Code developer documentation

April 2022

1 Introduction

This is a developer documentation for calculating barrier call option prices (an up-and-out call option) with the finite difference methods and Monte Carlo simulation implemented with MATLAB. The implemented finite difference methods are including explicit, implicit and Crank-Nicolson schemes. The purpose of this document is to describe the functionality and role of the various MATLAB scripts and functions that have been developed.

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: *explicit.m*, *implicit.m*, *crank.m*, *monte_carlo.m* and *main.m*

The first four scripts are the implementations (finite difference methods and Monte Carlo simulation) and the last file (main.m) is the main script that calls the implemented functions and displays the computed result and the corresponding plot.

2 Function available

2.1 explicit

explicit is an explicit finite difference method implementation for solving a local volatility Black-Scholes model shown in algorithm 1. It is the function that receives the information regrading the barrier call option price such as stock price (S0), strike price (K), barrier level (B), time to maturity (T), interest rate (r), dividend yield (d), the local volatility surface parameter (α) and the number of grid points for the space (N) and time intervals (M). Then produces the approximation the call option prices corresponding the given parameters.

Algorithm 1 explicit

```
Require: $S0, K, B, T, r, d, \alpha, N, M$;
 1: Rearrange parameters $r, d$, change from % to decimal;
 2: Construct the space interval [a,b] by given values $Smin = 0, Smax = B$ and the time interval [0,T];
 3: Compute $\Delta S$ and $\Delta \tau$;
 4: Calculate the local volatility surface function: $\sigma(S,t) = 0.25e^{-t}(100/S)^{\alpha}$;
 5: Calculate the $\alpha_{S,t}$ and $\beta_{S,t}$ parameters corresponding to $\sigma(S,t)$;
 6: Create the placeholder solution matrix to store value of $V(S,\tau)$ at all grid points with the initial condition
    $V(S,\tau=0) = \max\{S-K, 0\}$;
 7: for k=1 to M+1 do
 8:     Construct the tridiagonal matrix for the explicit scheme: $A_k^E$;
 9:     Iteration to compute the value $V_{k+1} = A_k^E V_k$;
10: end for
11: interpolation to find the call price at S0;
12: return Call option price at S0 and $V(S,\tau)$.
```

2.2 implicit

implicit is an implemented implicit finite difference method used for solving a local volatility model shown in algorithm 2.. This function receives the input parameters the same as in the explicit function above: $S0, K, B, T, r, d, \alpha, N, M$ and then return the corresponding approximation of the call option prices.

```
Algorithm 2 implicit
Require: $S0, K, B, T, r, d, \alpha, N, M$;
 1: Rearrange parameters $r, d$, change from % to decimal;
 2: Construct the space interval [a,b] by given values $Smin = 0, Smax = B$ and the time interval [0,T];
 3: Compute $\Delta S$ and $\Delta\tau$;
 4: Calculate the local volatility surface function: $\sigma(S,t) = 0.25e^{-t}(100/S)^{\alpha}$;
 5: Calculate the $\alpha_{S,t}$ and $\beta_{S,t}$ parameters corresponding to $\sigma(S,t)$;
 6: Create the placeholder solution matrix to store value of $V(S,\tau)$ at all grid points with the initial condition
    $V(S,\tau = 0) = \max\{S - K, 0\}$;
 7: for k=1 to M+1 do
 8:     Construct the tridiagonal matrix for the implicit scheme: $A^I_k$;
 9:     Iteration to compute the value $V_{k+1} = (A^I_k)^{-1}V_k$;
10: end for
11: interpolation to find the call price at S0;
12: return Call option price at S0 and $V(S,\tau)$.
```

