Barrier Option Pricing

– A Monte Carlo Simulation Approach

Västerås 2013-10-23
Mälardalen University
MMA 708
Author: Jessica Radeschnig
Instructor: Jan Röman

Abstract

This report describes how Matlab can be used to solve the barrier-asset pricing problem with methods of Monte Carlo Simulation.

Contents

Purpose

The purpose of solving this assignments is to develop practical skills in how to price financial derivatives in a computer programming language.

1 Introduction

This first section of this report will introduce the reader to Monte Carlo Simulation and Barrier options. The second section give the discretized Black-Scholes formula that needs in order to simulate the stock price, followed by an developed algorithm that could perform the specific task. Next, Section 3 will give some examples with the use of the algorithm, followed by conclusions from this seminar. At last, Appendix A contains the Matlab codes for the algorithm and the examples.

1.1 Monte Carlo Simulation

Monte Carlo methods contain many different models that can make it possible to create sample outcomes. These models can be simulated over and over again in order for the sample size to increase. Further, by the law of large numbers, the average value of the outcomes will converge towards the real value. Thus, applying simulation to stocks make it possible to for instance find the expected future option pay off, which further can be used for pricing the option in question. See Glasserman (2003) for more details about Monte Carlo Simulations.

1.2 Barrier Options

Barrier options are *path-dependent options* whose pay off depend on weather the price of the underlying stock reaches a given barrier or not. These options can be classified as Knock - In Options[1] or Knock - Out Options[2]. In the first of these, the price of the underlying must intersect the barrier for the option to exist and in the latter case, the option's existence is terminated if the barrier is being intersected. See Hull (2010) for further readings.

When simulating, each iteration i will produce a different value of the option. The price of the option however, is martingale and thus will be the average of all the estimated values, that is,

$$V = \frac{1}{n}\sum_{i=1}^{n} V_i \tag{1}$$

where n is the number of simulations and

$$V_i = \begin{cases} C_i, & \text{for call options} \\ P_i, & \text{for put options} \end{cases}$$

Furthermore,

$$C_i = \mathbf{1}e^{-rT}\max\{S_i(T) - K, 0\}$$

and

$$P_i = \mathbf{1}e^{-rT}\max\{K - S_i(T), 0\}$$

where r is the risk-free interest rate, T is the time to maturity, $S(T)$ is the stock price at maturity and K is the strike price. In addition, $\mathbf{1}$ is the indicator function:

$$\mathbf{1} = \begin{cases} 1, & \text{if in the contract} \\ 0, & \text{if out of the contract} \end{cases}$$

The upper and lower bound of the 95% confidence interval for the option value is $V + 1.96s$ and $V - 1.96s$ respectively[3], where s represent the sample variance and is given by

$$s = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(V_i - \hat{V}_n)^2} \quad ,$$

for a finite but rather large value of n (Glasserman, 2003).

[1]Could be either a down-and-in or a up-and-in option.
[2]Could be either a down-and-out or a up-and-out option.
[3]The value 1.96 is found from $Z_{0.5/2}$ in the Normal Table

2 The Black-Scholes Model

In the Black-Scholes model, t is the current time and $S(t)$ represents the stock price at that time. T is the time of maturity, r is the annualised risk-free interest rate, σ is the annualized volatility and $W(t)$ is the Brownian Motion. Furthermore, $W(t)$ has a distribution of $\sqrt{T}Z$, where Z is a standard normal variable. The movements of the stock price is explained by

$$\frac{dS(t)}{S(t)} = rdt + \sigma dW(t)$$

and at maturity, the solution to this stochastic differential equation is given by

$$S(T) = S(0)e^{(r-\sigma^2/2)T+\sigma W(T)}.$$

Furthermore, one can discretize the above continuous formula of the the stock value through dividing the time into subintervals. Then the stock vale $S(t_{j+1})$, where $j = 0, 1, 2, \cdots, m$, is the time step from time 0 to time T, can be estimated as follows:

$$\hat{S}(t_{j+1}) = \hat{S}(t_j)e^{(r-\sigma^2/2)T/m+\sigma\sqrt{T/m}Zj}, \tag{2}$$

where Z_j are independent from one another. (Glasserman, 2003)

