Uppsala University

Computational Finance

Monte Carlo Methods vs. Finite Difference Methods to Price Options

Authors:

Jim Lindberg

Alessandro Piccolo

Rui Hao

October 3, 2016

UPPSALA
UNIVERSITET

1 Introduction

The purpose of this report is to compare two different numerical methods for pricing a European call option when the underlying stock follows the CEV model as shown below.

$$dS_t = rS_tdt + \sigma S_t^{\gamma} dW_t \tag{1}$$

The two chosen methods being compared are:
I. Monte Carlo method based on Euler discretization with antithetic variates
II. Finite difference method (FDM) based on Euler backward discretization in time which is an implicit method and builds upon the following adjusted Black-Scholes equation.

$$\begin{aligned}&\frac{\partial v}{\partial t} + rs\frac{\partial v}{\partial s} + \frac{1}{2}\sigma^2 s^{2\gamma}\frac{\partial^2 v}{\partial s^2} - rv = 0\\ &v(T,s) = \phi(s)\end{aligned} \tag{2}$$

2 Numerical Methods

The essential ideas and algorithms of three different types of numerical methods are presented below.

2.1 Lattice Methods

The lattice method (binomial method) for pricing options is a discrete time model based on the assumption that there is a certain probability p of the asset price going upward or downward.

The algorithm consists of three steps:
1) Create the binomial price tree.
2) Compute the option value at each final node.
3) Compute option values at earlier nodes.

A simple illustration of the binomial tree is shown below.

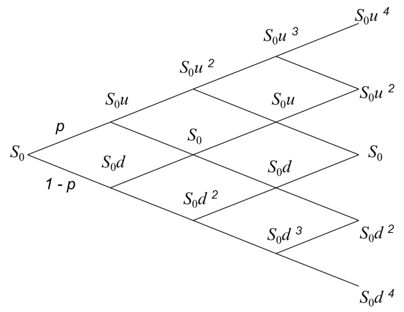


Figure 1: Binomial tree for option pricing

2.2 Monte Carlo Methods

Monte Carlo methods are computational algorithms that rely on repeated random sampling to obtain numerical results. In the case of option pricing, it is based on Euler discretization of the stochastic differential equation 1. The Algorithm is as follows.

For each MC simulation (sample path) we
1) compute the stock price at maturity with Euler method, and then
2) compute the value of the option at maturity using the pay-off function which is $(S_T - K)^+$ for a European call.

Then we take the mean of these different simulated option values at maturity and finally discount back to today's value.

2.3 Finite Difference Methods

Three typical finite difference methods for option pricing are Euler forward, Euler backward and Crank-Nicolson. The first is an explicit method while the latter two are implicit.

For Euler forward, the derivatives in the Black-Scholes equation 2 are approximated in the following way, which gives 1st order in time and 2nd order in space.

$$\begin{aligned}\frac{\partial v}{\partial t}(s_j, t_n) &= \frac{v_j^n - v_j^{n-1}}{\Delta t} \\ \frac{\partial v}{\partial s}(s_j, t_n) &= \frac{v_{j+1}^n - v_{j-1}^n}{2\Delta t} \\ \frac{\partial^2 v}{\partial s^2}(s_j, t_n) &= \frac{v_{j+1}^n - 2v_j^n + v_{j-1}^n}{\Delta s^2}\end{aligned} \tag{3}$$

By reordering equation 2 after substituting the derivatives one would get the following equation after even further reordering the variables.

$$v_j^{n-1} = v_j^n + \frac{\Delta t}{\Delta s^2}\frac{1}{2}\sigma^2 s_j^{2\gamma}(v_{j+1}^n - 2v_j^n + v_{j-1}^n) + \frac{\Delta t}{2\Delta s} r s_j (v_{j+1}^n - v_{j-1}^n) - r\Delta t v_j^n \tag{4}$$

The boundary conditions for a European call option are

$$v(T, s) = (S_T - K)^+ \tag{5}$$

$$v(t, 0) = 0 \tag{6}$$

$$v(t, s_{max}) = s_{max} - Ke^{-r(T-t)} \tag{7}$$

The explicit method is not universally stable as the implicit methods but has to hold equation 8. However the explicit method has a lesser computational cost than the implicit method.

$$\frac{\Delta t}{\Delta s^2} \leq \lambda \tag{8}$$

For Euler backward, we will get the following equation instead of equation 4. The approximation is still 1st order in time and 2nd order in space.

$$\begin{aligned} v_j^n &= v_j^{n-1} - \frac{\Delta t}{\Delta s^2}\frac{\sigma^2}{2}s_j^{2\gamma}(v_{j+1}^{n-1} - 2v_j^{n-1} + v_{j-1}^{n-1}) - \frac{\Delta t}{2\Delta s}rs_j(v_{j+1}^{n-1} - v_{j-1}^{n-1}) - r\Delta t v_j^{n-1} \\ &= v_j^{n-1} - \Delta t L v_j^{n-1} \end{aligned} \tag{9}$$

For Crank-Nicolson, we will have the following equation. The obtained approximation has 2nd order in time and 2nd order in space.

$$v_j^n + \frac{1}{2}\Delta t L v_j^n = v_j^{n-1} - \frac{1}{2}\Delta t L v_j^{n-1} \tag{10}$$

3 Results

The plots in this section show the main results from the implementations of the Monte Carlo method and the finite difference method.

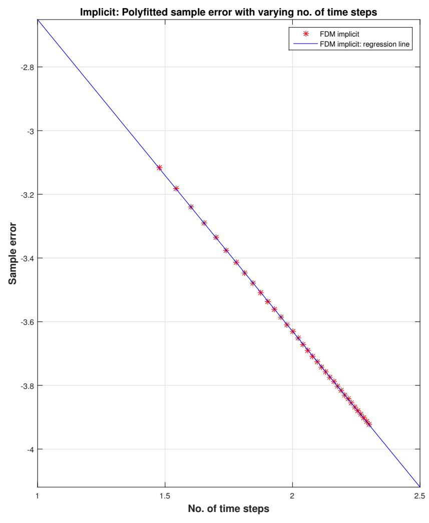
Input parameters are $r = 0.1$, $\sigma = 0.25$, $T = 0.5$, $\gamma = 1$, number of stock prices (space steps) $M = 1000$ and number of time steps $N = 10000$.