2.3 crank

An implementation of the Crank-Nicolson finite difference method is written in *crank* function which receives the input parameters and return the estimate of the call option price the same as in the explicit and implicit functions.

```
Algorithm 3 crank
Require: $S0, K, B, T, r, d, \alpha, N, M$;
 1: Rearrange parameters $r, d$, change from % to decimal;
 2: Construct the space interval [a,b] by given values $Smin = 0, Smax = B$ and the time interval [0,T];
 3: Compute $\Delta S$ and $\Delta\tau$;
 4: Calculate the local volatility surface function: $\sigma(S,t) = 0.25e^{-t}(100/S)^{\alpha}$;
 5: Calculate the $\alpha_{S,t}$ and $\beta_{S,t}$ parameters corresponding to $\sigma(S,t)$;
 6: Create the placeholder solution matrix to store value of $V(S,\tau)$ at all grid points with the initial condition
    $V(S,\tau = 0) = \max\{S - K, 0\}$;
 7: for k=1 to M+1 do
 8:     Construct the tridiagonal matrices for the Crank-Nicolson scheme: $A^E_k$ and $A^I_k$;
 9:     Iteration to compute the value $V_{k+1} = (A^I_k + I)^{-1}(A^E_k V_k)$;
10: end for
11: interpolation to find the call price at S0;
12: return Call option price at S0 and $V(S,\tau)$.
```

2.4 monte_carlo

monte_carlo is an implemented Monte Carlo simulation with antithetic sampling for solving a local volatility Black-Scholes model as shown in algorithm 4. It is the function that receives the information regrading the barrier call option price such as stock price (S0), strike price (K), barrier level (B), time to maturity (T), interest rate (r), dividend yield (d), the local volatility surface parameter (α) and the number of simulation (N). Then produces the approximation the call option price corresponding the given parameters along with its standard error.

Note

These functions are designed to be used for a specific problem of a local volatility model. If changing models, such as changing the local volatility function, it is necessary to modify the program (hard-code). Therefore, for flexibility, readability and maintainability of the program, the functions may be modified to receive the value of the surface function from outside instead of calculating the values within the function. In addition, this model is not generalized BlackScholes PDE models as interest rates and dividend rates are still constants. In the development of the program, the program may be able to calculate the option price under the situation where dividend and interest rates are not constants.

Algorithm 4 monte_carlo

```
Require: $S0, K, B, T, r, d, \alpha, N$;
 1: Rearrange parameters $r, d$, change from % to decimal;
 2: Calculate the local volatility surface function: $\sigma(S,t) = 0.25e^{-t}(100/S)^{\alpha}$ at (S0,T);
 3: Create placeholders to store sample of maturity stock prices and discounted call payoff prices;
 4: for k=1 to N do
 5:     sample Z ~ N(0,1);
 6:     calculate the stock price at time T: $S_k = S0\exp\{(r-d-0.5\sigma^2)T + \sigma\sqrt(T)Z\}$;
 7:     compute the call option price: $X_k = exp(-rT)\max\{S_k - K, 0\}$ when $S_k \le B$ otherwise $X_k = 0$;
 8:     calculate the stock price for the antithetic variates: $S_k = S0\exp\{(r-d-0.5\sigma^2)T - \sigma\sqrt(T)Z\}$;
 9:     compute the call option price: $Y_k = exp(-rT)\max\{S_k - K, 0\}$ when $S_k \le B$ otherwise $X_k = 0$;
10: end for
11: Compute the estimate call $\hat{C} = \frac{1}{N}\sum_{k=1}^{N}\frac{X_k+Y_k}{2}$ and the standard error $\sqrt{\frac{1}{N(N-1)}\left(\sum_{k=1}^{N}\left(\frac{X_k+Y_k}{2}\right)^2 - N\hat{C}\right)}$;
12: return Call option price at S0, $\hat{C}$, and its standard error.
```

