Assignment 1 - Monte Carlo SImulation to price European Options

Table of Contents

Written by Peili Guo (peili.guo.7645@student.uu.se) and Sijia Wang (Sijia.Wang.7090@student.uu.se) This report is for Computational Finance: Pricing and Valuation Assignment 1 Group 7.

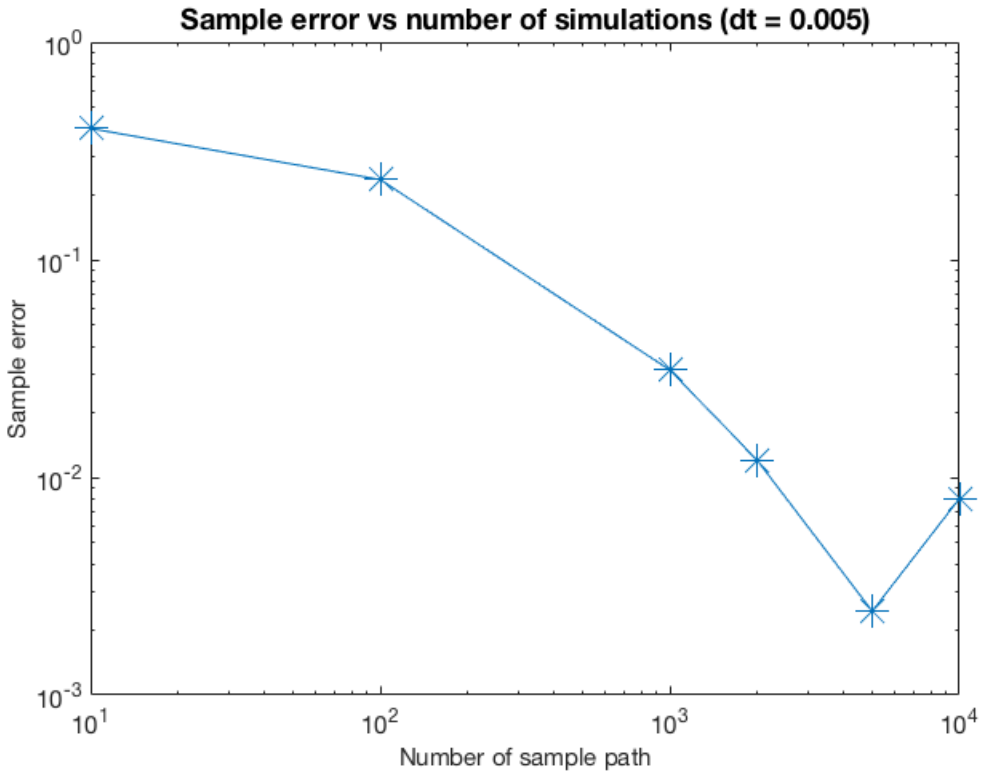
In this project we implemented Euler's method in Matlab to price a European call option. Different discretization size and number of Monte Carlo simulations were used to experiment the error when gamma = 1, where there is analytical solution for Black-Scholes Model. Antithetic Variates Method were used to reduce the sample error. Finally, a plot of V (option value) were plotted as a function of gamma.