3.1 Convergence Rate

Monte Carlo methods have a theoretical convergence rate of 0.5. Figure 4 shows a slope of normal MC -0.47 and antithetic -0.47 which is close to 0.5. The time step size was set to 500 with the number of sample paths varying from $1 - 1 \times 10^6$. However as shown in figure 2 the time step after *time step* $= 500$ should not affect the sample error too much. The antithetic versus normal MC plot shows that they almost have the same slope. One should keep in mind that the antithetic is actually improving the bias.

For FDM, the implicit Euler backward is 1st order in time and 2nd order in space, which should correspond to slope -1 if error is plotted against the number of time steps in log scale, and slope -2 if error is plotted against the number of space steps (price points) in log scale. Figure 3 has a slope of -1.88 which is close to -2, the small deviation is probably due to the fact that the time step is not infinite. Left plot of figure 2 shows a slope close to -0.98, any deviation is due to the fact that stock price step is not infinite.

All plots are in log scale and the values on the x and y axes are in terms of 10^x.



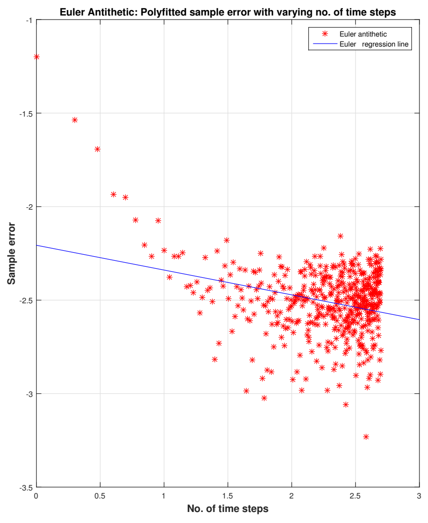


Figure 2: Shows the convergence rate for FDM to the left with a slope equal to -0.97874 and Monte Carlo Euler antithetic to the right with a slope equal to -0.13. For the Monte Carlo plot to the right, the number of sample paths is $1e5$.

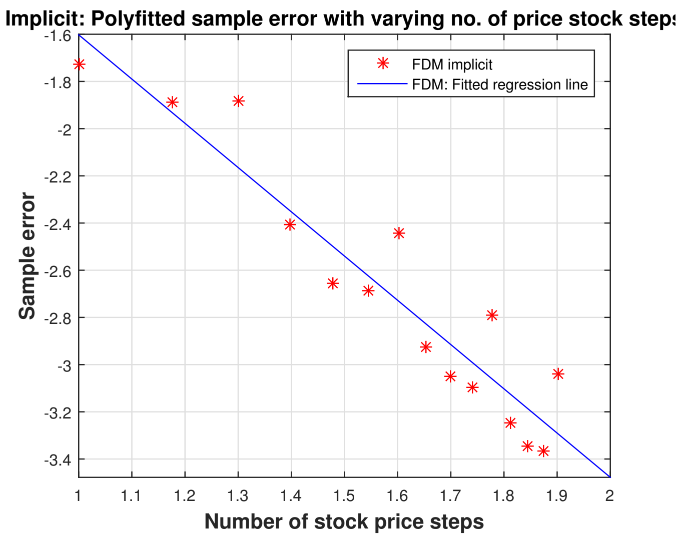


Figure 3: Shows the FDM implicit sample error vs. the number of stock prices with a slope of -1.8753.

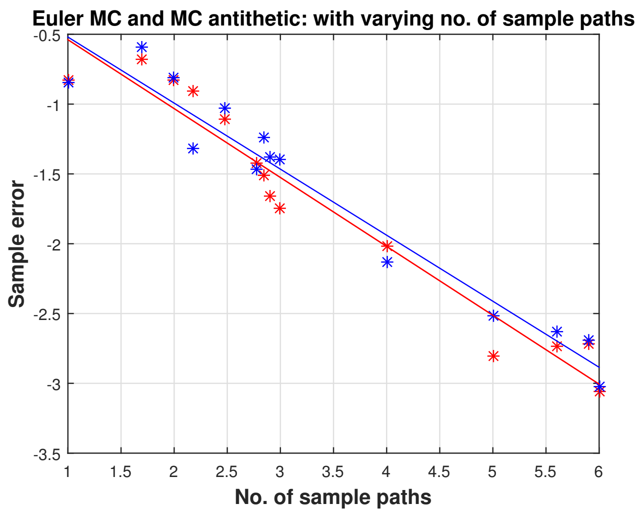


Figure 4: Shows the Monte Carlo normal and antithetic variant of the sample error vs. the number of sample paths, normal MC (blue line) has a slope of 0.47 meanwhile the antithetic has 0.49 (red line). The fixed time step was set to 500.

3.2 Monte Carlo Method

The absolute error and execution time for the Monte Carlo Euler antithetic method are presented below. Variable M is the number of space steps and N is the number of time steps.

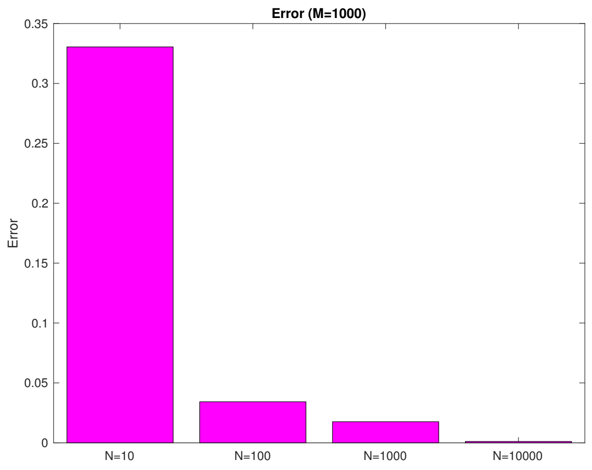


Figure 5: Shows the error for MC method for different values of N

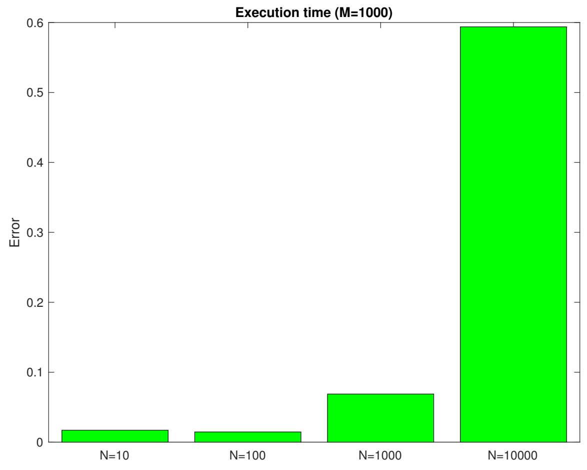


Figure 6: Shows the execution time for MC method for different values of N

3.3 Finite Difference Method

Absolute error and execution time for FDM implicit and explicit methods as well as a stability test for the explicit method are presented below.

