Code developer documentation and testing results

1 Explicit Finite Difference Method

The structure of the code used to implement this numerical method is shown in figure 1. The parameters which are required in the code in order for it to run are:

- K: Strike Price
- S_o: Spot Price
- r: Interest rate
- q: Dividend yield
- B: Barrier Level
- T: Time to maturity
- α: Exponent in local volatibity function
- N: Total number of divisions on the price domain
- M: Total number of time steps desired.

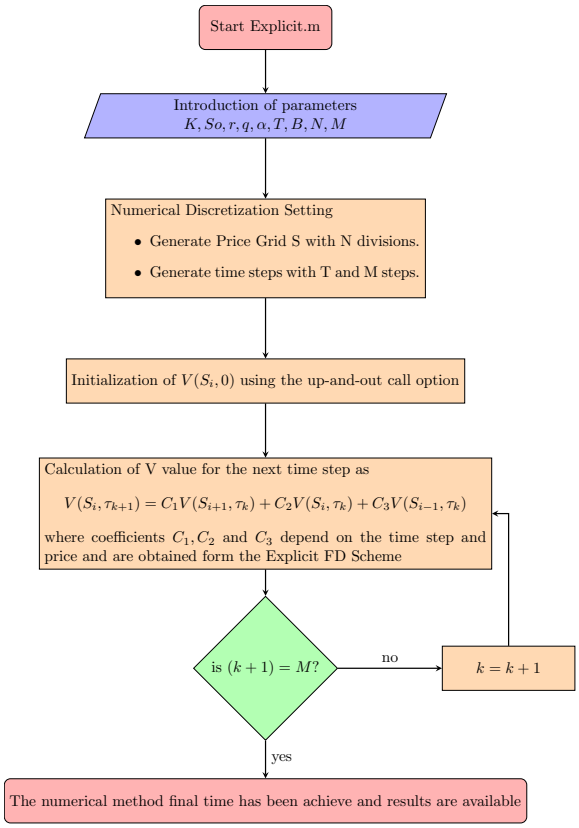

Figure 1: Explicit.m Code Structure

These parameters are must be specfied by the user in the first section of the code. Once these parameters are defined the code to generate the discretized domain where the Black-Scholes PDE is going to be solve. This is done by generating two arrays S and tau which contains the discretized stock prices and times. Then a matrix V_{kj} is created to store the values of the price of options for all time steps and stock price in vector S and tau. The first row of the V matrix is initialized by using the up-and-out call option initial condition. Then using a a double *for* loop the values of V for each time step and each stock price is calculated (This corresponds to lines 40 to 50 in the code). In order to take into account the boundary conditions, an if structure is whithin the loops so that in the boundaries of Matrix V a different function is used to calculate the new value of V for the next time step. If a programmer wants to extend the code by changing the PDE which is going to be solved or he wishes to optimize the code, the *Solve BS PDE (EXPLICIT)* section of the code has to be modified. Once the code has finished the double **for** loop, it generates the results. The results are the value of call option for the spot price at time

$\tau = T$ and two figures, one which is the option value price as function of S for $\tau = T$ and the other being a 3D surface of the behavior of V as function of stock price S and time.

The only function implemented in this code is the local volatibity function called *sigma*. This function has two parameters *ti* and *Sn* which are the time and stock price at which the volatibility is going to be evaluted. This function returns a vector or value corresponding to the volatibty value.

2 Implicit Finite Difference Method

The structure of the code used to implement this numerical method is shown in figure 2. The parameters which are required in the code in order for it to run are the same ones which were defined for the explicit FDM. The structure of the code regarding the parameter specification and the numerical discretization setting are the same as the one described for the explicit FDM.

For solving time steps for V a *for* loop is used (This corresponds to lines 40 to 48 in the code). If a programmer wants to extend the code by changing the PDE which is going to be solved or he wishes to optimize the code, the *Solve BS PDE (IMPLICIT)* section of the code has to be modified. Once the code has finished the **for** loop, it generates the results. The results are the value of call option for the spot price at time $\tau = T$ and two figures, one which is the option value price as function of S for $\tau = T$ and the other being a 3D surface of the behavior of V as function of stock price S and time.