2.1 The Algorithm

When pricing the barrier option, one first must simulate the price movements of the underlying stock in order to find the expected pay off. The Matlab function created to simulate the estimate and do the pricing is attached in Section 5.1[4] and works as follows:

First, an outer loop is created to represent the amount of simulations. Then, the indicator function, used to give value to the simulations that are in the contract, is defined to be equal 0 for knock-in option and 1 for knock-out). An inner loop is now created to calculate the movements for all the time steps. Inside this loop, we first must find a starting value and do so through an if-condition that separates the first simulation from the rest. Then the algorithm generates a value for the normal random variable $Z_i(t_1)$ and plug this together with the input of S_0 into (2). Next an if-command checks whether the price movement caused the option to be in the contract or not, and assign the corresponding value of the indicator function. From there, this procedure is repeated for $S_i(t_{j+1})$ but now plugging the value of $Z_i(t_1)$ and the simulated value of $S_i(t_1)$ into (2) instead. Next, the inner loop is closed, an if command specifies weather to find the call-price or the put-price, which is found for each simulation and set by the discounted pay off at maturity times the indicator function in order to sort out the ones being in -and out of the contract respectively. Finally, the outer loop is closed and the price at time 0 is set to be the discounted average pay off of all simulations. Let I and B represents the indicator function and the price-barrier respectively, then the above procedure is summarized below.

```
for $i = 1$ to $n$ do
    if Knock-In Option
    set $I = 0$
    elseif Knock-Out Option
    set $I = 1$
    for $j = 0$ to $m - 1$ do
            if $j = 0$
                generate $Z_i(t_1)$
                calculate $S_i(t_1)$ by (2)
                    if Down-and-In Option and $S_i(t_1) \leq B$, or Up-and-In Option and $S_i(t_1) \geq B$
                        $I = 1$
```

[4]The function in the appendix actually includes one more loop in order to generate vectors with different values for different number of simulations, this is for demonstration purposes only. The last value of this vector is the one used for the pricing.

```
                elseif Down-and-Out Option *and* $S_i(t_1) \leq B$, *or* Up-and-Out Option *and* $S_i(t_1) \geq B$
                    $I = 0$
                end if
            else $j \neq 0$
                generate $Z_i(t_{j+1})$
                calculate $S_i(t_{j+1})$ by (2)
                if Down-and-In Option *and* $S_i(t_1) \leq B$, *or* Up-and-In Option *and* $S_i(t_1) \geq B$
                    $I = 1$
                elseif Down-and-Out Option *and* $S_i(t_1) \leq B$, *or* Up-and-Out Option *and* $S_i(t_1) \geq B$
                    $I = 0$
                end if
            end if
    end for
    if Call-Option
    set $C_i = e^{-rT} max\{S_i(T) - K, 0\} \times I$
    elseif Put Option
    set $P_i = e^{-rT} max\{K - S_i(T), 0\} \times I$
    end if
end for
set $V$ by (1)
```

3 Examples

This section provides five examples of pricing barrier options with the Black-Scholes model. All of them has arbitrary values of input[5], where it is assumed, for simplicity, that a month has 30 trading days and that prices fluctuate once per day (giving the same number of time-steps in the simulator as the number of days till maturity). Moreover, the amount of stock-price simulations is 10.000 for each of the examples and the Matlab function and script used for execution are attached in the Appendix.

3.1 Example 1 – Pricing a Down-and-In-Call

Let the inputs to the Black-Scholes model be

$r = 0.02$	$S_0 = \$100$	$B = 95$
$T = 1/4$	$\sigma^2 = 0.12$	$K = 70$

The price of a down-and-in call option with this stock as the underlying turns out to be \$0.0617 and with 95% confidence, the value lies within the interval [\$0.0346 \$0.0888]. The computational time was 18.641742 seconds and Figure 2a demonstrates how the estimate converges towards the true value as the number of simulations increases.