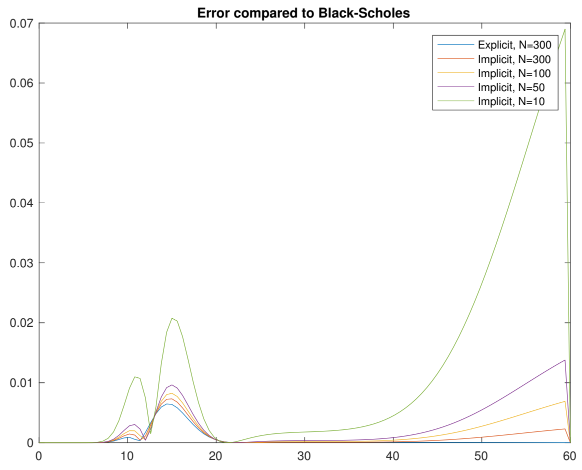


Figure 7: Shows the error for implicit and explicit FDM methods

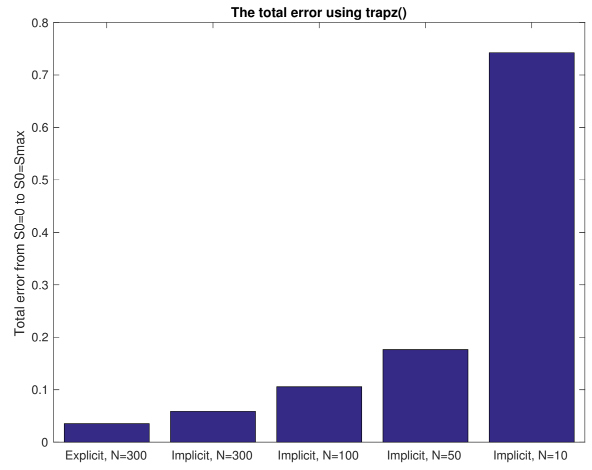


Figure 8: Shows the total error for these methods

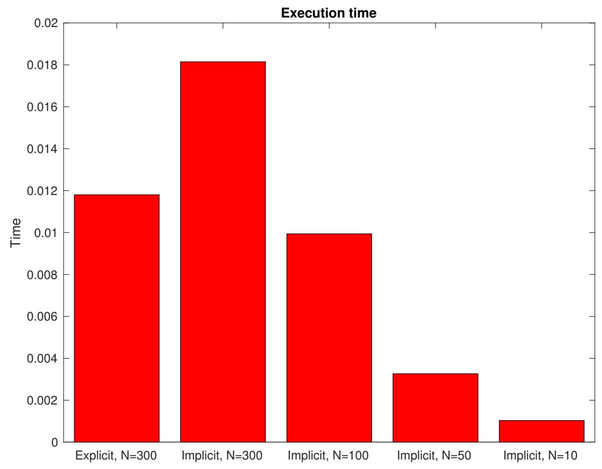


Figure 9: Shows the execution time for these methods

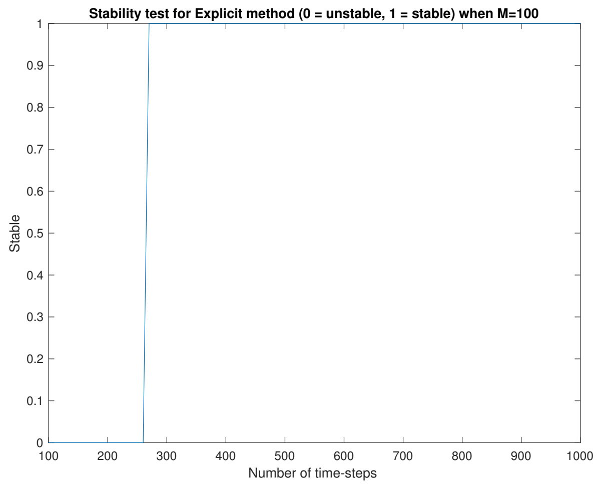


Figure 10: Shows the breaking point where the explicit method becomes stable. This happens when the number of time-steps is approximately 2.5 larger than the number of space-steps

4 Implementation Aspects: Monte Carlo vs FDM

The implementation of the Monte Carlo method is easier, since it is basically only stepping in time using equation 1, meanwhile the FDM method requires an approximation of the Black-Scholes equation using finite difference approximations of the derivatives shown in equation 3.

5 Accuracy Aspects: Monte Carlo vs FDM

As the plots in the result section show, both MC and FD are capable of producing low errors. However, the MC method has a longer execution time for producing the same level of error as the FDM. This is supported by the theory that the MC method converges at a rate of $\sqrt{N}$ whereas the FDM method converges at a rate of N in time and M^2 in space.

6 Several Underlying Assets

The pricing of options on several underlying assets can be achieved analogously. The following discussions explain the necessary adjustments for these two methods and the problems they can encounter. One conclusion is that the Monte Carlo method is preferred when pricing options with more than three underlying assets.

6.1 Monte Carlo

If one assumes that the only computational effort in the Monte Carlo method is computing the price(s) for the underlying asset(s) at each time step, and that the model used to compute these prices are the same for all assets, then adding another underlying asset should theoretically mean that the execution time increases linearly for every added asset. As Figure 11 shows, the Monte Carlo method is not too affected by another underlying asset.