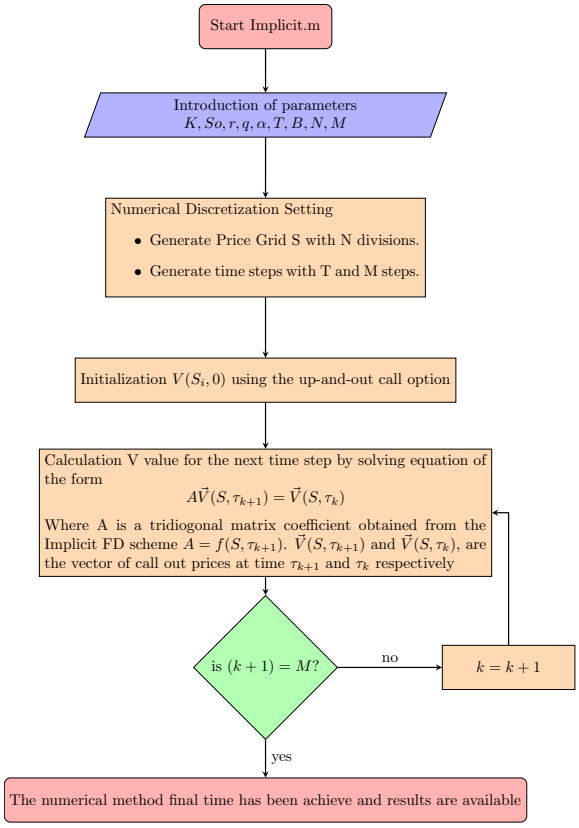

Figure 2: Implicit.m Code Structure

The first function implemented in this code is the local volatibity function called *sigma* which is idential to the one described previously in the Explicit FDM section.The second function in the code is the *tridiag* function which purpose iis to solve the matrix equation $Ax = B$ for vector x being A a tridigonal matrix. The parameters of this method are the vector corresponding to the diagonal, upper and lower coefficient vectors (D,U,L) and the solution vector B. As the return of this function is the vector x.

3 Crank Nicholson Finite Difference Method

The structure of the code used to implement this numerical method is shown in figure 3. The parameters which are required in the code in order for it to run are the same ones which were defined for the explicit FDM. The structure of the code regardign the parameter specification and the numerical discretization setting are equal to the one described for the explicit FDM.

For solving time steps for V a *for* loop is used (This corresponds to lines 39 to 60 in the code). If a programmer wants to extend the code by changing the PDE which is going to be solved or he wishes to optimize the code, the *Solve BS PDE (CN)* section of the code has to be modified.The code within the **for** loop is divided in two parts, the first one being generating the solution vector B from the values of V in a previous time step and the a tridiagonal coefficient matrix which is obtained from the explict part of the numerical scheme. The second part correspond to calculating the new time step with the coefficient matrix corresponding to the implicit part of the scheme and the solution vector B. Once the code has finished the **for** loop, it generates the results. The functions implemented for this code are the same ones as the functions described for the implict finite difference method.

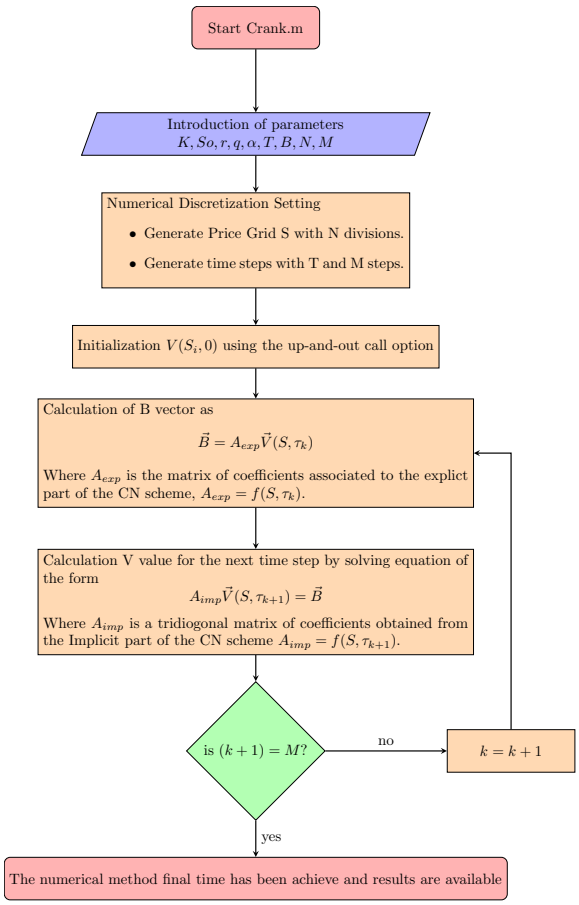

Figure 3: Crank.m Code Structure

4 Monte Carlo Simulation

The structure of the code used to implement this numerical method is shown in figure 3. The parameters which are required in the code in order for it to run are the same ones which were defined for the explicit FDM and and additional parameter $NSim$ which inidicate the number of path simulations which the user requires for the Monte Carlo method. The structure of the code regardign the parameter specification and the numerical discretization setting are equal to the one described for the explicit FDM.