3.2 Example 2 – Pricing a Down-and-Out-Put

Let the inputs to the Black-Scholes model be

$r = 0.02$	$S_0 = \$100$	$B = 88$
$T = 1/4$	$\sigma^2 = 0.12$	$K = 115$

The price of a down-and-out put option with this stock as the underlying turns out to be \$0.0242 and with 95% confidence, the value lies within the interval [\$0.0117 \$0.0368]. The computational time

[5]Barrier options are mostly traded at the over the counter market which makes it difficult to find any data to evaluate this model against.

was 16.026143 seconds and Figure 2b demonstrates how the estimate converges towards the true value as the number of simulations increases.

3.3 Example 3 – Pricing an Up-and-Out-Call

Let the inputs to the Black-Scholes model be

$r = 0.02$	$S_0 = \$100$	$B = 110$
$T = 1/4$	$\sigma^2 = 0.12$	$K = 70$

The price of a down-and-in call option with this stock as the underlying turns out to be \$0.0351 and with 95% confidence, the value lies within the interval [\$0.0186 \$0.0517]. The computational time was 20.661569 seconds and Figure 2c demonstrates how the estimate converges towards the true value as the number of simulations increases.

3.4 Example 4 – Pricing an Up-and-In-Put

Let the inputs to the Black-Scholes model be

$r = 0.02$	$S_0 = \$100$	$B = 110$
$T = 1/4$	$\sigma^2 = 0.12$	$K = 115$

The price of a down-and-in call option with this stock as the underlying turns out to be \$0.1098 and with 95% confidence, the value lies within the interval [\$0.0674 \$0.1523]. The computational time was 16.144229 seconds and Figure 2d demonstrates how the estimate converges towards the true value as the number of simulations increases.

3.5 Example 5 – Pricing an Up-and-In-Put

A similar trial as Example 1 was executed with identical numbers except for that the number of simulations were set to 100.000 rather than 10.000. The computational time increased to 200.918013 seconds while the price became \$0.0090, with 95% confidence interval [\$0.0058 \$0.0123]. Figure 1 in the Appendix demonstrates how the estimate converges towards the true value as the number of simulations increases.

Comments and Conclusion

From Figure 1 and 2 in the Appendix, it is obvious that as the number of simulations (i.e. n) increases, the variance of the distribution tend to become smaller. The smaller variance is of course desirable but it does also increase the computational time of the model, as demonstrated in Example 5. In addition, since the barrier options forces one to use the discretized version of the models' stock-price formulas, small errors will arise causing the result to be biased. By increasing the number of steps (i.e. m) however, this error will decrease, causing the estimate to converge asymptotically towards the true value but will in turn, increase the computational time. In other words, greater accuracy increases the computational time and one must decide at what level the benefits outweighs the costs. Anyhow, simulation is a good method of solving complicated stochastic differential equations and this assignment clearly demonstrates how Monte Carlo methods can be used in order to price more complex derivatives rather than non-complex such for instance a European option. However, there is more to come before I am convinced about the method being a good fit. Firstly, concerning the question of when benefits outweighs the losses, how long time is a reasonable time for execution (i.e. how urgent is it to have the results)? In this report the results were generated in a maximum of less than 4 minutes but running the trials (identically) over and over again gave values that fluctuated significantly between all trials. What if the simulations or time-steps were increased even further or an average of the value of many trials were calculated, could this generate more equal values from execution to execution? I truly suspect that there is a reason for Monte Carlo simulations within the field but more knowledge is forced before I can be convinced about the greatness of Monte Carlo Simulations when pricing financial derivatives.

References

Hull, John C. (2010) *Options, Futures, and other Derivatives*, 7th edt, United States, Pearson.

Glasserman, P. (2003) *Monte Carlo Methods in Financial Engineering*, United States, Springer.

A Appendix

This appendix contains all relevant Matlab scripts that has been created to solve the problems. Section 5.1 covers the first of these while section 5.2 covers the second one. Section 5.3 will cover the examples.