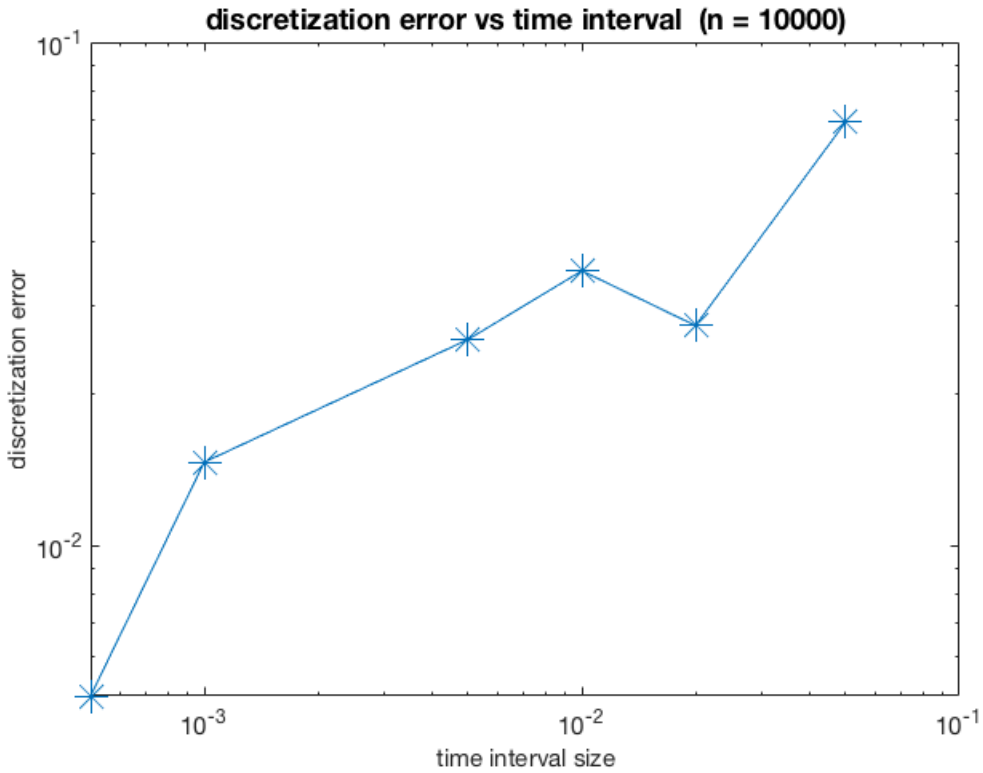
Part 1 - Experiments with error

```
clear all;
close all;
nrange = [10,100,1000,2000,5000,10000];%number of sample points for MC
ii = 0;
sigma = 0.25;
r = 0.1;
s0 = 14;
K = 15;
gamma = 1;
np =100;
T = 0.5;
rsol = bsexact(sigma,r,K,T,s0);

for nn = nrange
    ii = ii+1;

    [err_s(ii),V(ii)] = mc_euler(sigma, r, s0, K, gamma, np, T, nn,
 rsol);
end

%plot

f1 = figure('position', [0, 0, 700, 500]);
loglog(nrange,err_s,"*-",'MarkerSize',20)
set(gca, 'FontSize', 15)
xlabel('Number of sample path','FontSize',15);
ylabel('Sample error','FontSize',15);
title('Sample error vs number of simulations (dt =
 0.005)', 'FontSize', 18);
```

```
dtrange = [10,25,50,100,500,1000]; %number of discretization points
ii = 0;
for nn = dtrange
    ii = ii+1;

    [err_d(ii),V(ii)] = mc_euler(sigma, r, s0, K, gamma, nn, T, 10000,
 rsol);
end
```