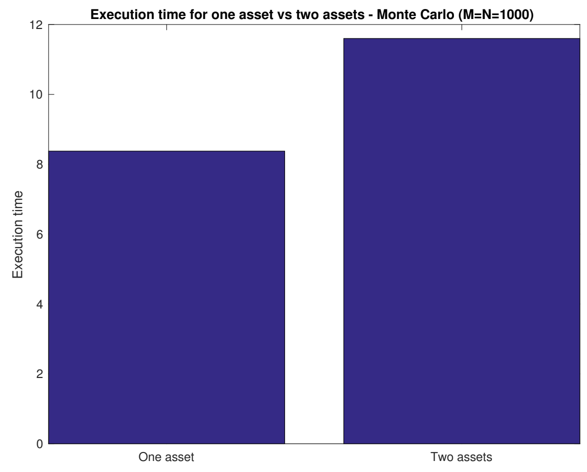


Figure 11: Execution time for a Monte Carlo solver, for one and two assets. In the case of two assets, the option was on the equally weighted portfolio of the two underlying assets. Their respective prices was calculated using Geometric Brownian Motion

6.2 Finite Difference Method

For a large number of underlying assets, the FDM suffers from the curse of dimension. For one underlying asset, the number of space points that need to be updated for each time step is simply M, resulting in $M \times N$ space points in total for all time steps. But for two underlying assets, when using the same price-step size for both assets, the number of space points to be calculated, in the now 3-dimensional grid, is $M \times M \times N$. The number of space points, p, that must be updated at each time step can be generalized into the following formula where M is the number of price steps, N is the number of time steps and n stands for the number of underlying assets.

$$p = M^n N \tag{11}$$

For relatively large M and N, which is required to get an accurate result, equation 11 shows that the number of points to be calculated increases rapidly when another asset is added, thus posing a much higher memory demand and causing the method to be less efficient than Monte Carlo.

Equation 12 shows the corresponding PDE used to price options with several underlying assets.

$$\frac{\delta V}{\delta t} + \frac{1}{2}\sum_{i,j=1}^{n} \rho_{ij}\sigma_i\sigma_j S_i S_j \frac{\delta^2 V}{\delta S_i \delta S_j} + \sum (r-\delta_i) S_i \frac{\delta V}{\delta S_i} - rV = 0 \tag{12}$$

7 Computation of the Greek Δ

For the greek $\Delta = \frac{\partial v}{\partial s}$, the analytical solution is simply $N(d1)$ (Seydel, 2009, p.357 & Björk, 2009), where N is the standard normal cumulative distribution function and d_1 is the same as in the Black-Scholes solution, i.e.

$$d_1 = \frac{\log \frac{S}{K} + (r + \frac{\sigma^2}{2})(T-t)}{\sigma\sqrt{T-t}} \tag{13}$$

For the FDM, $\frac{\partial v}{\partial s}$ is approximated with the Euler centered scheme (see eq.3). So the greek Δ can simply be computed with the same formula. For the Monte Carlo method, one can choose any approximation scheme for calculating the derivative. Euler centered was chosen again to create a fair comparison. The computed results for Δ are shown below. Here we used 100 space steps, 1000 time steps and 10000 Monte Carlo sample paths.

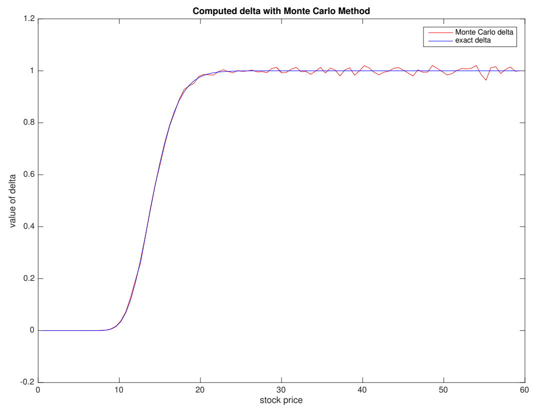


Figure 12: Shows computed Δ vs. stock price with Monte Carlo method

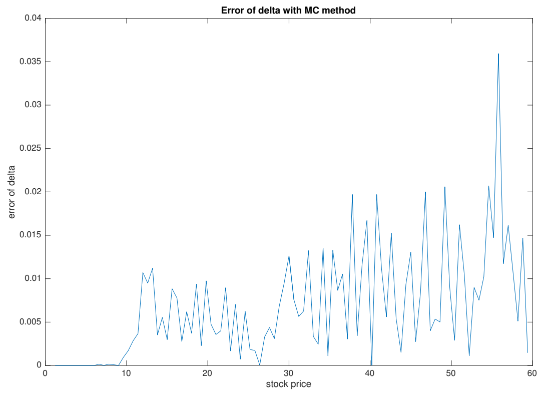


Figure 13: Shows error of Δ vs. stock price with Monte Carlo method

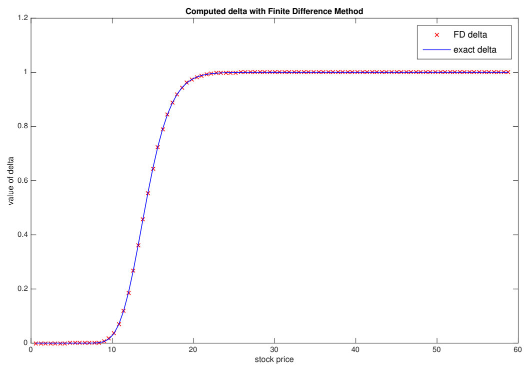


Figure 14: Shows computed Δ vs. stock price with implicit FDM

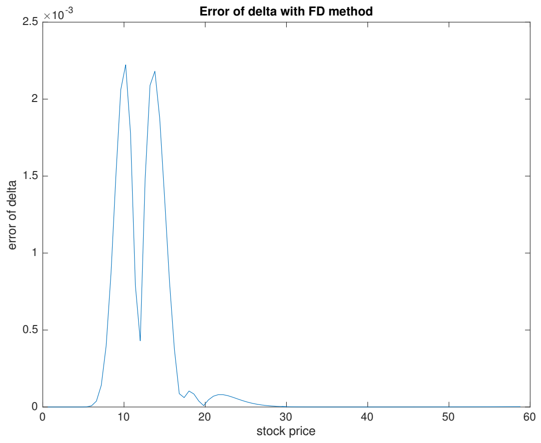


Figure 15: Shows error of Δ vs. stock price with implicit FDM

One can see that using the same Euler centered scheme to approximate the derivative Δ, the FDM gives a much smoother plot as well as a more accurate value than Monte Carlo. Besides, there is a big difference between the execution time for these two methods, with Monte Carlo taking around 42 seconds and FDM taking only 0.27 seconds.

