Developer documentation

January 2022

1 Introduction

This is a developer documentation for calculating barrier option prices with the finite difference method implemented with MATLAB. The purpose of this document is to describe the functionality and role of the various MATLAB scripts and functions that have been developed.

2 What contain in the zip file

When you unzip the file you will find a number of files which are MATLAB scripts, two documents, the developer documentation (this document) and end-user's instruction. The MATLAB scripts (.m files) consist of the following files:

1. BarrierCallOption_BS.m
2. CrankNicolsonFDM.m
3. tridiag.m
4. tridiag_prod.m
5. test.m
6. test_impact_of_M.m
7. test_impact_of_N.m

3 Function available

3.1 BarrierCallOption_BS

BarrierCallOption_BS is the main function which was implemented, it is the function that receives the information regrading the barrier option price such as the minimun stock price (Smin), the maximum stock price (Smax), strike price (K), interest rate (r, in %), dividend yield rate (d, in %), volatility rate (sigma, in %), barrier level (B), the number of grid points for the space (N) and time intervals (M). And the boolean parameter (plt) controls whether the plots, the prices of the option at time 0 against the stock spot prices and the surface plot for the option prices corresponding to the FDM grid in S and t will be displayed. Then produce the approximation the call option prices corresponding the given parameters.

Algorithm 1 BarrierCallOption_BS

```
Require: $Smin, Smax, T, N, M, r, d, \sigma, K, B, plt$;
 1: Rearrange parameters $r, d, \sigma$, change from % to decimal;
 2: Construct the space interval [a,b] by given values $Smin, Smax$;
 3: Construct the time interval [0,$\sigma^2 T/2$];
 4: Compute $\Delta x$ and $\Delta \tau$;
 5: Compute the $\rho$ paramerter for the FDM;
 6: Compute $\alpha$ and $\beta$ parameters for $v(x, \tau)$ by using $r, d, \sigma^2$;
 7: Create the placeholder solution matrix to store value of $u(x, \tau)$ at all grid
    points;
 8: Introduce the initial and boundary conditions for the heat equation $u(x, \tau)$;
    $\triangleright$ * In this implementation we use an up-and-out call option with zero rebate
    payoff function;
 9: Solve the heat equation via Crank Nicolson FDM;
10: Create the placeholder for $V(S, t)$ with size (N-1,M+1);
11: for k=1 to M+1 do
12:     Compute $V(S, t)$ from $u(x, \tau)$;
13: end for
14: Calculate $S$ from $x$;
15: Calculate $t$ from $\tau$;
16: if plt is TRUE then
17:     2D_plot(S,V);
18:     3D_plot(t,S,V);
19: end if
20: return $(S, t, V)$
```

Note: One may consider to extend this MATLAB function. For example, by adding a boolean parameter to let users choose their perfer FDM solver, e.g., implicit or explicit methods. An another way to extend this implementation to general cases is using different payoff function, e.g., Down-and-in Call/Put, Up-and-in Call/Put or Down-and-out Call/Put.

3.2 CrankNicolsonFDM

CrankNicolsonFDM is the function to solve the solution for heat equation via Crank-Nicolson finite difference method (FDM) which receives the tri-diagonal matrix UC corresponding to the heat equation with its initial and boundary conditions, the parameter ρ for the Crank-Nicolson FDM and the number of grid points for the space and time intervals.

This script is implemented by dividing the calculation into 2 parts and each with two sub-steps:

1. Prepare the tri-diagonal matrices AE (Explict) and AI (Implict).
   (a) Set the diagonal (Di, size=N-1), upper diagonal (Ui, size=N-1), lower diagonal (Li, size=N-1) vectors for the tri-diagonal matrix AI (Implict).
   (b) Set the diagonal (De, size=N-1), upper diagonal (Ue, size=N-1), lower diagonal (Le, size=N-1) vectors for the tri-diagonal matrix AE (Explict).
2. Iteration to compute the approximation matrix UC; iterate k=1 to k=M,
   (a) Multiply UC at k by I+Ae by using *tridiag_prod* function.
   (b) Solve tri-diagonal linear system of equations via *tridiag* function.

Algorithm 2 CrankNicolsonFDM

```
Require: $rho, UC, N, M$;
1: Set (Di, size=N-1), (Ui, size=N-1), (Li, size=N-1) vectors;
2: Set (De, size=N-1), (Ue, size=N-1), (Le, size=N-1) vectors;
3: for k=1 to M do
4:     x = tridiag_prod (De,Ue,Le,UC(k));
5:     UC(k+1) = tridiag (Di,Ui,Li,x);
6: end for
7: return $UC$
```

3.3 tridiag and tridiag_prod

tridiag is a function to find a solution x of linear system $Ax = b$ when A is a tri-diagonal matrix. To perform this function, it receives 3 arrays represent diagonal, upper-diagonal, lower-diagonal values (D, U, L) and the right-hand side vector B. Then return a solution for this linear system.

tridiag_prod is a function to compute the product between a tri-diagonal matrix and a vector. To perform this function, it receives 3 arrays represent diagonal, upper-diagonal, lower-diagonal values (D, U, L) and the vector X. Then return the product AX.

Note: One may consider use the MATLAB build-in functions or other implements that able to solve the linear system of a tri-diagonal matrix and compute the product between a tri-diagonal matrix and a given vector effectively (in sense of computational time and accuracy).

