \frac{1}{2}\sigma^2 \frac{f_{i,j+1} + f_{i,j-1} - 2f_{i,j}}{\Delta Z^2} = r f_{i,j}, \tag{13}$$

that can be resumed, using a different notation than [2],

$$a_j f_{i,j-1} + b_j f_{i,j} + c_j f_{i,j+1} = f_{i+1,j}, \tag{14}$$

where

$$\begin{aligned} a_j &= \frac{\Delta t}{2\Delta Z}\left(r - \sigma^2/2\right) - \frac{\Delta t}{2\Delta Z^2}\sigma^2, \\ b_j &= 1 + \frac{\Delta t}{\Delta Z^2}\sigma^2 + r\Delta t, \\ c_j &= -\frac{\Delta t}{2\Delta Z}\left(r - \sigma^2/2\right) - \frac{\Delta t}{2\Delta Z^2}\sigma^2. \end{aligned} \tag{15}$$

The last step is the **Resolution**. We can observe that it is possible to rewrite equation (14) in Matrix form

$$\begin{bmatrix} a_1 & b_1 & c_1 & & & & 0 \\ & a_2 & b_2 & c_2 & & & \\ & & a_3 & b_3 & \ddots & & \\ & & & \ddots & \ddots & c_{M-2} & \\ 0 & & & & a_{M-1} & b_{M-1} & c_{M-1} \end{bmatrix} \begin{bmatrix} f_{i,0} \\ f_{i,1} \\ f_{i,2} \\ f_{i,3} \\ \vdots \\ f_{i,M} \end{bmatrix} = \begin{bmatrix} f_{i+1,1} \\ f_{i+1,2} \\ f_{i+1,3} \\ \vdots \\ f_{i+1,M-1} \end{bmatrix}, \tag{16}$$

Since the value of $f_{i,0}$ and $f_{i,M}$ are constant (being the boundary values, as previously explained), we can remove the first and last column, to rewrite the matrix equation in a square matrix form:

$$\begin{bmatrix} b_1 & c_1 & 0 & \dots & 0 \\ a_2 & b_2 & c_2 & \ddots & \vdots \\ 0 & a_3 & b_3 & \ddots & 0 \\ \vdots & \ddots & \ddots & \ddots & c_{M-2} \\ 0 & \dots & 0 & a_{M-1} & b_{M-1} \end{bmatrix} \begin{bmatrix} f_{i,1} \\ f_{i,2} \\ f_{i,3} \\ \vdots \\ f_{i,M-1} \end{bmatrix} = \begin{bmatrix} f_{i+1,1} - a_1 f_{i,0} \\ f_{i+1,2} \\ f_{i+1,3} \\ \vdots \\ f_{i+1,M-1} - c_{M-1} f_{i,M} \end{bmatrix}. \tag{17}$$

In order to simplify the notation, we will use the following:

$$\begin{bmatrix} b_1 & c_1 & 0 & \dots & 0 \\ a_2 & b_2 & c_2 & \ddots & \vdots \\ 0 & a_3 & b_3 & \ddots & 0 \\ \vdots & \ddots & \ddots & \ddots & c_{M-2} \\ 0 & \dots & 0 & a_{M-1} & b_{M-1} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_{M-1} \end{bmatrix} = \begin{bmatrix} d_1 \\ d_2 \\ d_3 \\ \vdots \\ d_{M-1} \end{bmatrix}. \tag{18}$$

The Matrix is a Tridiagonal matrix, hence to solve this problem we can use the **Thomas Algorithm**, a simplified version of Gaussian elimination that can be used to solve tridiagonal systems of equations. The Algorithm is the following

$$c'_i = \begin{cases} \frac{c_i}{b_i}, & i = 1 \\ \frac{c_i}{b_i - a_i c'_{i-1}}, & i = 2, 3, \dots, M-2 \end{cases} \tag{19}$$

and

$$d'_i = \begin{cases} \frac{d_i}{b_i}, & i = 1 \\ \frac{d_i - a_i d'_{i-1}}{b_i - a_i c'_{i-1}}, & i = 2, 3, \dots, M-1. \end{cases} \tag{20}$$

By back substitution the solution is then obtained:

$$\begin{aligned} x_{M-1} &= d'_{M-1} \\ x_i &= d'_i - c'_i x_{i+1}, \quad i = M-2, M-3, \dots, 1. \end{aligned} \tag{21}$$

We will repeat this procedure N times, in order to compute the vector of prices at time $t = 0$, i.e. $\left[\ f_{0,1}, f_{0,2}, ..., f_{0,M-1}\ \right]$.

However, for the American there is an intermediate step, each time in fact, in the case of a put option for example, $f_{N-1;j}$ is compared with $K - S_j$. If $f_{N-1;j} < K - S_j$, early exercise at time $T - dt$ is optimal and $f_{N-1;j}$ is set equal to $K - S_j$. In a similar way, the following nodes corresponding to $T - 2dt, T - 3dt, ...$ are handled. At the end, $\left[\ f_{0,1}, f_{0,2}, ..., f_{0,M-1}\ \right]$ are obtained.