A.1 The Algorithm

Here follows the Matlab function for pricing the down-and-in call option with the Black-Scholes model:

```
% Course: Monte Carlo Simulation
% Author: Jessica Radeschnig
% Assignment 1/2 - Pricing discretely monitored down-and-in call option in
% B-S model
function[Price, upper, lower, moments]= BlackScholes(n,m,Option,Knock,r,T,...
                                           Var,S0,K,B)
                                        % Defines the function with output and
                                          inputs.
                                        % This function can be used by other
                                          scripts


Price = 1:50;    % Makes the output to come in vector form
upper=1:50;          % -||-
lower=1:50;          % -||-
moments=1:50;        % Defines how many different sample sizes to measure


for N=1:50           % Make the output vector represent different sizes of
   simulations

    moments(N)=(n/50)*N;
end
N=1;
h=waitbar(0,'Work in progress, please wait...');% Creates window counting down
   until
                                                % the results of the execution
                                                  are
                                                %
                                                  available



tic                            % Starts calculating the time for calculations
for i = 1:n;                   % To make n simulations
   if Knock==11 || Knock==21;  % Indicator function: Initially one is out of the
    I=0;                       % contract
    elseif Knock==12 || Knock==22;
        I=1;                   % Indicator function: Initially in contract
    end


    for j = 0:m-1;             % To make m timesteps untill maturity
        Z(j+1)=randn(1,1);     % Generates the random normal variable
        dW=sqrt(T/m)*Z(j+1);   % The distribution of the Brownian motion
```

```matlab

        if j==0                 % The S of the first time period must be priced
                                %                        with an explicit input

            S(j+1) = S0*exp((r- Var/2)*(T/m) + sqrt(Var)* dW);    % Defines the
                                                     %   first next stock price

            if all([Knock==11, I==0, S(j+1)<= B]) || all([Knock==21, I==0, S(j+1)>=
               B]);

                                     % Condition for being in the contract is to
                                       fall
                                     %                        or reach to the
                                       barrier
                    I = 1;

            elseif all([Knock==12, I==1, S(j+1)<= B]) || all([Knock==22, I==1, S(j
              +1)>= B])
                                       % Condition for out of the contract is to
                                         fall
                                       %                      or reach to the
                                         barrier
                    I = 0;
            end


        else % All time periods after first: is priced using estimated values of S

            S(j+1) = S(j)*exp((r - Var/2)*(T/m) + sqrt(Var)* dW);  % Simulates all
                                                     % values of S after the first

        if all([Knock==11, I==0, S(j+1)<= B]) || all([Knock==21, I==0, S(j+1)>= B])
          ;

                                     % Condition for being in the contract is to
                                       fall
                                     %                        or reach to the
                                       barrier
                    I = 1;
            end

            if all([Knock==12, I==1, S(j+1)<= B]) || all([Knock==22, I==1, S(j+1)>=
               B])
                                       % Condition for out of the contract is to
                                         fall
                                       %                      or reach to the
                                         barrier
                    I = 0;
            end
        end

    end  % End j-loop


    if i==moments(N)
        if Option==1;  %Creates N different sample sizes (# of simulations)
        C(i) = exp(-r*T)* max((S(m-1)-K), 0)*I; % Vector with call price for all
                                            %                        iteration
        Price(N)=sum(C(1:i))/i; % Calculates the price of the down-and-in call

        s=std(C(1:i)); % Gives the standard deviation of each sample

        upper(N) = Price(N) + 1.96*s/(sqrt(i));% The upper confidence inteval
                                                   %        bound for the samples

        lower(N) = Price(N) - 1.96*s/(sqrt(i));% The lower confidence inteval
                                              %        bound for the samples
        N=N+1;

        elseif Option==2;     %Creates N different sample sizes (# of simulations)
        P(i) = exp(-r*T)* max((K - S(m-1)), 0)*I; % Vector with put price for all
                                                   %                     iterations

        Price(N)=sum(P(1:i))/i; % Calculates the price of the down-and-in call

        s=std(P(1:i)); % Gives the standard deviation of each sample

        upper(N) = Price(N) + 1.96*s/(sqrt(i));% The upper confidence inteval
                                                  %         bound for the samples

        lower(N) = Price(N) - 1.96*s/(sqrt(i));% The lower confidence inteval
                                              %         bound for the samples
        N=N+1;
        end
    end



waitbar(N/50); % Now, the count down in the waitbar should proceed

end       % End i-loop

close(h); % Not untill here, the count-down (in window) is finnished

toc       % Stops calculating the time for calculations

end       % Ends the function
```