4 Numerical tests

4.1 Example

The following is an example of using my implementation for calculating barrier option prices with the Crank-Nicolson FDM (as in the test.m file).

```
%Clear memory and console output
clc
clear

%Set the number of grid points
N = 500;          % For the space interval [a,b]
M = 100;          % For the time interval [0,T]

%Paramerters for Barrier Option Pricing
S0 = 95;          % Stock Price
K = 110;          % Strike Price
B = 4*K;          % Barrier level (use 4*K)
Smin = 0.001;     % Set minimum stock price
Smax = 4*K;       % Set maximum stock price
T = 1;            % Term (years)
r = 5;            % Interest Rate (%)
d = 0;            % Dividend Yield (%)
sigma = 25;       % Volatility (%)

%Solve FDM for BS
[S, t, V] = BarrierCallOption_BS(Smin,Smax,T,N,M,r,d,
    sigma,K,B,true);

%Calculate Up-and-out Barrier option price when the Stock
     Price is S0
interp1(S,V(:,end),S0)
```

This example takes values for each parameters as its shown in the code, and obtained the call option price (By giving the Stock Price is $S0$) is 5.8424. By using an online tool (http://www.coggit.com/freetools), one will obtains the (Up-and-out) call option price for this setting is 5.84. That is the absolute different between our implementation's and online tool's result is 0.0024.

Moreover, one will obtains the plot for the option at time 0 against the stock spot price and the 3D plot for the option prices with the whole FDM grid in S and t as in the figures (1, 2) below.

4.2 Performance

To demonstrate this implementation performance, one can try to run the MATLAB *test_impact_of_M.m* or *test_impact_of_N.m* scripts. These scripts will gives

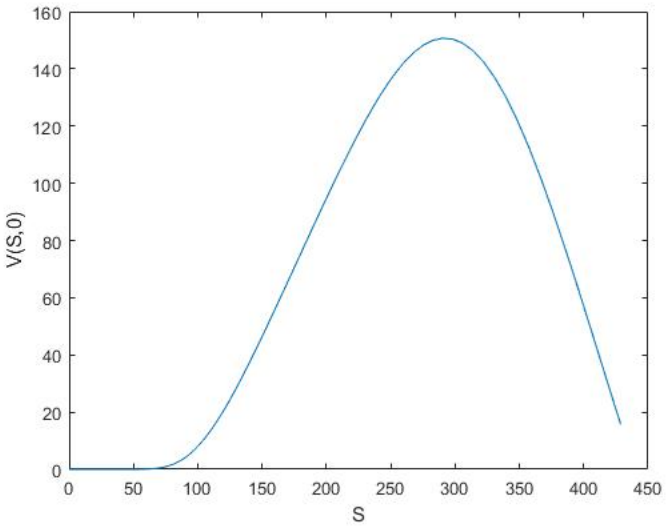

Figure 1: the option at time 0 plot against the stock spot price

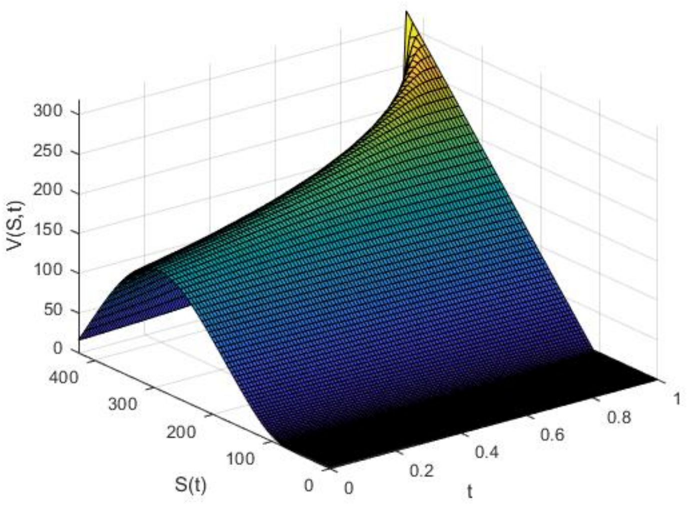

Figure 2: the option prices plot with the whole FDM grid in $S(t)$ and t

N	500	500	500	500	500	500	500	500	500	500
M	100	200	300	400	500	600	700	800	900	1000
call option price	5.8424	5.8424	5.8424	5.8424	5.8424	5.8424	5.8424	5.8424	5.8424	5.8424
error	0.0024	0.0024	0.0024	0.0024	0.0024	0.0024	0.0024	0.0024	0.0024	0.0024

Table 1: N fixed and M varied.

N	100	200	300	400	500	600	700	800	900	1000
M	100	100	100	100	100	100	100	100	100	100
call option price	6.0876	5.9270	5.8472	5.8537	5.8424	5.8395	5.8466	5.8449	5.8391	5.8404
error	0.2476	0.0870	0.0072	0.0137	0.0024	0.0005	0.0066	0.0049	0.0009	0.0004

Table 2: N varied and M fixed.

the absolute errors between our and online's tool results by using the same setting but different the value of $M(N)$ which can be summarised in the tables (1, 2).

Note: All scripts used the same setting except for the number of grid points N and M.