The error obtained from the Monte Carlo method can be reduced as the number of sample paths increases, but at a cost of even longer execution time. The results from 100 000 Monte Carlo sample paths are shown below. The execution time was 438 seconds.

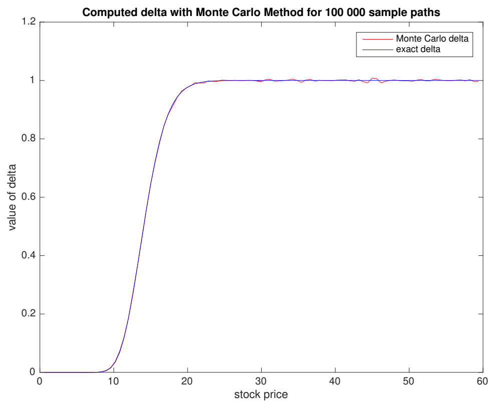


Figure 16: Shows computed Δ vs. stock price with Monte Carlo for 100 000 sample paths.

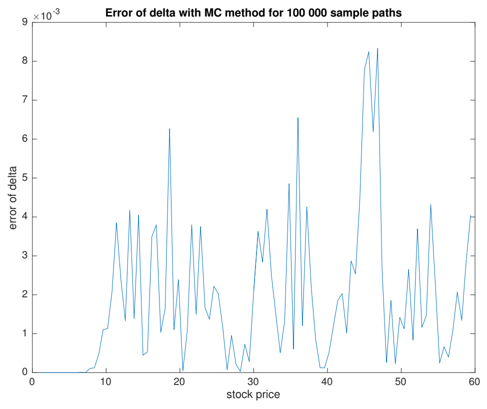


Figure 17: Shows error of Δ vs. stock price with Monte Carlo for 100 000 sample paths.

8 Conclusion

For options with less than 3 underlying assets, finite difference methods are competitive to Monte Carlo methods and may offer faster execution; but for higher dimensions, Monte Carlo methods are the optimal choice (Seydel, 2009, p.279).

9 References

Björk, T. 2009. Arbitrage theory in continuous time. Oxford: Oxford University Press.
Hirsa, A. 2013. Computational methods in finance. Boca Raton, FL: CRC Press.
Seydel, R. 2009. Tools for computational finance. Berlin: Springer.