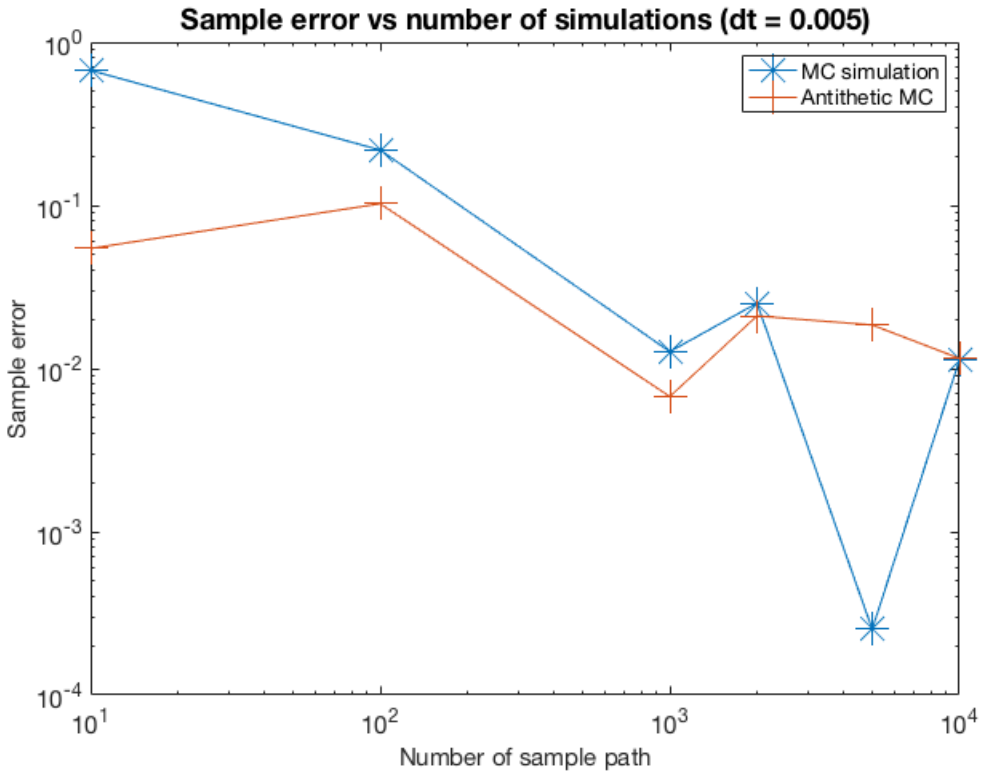
```
f2 = figure('position', [0, 0, 700, 500]);
loglog(T./dtrange, err_d,'*-','MarkerSize',20)
set(gca, 'FontSize', 15)
xlabel('time interval size','FontSize',15);
ylabel('discretization error','FontSize',15);
title('discretization error vs time interval  (n =
 10000)', 'FontSize', 18);

% in this part, we set the discretize size = 0.005 to test the sample
 error
% as a function of the number of the sample paths. and set the number
 of MC
% runs to 10000 to test the discretization error as a function of the
 time
% step.
% we can observe in sample error, the larger the sample paths, the
 smaller
% the error, until the discretization error is significant in the
 total
% error.
% In discretization error, when reduce the time interval from 0.05 to
% 0.001, the error decreases linearly.
% Euler's method converge until either sample error or discretization
 error
% dominates.
```


Sample error vs number of simulations (dt = 0.005)
Sample error
Number of sample path
10^{0}
10^{-1}
10^{-2}
10^{-3}
10^{1}
10^{2}
10^{3}
10^{4}


discretization error vs time interval (n = 10000)
discretization error
time interval size
10^{-1}
10^{-2}
10^{-3}

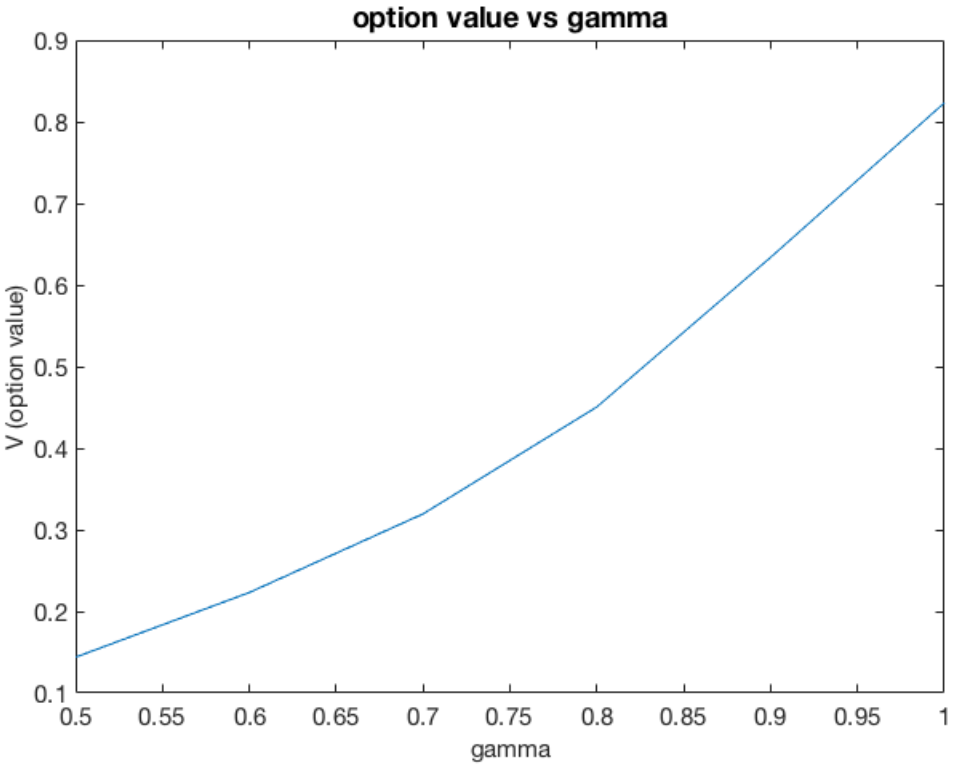
Part 2 - Antithetic variates to improve the results

```
ii = 0;
for nn = nrange
    ii = ii+1;

    [err_sa(ii),err_n(ii),V(ii)] = mcan_euler(sigma, r, s0, K, gamma,
 100, T, nn, rsol);

end


f3 = figure('position', [0, 0, 700, 500]);
loglog(nrange,err_n,"*-",'MarkerSize',20)
hold on
loglog(nrange,err_sa,"+-",'MarkerSize',20)
legend("MC simulation", "Antithetic MC")
set(gca, 'FontSize', 15)
xlabel('Number of sample path','FontSize',15);
ylabel('Sample error','FontSize',15);
title('Sample error vs number of simulations (dt =
 0.005)', 'FontSize', 18);

% use anithetic variate method, may improve the accuracy of MC method,
 but
% depends on the random number generator of the software. To get a
 more
% accurate of the SDE, a better discretization scheme could be used,
 e.g.
% Runge Kutta scheme, Milstein Scheme, that give a higher order of
% accuracy.
```