3 Greeks

Greeks are very useful measure in Mathematical Finance, since they represent the sensitivities of the option to a movement in the underlying parameters. The most famous greek is the *Delta*

$$\Delta = \frac{\partial \phi}{\partial S}, \tag{22}$$

to compute this greek we have used the formula (8) and the values of the stock grid of the Finite Difference method.
The *Gamma* is

$$\Gamma = \frac{\partial^2 \phi}{\partial S^2}, \tag{23}$$

so to compute we have used the formula (9).
The other greeks are *Rho, Vega, Theta*

$$\rho = \frac{\partial \phi}{\partial r}, \quad \mathcal{V} = \frac{\partial \phi}{\partial \sigma}, \quad \Theta = \frac{\partial \phi}{\partial T}, \tag{24}$$

to calculate these greeks we have the formula (7), since would be to computational expensive to use the formula (8).

4 The Program

The implementation of the code has been done both in `C++` and in `VBA`.

The former has been used in order to compute prices using the Finite Differences method. More precisely, we used two classes to define the needed objects:

- class `mesh` defines both time and stock meshes. The latter is actually defined using a derived class `stock_mesh`, that also stores the index of the actual initial spot price S_0 which is needed when finding the price of the option;

- class `option` takes in the constructor all the input parameters, and internally computes the option price and greeks, as well as all the vectors needed in order to plot graphs in Excel.

As previously explained, the option is priced using an implicit scheme, and the algorithm used to solve the equation is the Thomas Algorithm.
An auxiliary function `compute_all` is then passed on to VBA using a `.dylib` Dynamic Library[1]. In order to pass multiple values onto `VBA`, the option price and all the greeks are stored in an array passed by reference as an input of the function from `VBA` to C++, whereas all the values needed to plot graphs are stored in another array, always passed onto C++ by reference.
The figures below show the program's interface with a mock example. It can be seen that the also the prices computed using BS Formula are always shown on the program, and are instantaneously updated. They can be used as reference to check results obtained using the approximation scheme.

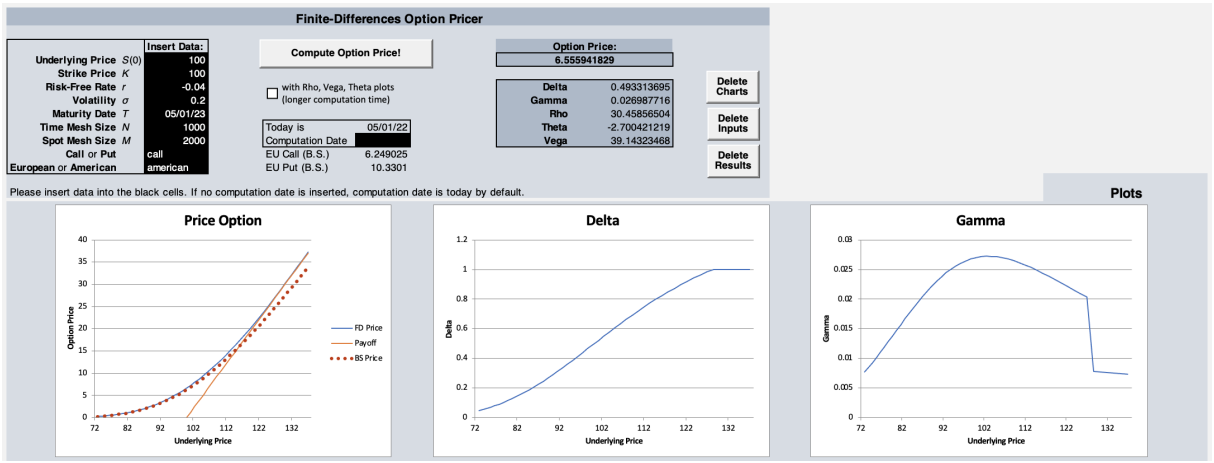

Figure 1: Program's Interface on Excel. A mock example is shown with an American Call, negative interest rate.

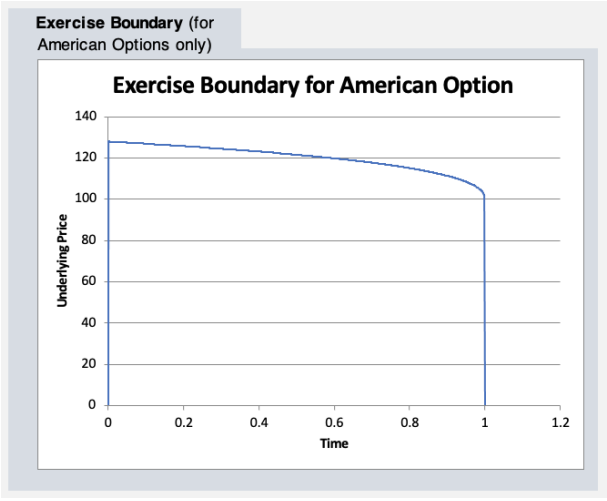

Figure 2: Plot of option's exercise barrier.

[1]It must be noted that all of the parameters passed from VBA to this function are of `double` type, even the ones which are supposed to be `long int` (such as N, M) and `bool` (such as `isAmerican, isCall`). This is because, if passed as long integers and booleans on C++, Excel would crash in all of our trials. Due to lack of documentation on the Internet related to using .dylib files on VBA, we found this as a solution to the problem. Passing everything should not generate errors: the values of the booleans are set to 1.0 or 0.0 on the VBA code (so they cannot be some other number which is not recognized as a boolean), whereas N and M, in case they present decimal points, are going to be rounded on C++ when converted into long integers.

Bibliography

[1] Stéphane Crépey (2013) *Financial Modeling a Backward Stochastic Differential Equations Perspective*, Springer.

[2] Hull J.C. (1994) *Futures and Other Derivatives*, 9th edition.