3 Numerical examples

3.1 Performance and Timing

In order to demonstrate the use and test the performance of the program. One can set the pricing parameters and calls the option price functions using different methods as the following code snippet:

```
%% the problem parameters
S0 = 100;              % spot price (in British Pound)
K = 90;                % strike price (in British Pound)
B = 130;               % barrier level (in British Pound)
r = 3;                 % risk-free rate (in %)
q = 5;                 % dividend yield (in %)
T = 0.5;               % time to maturity (years)
vola_alpha = 0.35;     % the local volatility alpha

%% FDM: Set the number of grid points
N = 50;             % For the space interval [a,b]
M = 500;            % For the time interval [0,T]

%% solving the Black-Scholes PDE using explicit FDM
[call, V] = explicit(S0,K,B,T,r,q,vola_alpha,N,M);

%% solving the Black-Scholes PDE using implicit FDM
[call, V] = implicit(S0,K,B,T,r,q,vola_alpha,N,M);

%% solving the Black-Scholes PDE using Crank-Nicolson FDM
[call, V] = crank(S0,K,B,T,r,q,vola_alpha,N,M);

%% Monte Carlo simulation
N_sim = 10000;  % Number of simulations
[call, se_call] = monte_carlo(S0,K,B,T,r,q,vola_alpha,N_sim);
```

The call option prices obtained by the implemented functions are 9.3014, 9.3012, 9.3012 and 9.6959 for explicit, implicit, Crank-Nicolson and Monte Carlo schemes respectively. Also, the total time spent for each function are 0.019, 0.024, 0.032 and 0.013 seconds respectively (measured by the MATLAB profiler). Among the finite difference methods, it can be observed that Crank-Nicolson scheme takes the longest which is make sense because it is necessary to calculate the two matrices, A^E and A^I, and as well as calculating the matrix's inverse. While the explicit method is the second rank and the implicit method takes the least time. However, explicit method may not able to obtain the solution under some choice of (N,M) due to its stability problem that the error at some stage of the computation is blow up. To avoid such an issue, it is recommended that one must choose a pair of (N,M) in which keeps $\frac{d\tau}{dS}$ small as much as possible (where $dS = \frac{B}{N}$ and $d\tau = \frac{T}{M}$). The computational time of Monte Carlo simulation depends only on the number of simulation (N_sim in the snippet), larger number of simulation, higher computational time.

3.2 Comparison and Convergence

In order to compare with different sizes of grids and results from Monte Carlo method, the used grid sizes and their $\frac{d\tau}{dS}$ ratio is according to the following tables.

N	M	dS	$d\tau$	$\frac{dS}{d\tau}$
50	100	2.6	0.0050	0.00192
50	200	2.6	0.0025	0.00096
50	300	2.6	0.0017	0.00064
50	400	2.6	0.0013	0.00048
50	500	2.6	0.0010	0.00038
100	100	1.3	0.0050	0.00385
100	200	1.3	0.0025	0.00192
100	300	1.3	0.0017	0.00128
100	400	1.3	0.0013	0.00096
100	500	1.3	0.0010	0.00077

The table below displays the call option price estimated by each FDM scheme compare with different grid sizes. Theoretically, when dS and $d\tau$ close to zero, the solution obtained by FDM will be consistent and if the scheme is stable then the finite difference solution converges to the original partial differential equation solution. That is a larger number of grid points will leads to a good approximation, however, not that the selection of any grid sizes will make the explicit scheme stable as it was shown in the table. It can be seen that it is empty at the grid size (N=100, M=100) because of stability issue. In other words, there is a trade-off for limitations in choosing the grid size and the calculation time used.

Methods		M=100	200	300	400	500
Explicit	N=50	9.3019	9.3016	9.3015	9.3015	9.3014
	N=100	*	9.3253	9.3251	9.325	9.3249
Implicit	N=50	9.3005	9.3009	9.3011	9.3011	9.3012
	N=100	9.3235	9.3241	9.3243	9.3244	9.3245
Crank-Nicolson	N=50	9.3014	9.3013	9.3013	9.3013	9.3013
	N=100	9.3249	9.3248	9.3247	9.3247	9.3247

For the Monte Carlo simulation, the approximation solution will be consistent and accurate when the number of simulations approaches infinity. This can be seen in the follows table, higher number of simulations, smaller the standard error.

Monte Carlo	10000	20000	40000	80000	160000
Estimate	9.6562	9.696	9.6926	9.6775	9.6827
SE	0.0874	0.0624	0.044	0.031	0.022