Appendices

A Code for section 3.1 Convergence Rate

A.1 Main program for convergence rate

```
%{
  Convergence rate for Finite difference methods vs for Euler antithetic
  to price a European Options

  By Alessandro Piccolo, Jim Lindberg and Rui Hao
%}
clear all; close all;

%% General input parameters
K        = 15;              % Strike price
r        = 0.1;             % Interest rate
sigma    = 0.25;            % Diffusion paramter
T        = 0.5;             % Final time
gamma    = 1;               % Elasticity variable

s_min    = 0;               % Min stock price
s_max    = 4*K;             % Max stock price

M        = 1000;            % Number of stock prices (steps)
N        = 10000;           % Number of time steps

Nvec     = 30:5:200;        % Time data points for convergence rate
Mvec     = 10:5:80;         % Stock price data points for convergence rate
lengthNvec = length(Nvec);
lengthMvec = length(Mvec);

%% Extra input parameters for Monte-Carlo method (Euler antithetic)
S0            = 14;         % Initial stock price
nrSimulations = 5;          % Number of Monte-Carlo simualtions
sample_path   = 1e5;        % Number of sample paths <-- changed from 5 000
numN          = [1:500];    % Number of time steps <--- change to 500
numN_length   = length(numN);
errN(numN_length, nrSimulations) = 0;

% No. of sample paths
time_steps = 500; % <-- changed from 200
numM  = [1e1, 50, 1e2, 150, 300, 600, 700, 800, 1e3, 1e4, 1e5,4e5,8e5,1e6];
numM_length = length(numM);
errM(numM_length, nrSimulations) = 0;    % MC antithetic
errM_mc(numM_length, nrSimulations) = 0; % MC

%% FDM: Accuracy, error - Varying time steps values, N
tmp_error_N = zeros(1, M+1);
h = waitbar(0, 'FDM implicit: Loading please wait...');
for i = 1:lengthNvec
    [v,~,~,~,svec] = implicit(K,r,sigma,T,gamma,s_min,s_max,M,Nvec(i));
    for j = 1:M+1
        v_exact        = bsexact(sigma, r, K, T, svec(j));
        tmp_error_N(j) = abs(v(j,1)-v_exact); % Calculate error
    end
    error_mean_N(i) = mean(tmp_error_N);
    waitbar(i/lengthNvec, h);
end
close(h);

p_error_Nvec = polyfit(log10(Nvec),log10(error_mean_N),1); % linear approx

figure(1)
subplot(1,2,1)
plot(log10(Nvec), log10(error_mean_N), 'r*');
l_line = lsline; set(l_line(1),'color','b')
str = 'Implicit: Polyfitted sample error with varying no. of time steps';
hlt = title(str);
hlx = xlabel('No. of time steps');
hly = ylabel('Sample error');
set(hly,'FontSize',13,'FontWeight', 'bold');
set(hlx,'FontSize',13,'FontWeight', 'bold');
set(hlt,'FontSize',13,'FontWeight', 'bold');
legend('FDM implicit','FDM implicit: regression line')
grid on

display(['FDM implicit: Varying time step - Slope = ' ...
    num2str(p_error_Nvec(1))]);

%% Euler Antithetic: Discretization error for varying number of time steps
k = 1;
for time_step = numN
    for i = 1:nrSimulations
        Z = randn(sample_path,time_step);
        [~,errN(k,i)] = mc_antithetic(T,S0,K,r,sigma,gamma,time_step,...
            sample_path,Z);
    end
    k = k+1;
end

% Polyfit to make linear approximation of data
p_mc_a_time_step = polyfit(log10(numN),log10(mean(errN')),1);

figure(1)
subplot(1,2,2)
plot(log10(numN), log10(mean(errN')), 'r*');
l_line = lsline; set(l_line(1),'color','b')
str = ['Euler Antithetic: Polyfitted sample error ' ...
    'with varying no. of time steps'];
hlt = title(str);
hlx = xlabel('No. of time steps');
hly = ylabel('Sample error');
set(hly,'FontSize',13,'FontWeight', 'bold');
set(hlx,'FontSize',13,'FontWeight', 'bold');
set(hlt,'FontSize',13,'FontWeight', 'bold');
legend('Euler antithetic','Euler   regression line')
grid on

display(['Euler antithetic: Varying time step - Slope = ' ...
    num2str(p_mc_a_time_step(1))]);
```

```
%% Accuracy, error - Varying price option steps values, M
for i = 1:lengthMvec
    [v,~,~,~,svec] = implicit(K,r,sigma,T,gamma,s_min,s_max,Mvec(i),N);
    tmp_error_M = zeros(1, Mvec(i)+1);
    for j = 1:Mvec(i)+1
        v_exact        = bsexact(sigma, r, K, T, svec(j));
        tmp_error_M(j) = abs(v(j,1)-v_exact); % Calculate error
    end
    error_mean_M(i) = mean(tmp_error_M);
end

p_error_Mvec = polyfit(log10(Mvec),log10(error_mean_M),1); % linear approx

figure(3)
plot(log10(Mvec), log10(error_mean_M), 'r*');
l_line = lsline; set(l_line(1),'color','b')
str = ['Implicit: Polyfitted sample error with varying' ...
    ' no. of price stock steps'];
hlt = title(str);
hlx = xlabel('Number of stock price steps');
hly = ylabel('Sample error');
set(hly,'FontSize',13,'FontWeight', 'bold');
set(hlx,'FontSize',13,'FontWeight', 'bold');
set(hlt,'FontSize',13,'FontWeight', 'bold');
legend('FDM implicit','FDM:   regression line')
grid on

display(['Varying stock price step - Slope = ' num2str(p_error_Mvec(1))]);

%% Euler antithetic: Sample error for varying number of sample paths
%  And Normal Monte-Carlo (MC)
k = 1;
for sample_paths = numM
    for j = 1:nrSimulations
        Z = randn(sample_paths, time_steps);
        [~,errM(k,j)] = mc_antithetic(T,S0,K,r,sigma,gamma,time_steps,...
            sample_paths,Z);
        [~,errM_mc(k,j)] = mc(T,S0,K,r,sigma,gamma,time_steps, ...
            sample_paths,Z);
    end
    k = k+1;
end

% Polyfit to make linear approximation of data
p_mc_a_sample_path = polyfit(log10(numM),log10(mean(errM')),1);
p_mc_sample_path = polyfit(log10(numM),log10(mean(errM_mc')),1);

figure(4)
subplot(1,2,1)
plot(log10(numM), log10(mean(errM')), 'r*');
l_line = lsline; set(l_line(1),'color','r')
grid on
hold on

display(['Euler antithetic: Varying sample path - Slope = ' ...
    num2str(p_mc_a_sample_path(1))]);

figure(4)
```

```
subplot(1,2,2)
plot(log10(numM), log10(mean(errM_mc')), 'b*');
l_line = lsline; set(l_line(1),'color','b')
str = ['Euler MC and MC antithetic: with varying no. of sample paths'];
hlt = title(str);
hlx = xlabel('No. of sample paths');
hly = ylabel('Sample error');
set(hly,'FontSize',13,'FontWeight', 'bold');
set(hlx,'FontSize',13,'FontWeight', 'bold');
set(hlt,'FontSize',13,'FontWeight', 'bold');
grid on

display(['Euler MC (normal): Varying sample path - Slope = ' ...
    num2str(p_mc_sample_path(1))]);
```

A.2 Implicit finite difference method function

```
function [v,ds,dt,tvec,svec] = implicit(K,r,sigma,T,gamma,s_min,s_max,M,N)
% Strike price, K
% Interest rate, r
% Diffusion paramter, sigma
% Final time, T
% Elasticity variable, gamma
% Min stock price, s_min
% Max stock price, s_max
% Number of stock prices (steps), M
% Number of time steps, N

ds      = s_max/M;
dt      = T/N;
tvec    = 0:dt:T;          % Length = N+1
svec    = s_min:ds:s_max; % Length = M+1

v(M+1, N+1) = 0;                          % Final value of option matrix
v(:, N+1)   = max(svec-K,0);              % Final conditions
v(M+1, :)   = s_max-K*exp(-r*(T-tvec)); % Final Boundary condition

%% Creating A and B matrix
alpha = 0.5*(sigma^2*svec.^(2*gamma))/ds^2;
beta  = (r*svec)/(2*ds);

c = -beta + alpha;
b = -2*alpha - r;
a = beta + alpha;

% Update A matrix in boundary
a(1) = 0;              % Set to 0 in order to satisfy, Final BC (important)
A    = diag(b)+diag(c(2:M+1),-1) + diag(a(1:M),1);
A    = -A*dt+eye(M+1); % Update A with time step disc

%% Calculate v-vector and update v-matrix
for t_i = N:-1:1 % Iterating backwards, since t+1 values are known
    % Create a vector B for the BC, basically zeros except last value
    B(M,1) = a(M)*dt*(s_max-K*exp(-r*(T-tvec(t_i))));
    v(1:M,t_i) = A(1:M, 1:M)\(v(1:M, t_i+1) + B);
end
```