For solving V profile at time $\tau = T$ a *for* loop is used (This corresponds to lines 39 to 60 in the code). If a programmer wants to extend the code by changing the PDE which is going to be solved or he wish to optimize the code the *Solve BS MC* section of the code has to be modified.The code within the **for** loop is divided in two parts, the first one being generating the paths followed by a stock price following the BS equation. The second part correspond to calculating the value of V at time $\tau = T$

by taking the avarerage path of prices S in time and applying the up-and-out call option. Once the code has finished the **for** loop, it generates the results. The only function implimented in this code is the volativity local function *sigma* which has already been described in the explicit FD method.

5 Test runs, Timing and Convergence

Test runs were made in which the stock price S mesh base size was variated. The number of divisions of S evaluated in the test runse were $N = 10, 30, 90, 120, 150$. The results regarding timing of each on of the codes are shown in figure 5 . Similarly the analyis of convergence of the FDM codes with respect to the Monte Carlo method is shown in figure 6.

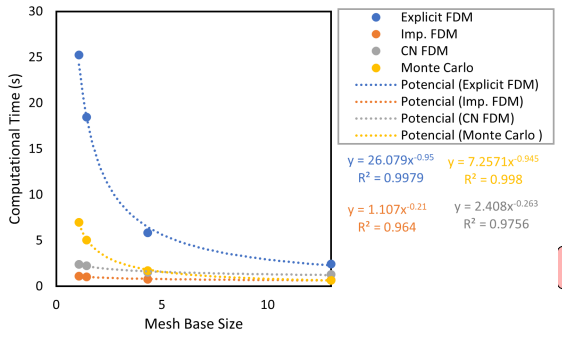

Figure 5: Timing evaluation of the FDM and Monte Carlo methods.

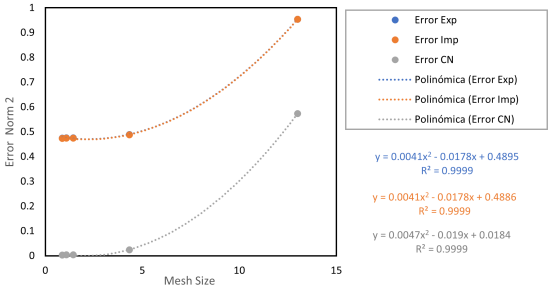

Figure 6: Convergence evaluation of the FDM methods with respect to Monte Carlo.

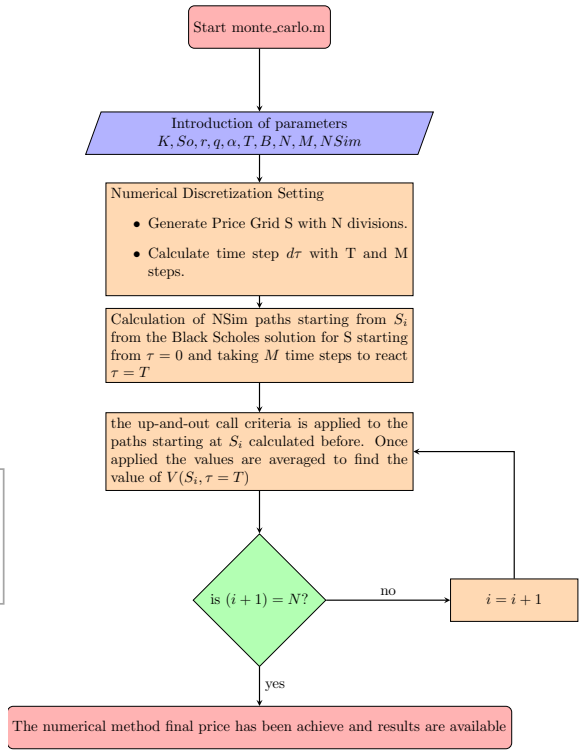

Figure 4: monte_carlo.m Code Structure

With respect to timing, methods such as Explicit FDM and Monte Carlo given to the presence of the double *for* loop in the coding the computational time taken by these method is $Comp.Time \sim 1/h$ while the Implict and CN FDM implementation only have one loop and are solve by mean of matrices which is why this methods are faster and $Comp.Time \sim h^{-0.25}$ where $h \sim 1/N$. On the other hand with respect to the convergen evaluation given that all implementations showed a cuadratic order of convergence $O(h^2)$ which was expected given that only second order schemes were used for the FDM codes.