A.2 The Examples

Here follows the Matlab file used for executing the results in the Examples.

```
% Course: Analytical Finance II
% Author: Jessica Radeschnig
% Seminar: Pricing discretely monitored Barrier Options in B-S Model
% Main File, Example
clear
clc


n=10000;
m=90;

Matlab asks for the model inputs
Option=input('What kind of option is it? Press "1" for call and "2" for put');
Knock=input('What kind of option is it? Press "11" for down and in, "12" for down
  and out, "21" up and in, or "22" for up and out.');
r=input('What is the interest rate?');
T=input('What is the time to maturity?');
S0=input('What is the initial stock price?');
Var=input('What is the variance?');
B=input('What is the Barrier level?');
K=input('What is the strike price?');


[Price, upper, lower, moments]= BlackScholes(n,m,Option,Knock,r,T,Var,S0,K,B);

if Option==1;
fprintf('The Price for the Call-Option is %3.4\n',Price)
Price(1, 50)        % Gives the price (from the value generated by n=100000)
fprintf('The Upper Bound is %3.4\n' ,upper(1, 50))
upper(1, 50)      % Gives the upper bound (from the value generated by n=100000)
fprintf('The Lower Bound is %3.4\n' ,lower(1, 50))
lower(1, 50)      % Gives the lower bound (from the value generated by n=100000)

plot(moments, upper, moments, Price, moments, lower)  % Plots the average option
                                    % price and confidence interval corresponding to
                                      all
```

```
                                          %                the different amount of sample
                                            sizes
title('Simulating Barrier Option Price using Black-Scholes Model')
xlabel('Number of Simulations');
ylabel('Call-Option Price');
legend('Upper Bound','Call-Price', 'Lower Bound')
grid on;
elseif Option==2;
        fprintf('The Price for the Put-Option is %3.4\n',Price)
        Price(1, 50)         % Gives the price (from the value generated by n
          =100000)
        fprintf('The Upper Bound is %3.4\n' ,upper(1, 50))
        upper(1, 50)       % Gives the upper bound (from the value generated by n
          =100000)
        fprintf('The Lower Bound is %3.4\n' ,lower(1, 50))
        lower(1, 50)       % Gives the lower bound (from the value generated by n
          =100000)

plot(moments, upper, moments, Price, moments, lower)  % Plots the average option
                                          % price and confidence interval corresponding to
                                            all
                                          %                the different amount of sample
                                            sizes
title('Simulating Barrier Option Price using Black-Scholes Model')
xlabel('Number of Simulations');
ylabel('Put-Option Price');
legend('Upper Bound','Put-Price', 'Lower Bound')
grid on;
end
```