A.3 Monte-Carlo antithetic function

```
function [v,err] = mc_antithetic(T,S0,K,r,sigma,gamma,M,N,Z)
% This function calculates the price of a European call option using
% Euler's method and the following stock dynamics:
%      dS(t) = r*S(t)*dt + sigma*S(t)^gamma*dW(t)
%
% T     - Time to maturity
% M     - Number of time teps
% N     - Number of paths
% S0    - Initial stock price
% K     - Strike price
% r     - Risk-free interest rate
% sigma - Volatility
% gamma -  Elasticity

dt = T/M;               % Time steps
t = 0:dt:T;             % Time vector

S(N,1) = 0;
S(:,1) = S0;

% Stepping of Monte-Carlo, updating all sample paths simultaniously
for i = 1:M
    dw1 = Z(:,i)*sqrt(dt);
    dw = dw1(1:N/2,1);
    S = S + r*S*dt + sigma*(S.^gamma).*[dw;-dw];
end

VT = max(S-K,0); % Pay off function
E = mean(VT);
v = exp(-r*T)*E;

V0true = bsexact(sigma,r,K,T,S0);
err = abs(V0true - v);

end
```

A.4 Monte-Carlo function

```
function [v,err] = mc(T,S0,K,r,sigma,gamma,M,N,Z)
%T     - Time to maturity
%M     - Number of time teps
%N     - Number of paths
%S0    - Initial stock price
%K     - Strike price
%r     - Risk-free interest rate
%sigma - Volatility
%gamma - Elasticity

dt = T/M;               % Time steps
t = 0:dt:T;             % Time vector

S(N,1) = 0;
S(:,1) = S0;
```

```
for i = 1:M
    dw = Z(:,i)*sqrt(dt);
    S = S + r*S*dt + sigma*(S.^gamma).*dw;
end

VT = max(S-K,0);
E = mean(VT);
v = exp(-r*T)*E;

V0true = bsexact(sigma,r,K,T,S0);
err = abs(V0true - v);
end
```

A.5 Analytic option price solver function

```
function sol = bsexact(sigma, r, E, T, s);

d1 = ( log(s/E) + (r + 0.5*sigma^2)*T )/(sigma*sqrt(T));
d2 = d1 - sigma*sqrt(T);

F = 0.5*s*(1+erf(d1/sqrt(2))) - exp(-r*T)*E*0.5*(1+erf(d2/sqrt(2)))';

sol = F;
```

B Code for section 3.2 Monte Carlo Method

```
clear all

S0=14; K=15; r=0.1; sigma=0.25; T=0.5; gamma=1;
M=1000; N=100;

i = 1;
for N = [10, 100, 1000, 10000]
    tic
    error(i) = abs(price_european_antithetic(S0, K, r, sigma, T, gamma, M/2, ...
        N)-bsexact(sigma, r, K, T, S0))
    time(i) = toc
    i = i+1;
end

figure();
bar(error, 'm');
title('Error (M=1000)')
ylabel('Error')
set(gca,'xticklabel',{'N=10', 'N=100', 'N=1000', 'N=10000'})

figure();
bar(time, 'green');
title('Execution time (M=1000)')
ylabel('Error')
set(gca,'xticklabel',{'N=10', 'N=100', 'N=1000', 'N=10000'})
```

C Code for section 3.3 Finite Difference Method

```
clear all;
S=0; K=15; r=0.1; sigma=0.25; T=0.5; gamma=1;
M=100; N=300;

tic
[stock_price, time, option_price_expl] = price_european_explicit(S, K, r, sigma, T, ...
    gamma, N, M);
time_expl = toc;

tic
option_price_impl_300 = price_european_implicit(S, K, r, sigma, T, gamma, N, M);
time_impl_300 = toc;

N=100;
tic
option_price_impl_100 = price_european_implicit(S, K, r, sigma, T, gamma, N, M);
time_impl_100 = toc;

N=50;
tic
option_price_impl_50 = price_european_implicit(S, K, r, sigma, T, gamma, N, M);
time_impl_50 = toc;

N=10;
tic
option_price_impl_10 = price_european_implicit(S, K, r, sigma, T, gamma, N, M);
time_impl_10 = toc;

for i = 1:length(stock_price)
    black_scholes(i) = bsexact(sigma, r, K, T, stock_price(i));
end

figure()
plot(stock_price, abs(black_scholes-option_price_expl));
hold all
plot(stock_price, abs(black_scholes-option_price_impl_300));
plot(stock_price, abs(black_scholes-option_price_impl_100));
plot(stock_price, abs(black_scholes-option_price_impl_50));
plot(stock_price, abs(black_scholes-option_price_impl_10));
title('Error compared to Black-Scholes')
legend('Explicit, N=300', 'Implicit, N=300', 'Implicit, N=100', 'Implicit, N=50', ...
    'Implicit, N=10')

total_error_expl = trapz(stock_price, abs(black_scholes-option_price_expl));
total_error_impl_300 = trapz(stock_price, abs(black_scholes-option_price_impl_300));
total_error_impl_100 = trapz(stock_price, abs(black_scholes-option_price_impl_100));
total_error_impl_50 = trapz(stock_price, abs(black_scholes-option_price_impl_50));
total_error_impl_10 = trapz(stock_price, abs(black_scholes-option_price_impl_10));

errors(1) = total_error_expl;
errors(2) = total_error_impl_300;
errors(3) = total_error_impl_100;
errors(4) = total_error_impl_50;
errors(5) = total_error_impl_10;
```

```
figure();
bar(errors);
title('The total error using trapz()')
ylabel('Total error from S0=0 to S0=Smax')
set(gca,'xticklabel',{'Explicit, N=300', 'Implicit, N=300', 'Implicit, N=100', ...
    'Implicit, N=50', 'Implicit, N=10'})

times(1) = time_expl;
times(2) = time_impl_300;
times(3) = time_impl_100;
times(4) = time_impl_50;
times(5) = time_impl_10;

figure();
bar(times, 'red');
title('Execution time')
ylabel('Time')
set(gca,'xticklabel',{'Explicit, N=300', 'Implicit, N=300', 'Implicit, N=100', ...
    'Implicit, N=50', 'Implicit, N=10'})

stability_test();


function stability_test()
    S=0; K=15; r=0.1; sigma=0.25; T=0.5; gamma=1;
    M=100;

    N = 100:10:1000;

    for i = 1:length(N)
        [stock_price, time, option_price, stable] = price_european_explicit(S, K, r, ...
            sigma, T, gamma, N(i), M);

        s(i) = stable;
    end

    figure()
    plot(N, s);
    title('Stability test for Explicit method (0 = unstable, 1 = stable) when M=100');
    xlabel('Number of time-steps')
    ylabel('Stable')
end
```