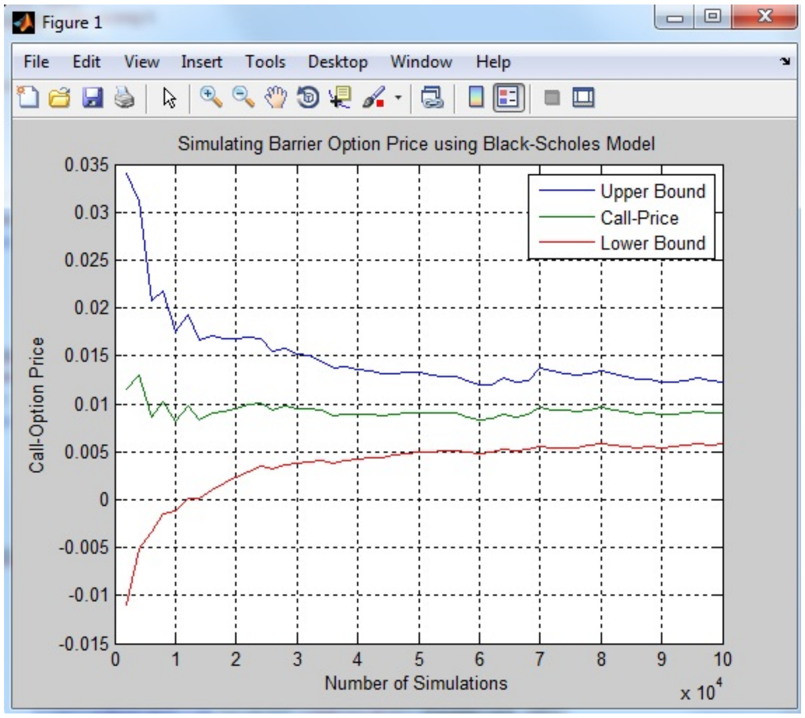
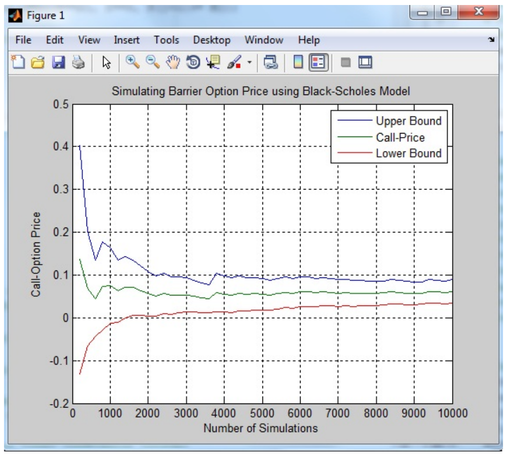
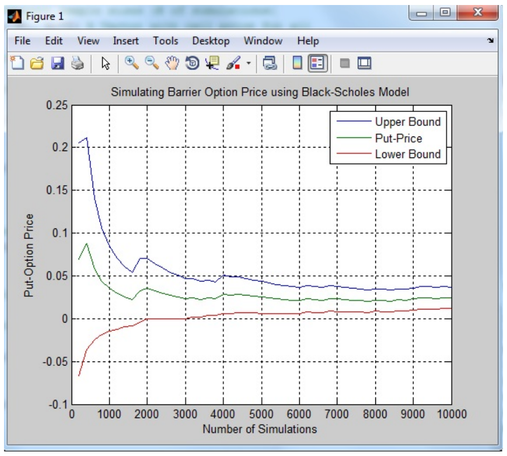


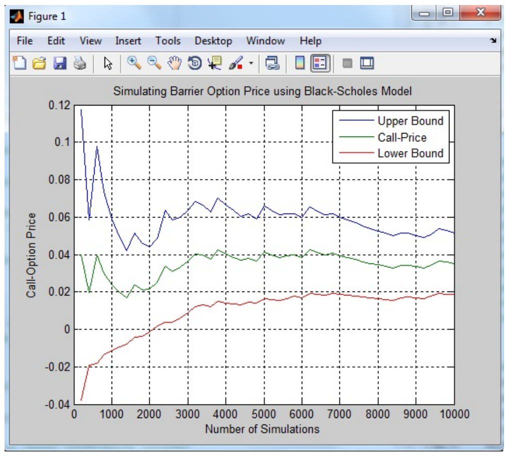
Figure 1: Estimating using the Black-Scholes Model – Increasing the simulations
The graph shows how the variance of the estimated option price decreases as the number of iterations (simulations) increases. Total number of simulations is 100.000.



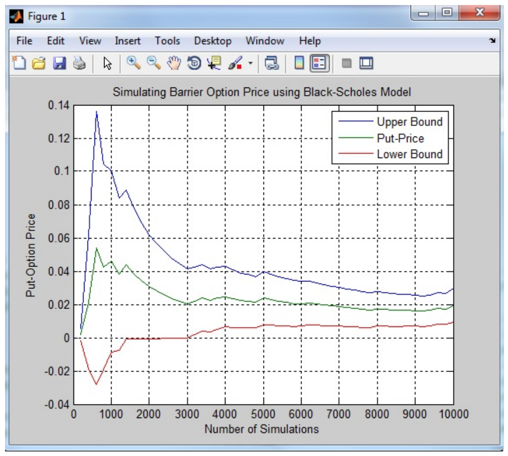
(a) Down-and-In Call



(b) Down-and-Out Put



(c) Up-and-Out Call



(d) Up-and-In Put

Figure 2: Estimating using the Black-Scholes Model
The graphs shows how the variance of the estimated option price decreases as the number of iterations (simulations) increases. Total number of simulations is 10.000.