D Code for section 7 Computation of the Greek Δ

D.1 Main program

```
% Assignment 3 - Monte Carlo Methods vs. Finite Difference Methods

clear all;
close all;

% Parameter values
y_0 = 14;     % S(0)
```

```matlab
K = 15;
r = 0.1;
sigma = 0.25;
T = 0.5;
gamma = 1;
s_max = 4*K;


MC_steps = 10000;
M = 100;      % M is the number of price points (space steps)
N = 1000;     % N is the number of time steps

delta_s = s_max/M;
s = delta_s:delta_s:s_max;

%% Exact value of delta for European call (equal to N(d1))
d1 = ( log(s./K) + (r + 0.5.*sigma.^2).*T )./(sigma.*sqrt(T));
delta_exact = 0.5.*(1+erf(d1./sqrt(2)));
% figure;
% plot(s,delta_exact);


%% Computation of delta
% Monte Carlo method (Euler with antithetic variates)
tic;
[ V,delta_MC ] = Euler_antithetic( MC_steps,M,N,K,r,sigma,T,gamma);
disp(toc);

figure;
plot(s(1:end-1),delta_MC,'r',s(1:end-1),delta_exact(1:end-1),'b');
title('Computed delta with Monte Carlo Method');
xlabel('stock price');
ylabel('value of delta');
legend('Monte Carlo delta','exact delta');

error_MC = abs(delta_exact(1:end-2) - delta_MC(1:end-1));
figure;
plot(s(2:end-1),error_MC);
title('Error of delta with Monte Carlo method');
xlabel('stock price');
ylabel('error of delta');

% Finite difference method (Euler backward)
tic;
[ s_FD,price,delta_FD ] = Euler_Backward( M,N,K,r,sigma,T,gamma);
disp(toc);

figure;
plot(s(1:end-2),delta_FD,'r',s(1:end-2),delta_exact(1:end-2),'b');
title('Computed delta with Finite Difference Method');
xlabel('stock price');
ylabel('value of delta');
legend('FD delta','exact delta');

error_FD = abs(delta_exact(1:end-2) - delta_FD);
figure;
plot(s_FD(1:end-1),error_FD);
title('Error of delta with Finite Difference method');
```

```
xlabel('stock price');
ylabel('error of delta');
```

D.2 Function Euler antithetic (MC method)

```
function [ V,delta ] = Euler_antithetic( MC_steps,M,N,K,r,sigma,T,gamma)
% This function calculates the price of a European call option using
% Euler's method and the following stock dynamics:
%      dS(t) = r*S(t)*dt + sigma*S(t)^gamma*dW(t)

% MC_steps is the number of Monte-Carlo simulations
% N is the number of time steps

delta_t = T/N;
s_max = 4*K;
delta_s = s_max/M;


s_matrix = ones(MC_steps,M);
for i = 1:M
    s_matrix(:,i) = delta_s*i;
end

y1 = s_matrix;
y2 = y1;
for j = 1:N
    randns = randn(MC_steps,M);
    delta_W = randns * sqrt(delta_t);
    y1 = y1 + r*delta_t*y1 + sigma*y1.^gamma.*delta_W;
    y2 = y2 + r*delta_t*y2 - sigma*y2.^gamma.*delta_W;
end

% Compute V(T) = max(S(T) - K, 0)
V1 = max(y1-K,0);
V2 = max(y2-K,0);
V = exp(-r*T) * mean((V1+V2)/2);


% s = delta_s:delta_s:s_max;
% V(M) = 0;
% for k = 1:length(s)
%     y1 = s(k)*ones(MC_steps,1);
%     y2 = y1;
%
%     for j = 1:N
%         randns = randn(MC_steps,1);
%         delta_W = randns * sqrt(delta_t);
%         y1 = y1 + r*delta_t*y1 + sigma*y1.^gamma.*delta_W;
%         y2 = y2 + r*delta_t*y2 - sigma*y2.^gamma.*delta_W;
%     end
%
%     % Compute V(T) = max(S(T) - K, 0)
%     V1 = max(y1-K,0);
%     V2 = max(y2-K,0);
%     V(k) = exp(-r*T) * mean((V1+V2)/2);
% end
% Compute delta dV/ds with Euler centered difference?
delta(M-1) = 0;
for k = 2:M-1
    delta(k) = (V(k+1)-V(k-1))/(2*delta_s);
end

end
```

D.3 Function Euler Backward (FDM)

```
function [ s,price,delta ] = Euler_Backward( M,N,K,r,sigma,T,gamma)
% Euler backward method to price European call

v = zeros(M-1,N);
s_max = 4*K;
delta_s = s_max/M;
delta_t = T/N;

s = delta_s:delta_s:s_max;
t = delta_t:delta_t:T;
v(:,N) = max(s(1:end-1)-K,0); % Final condition (pay-off function)

for n = N:-1:2
    % Define matrix A
    A = zeros(M-1,M-1);
    A(1,1) = 1;
    for j = 2:(M-1)
        a(j) = - 0.5*sigma^2*s(j)^(2*gamma)*delta_t/(delta_s)^2 + ...
            0.5*r*s(j)*delta_t/delta_s;
        b(j) = 1 + sigma^2*s(j)^(2*gamma)*delta_t/(delta_s)^2 + delta_t*r;
        c(j) = - 0.5*r*s(j)*delta_t/delta_s - ...
            0.5*sigma^2*s(j)^(2*gamma)*delta_t/(delta_s)^2;
    end
    for j = 2:M-2
        A(j,j-1:j+1) = [a(j) b(j) c(j)];
    end
    A(M-1,M-2:M-1) = [a(M-1) b(M-1)];

    % Define matrix B
    B = zeros(M-1,1);
    B(M-1) = c(M-1) * (s_max - K*exp(-r*(T-t(n-1))));

    % Euler backward
    v(:,n-1) = A\(v(:,n) - B);
end

s = s(1:end-1);
price = v(:,1);

% surf(t,s,v);

% Compute delta dv/ds
delta(M-2) = 0;
for j = 2:M-2
```

```
    delta(j) = (v(j+1,1)-v(j-1,1))/(2*delta_s);
end

end
```