Part 3 - V of a function of gamma.

```
gammarange = 0.5:0.1:1;
ii = 0;
for g = gammarange
    ii = ii+1;

    [err_s(ii),V(ii)] = mc_euler(sigma, r, s0, K, g, 100, T, 10000,
 rsol);
end

f4 = figure('position', [0, 0, 700, 500]);
plot(gammarange, V)
set(gca, 'FontSize', 15)
xlabel('gamma','FontSize',15);
ylabel('V (option value)','FontSize',15);
title('option value vs gamma', 'FontSize', 18);

% The option value is positive correlated to gamma.
```



function called in script:

```
dbtype('mc_euler.m')
dbtype('mcan_euler.m')
```

```
1     function [err,V] = mc_euler(sigma, r, s0, K, gamma, np, T, n,
 rsol)
2     % dt = time interval size
3     % T = final time
4     % n = time of monte carlo simulation
5     % default we set to run 10 times of MC
6     % rsol = real analytical solution
7
8     S(1) = s0;
9     dt = T/np;
10
11    range = linspace(0,T,np);
12
13
14
15        for j = 1:n
16            S(j) = s0;
17
18
19
20            for i = 2:length(range)
```

```
21
22
23                S(j) = S(j) + r*S(j)*dt +
 sigma*(S(j)^gamma)*randn()*sqrt(dt);
24
25
26            end
27
28            V(j) = max(S(j)-K, 0);
29
30        end
31
32        EqV = mean(V);
33
34        V0 = exp(-r*T) * EqV;
35
36
37        err = abs(rsol - V0);
38        V=V0;
39
40    end
```

```
1
2     function [err2,errn,V] = mcan_euler(sigma, r, s0, K, gamma, np,
 T, n, rsol)
3     % dt = time interval size
4     % T = final time
5     % n = time of monte carlo simulation
6     % default we set to run 10 times of MC
7     % rsol = real analytical solution
8
9     S(1) = s0;
10
11
12    dt = T/np;
13    range = linspace(0,T,np);
14
15
16
17        for j = 1:n
18            S(j) = s0;
19            S2(j) = s0;
20
21
22
23            for i = 2:length(range)
24
25                my_ran = randn();
26
27
28                S(j) = S(j) + r*S(j)*dt +
 sigma*(S(j)^gamma)*my_ran*sqrt(dt);
29                S2(j) = S2(j) + r*S2(j)*dt -
 sigma*(S2(j)^gamma)*my_ran*sqrt(dt);
```

```
30
31
32
33              end
34
35              V(j) = max(S(j)-K, 0);
36              V2(j) = max(S2(j)-K, 0);
37
38
39
40          end
41
42          EqV = mean(V);
43          EqV2 = mean(V2);
44
45
46          V0 = exp(-r*T) * EqV;
47          V02 = exp(-r*T) * EqV2;
48
49          V = 0.5*(V0+V02);
50
51          errn = abs(rsol - V0);
52          err2 = abs(rsol - V);
53
54
55      end
```

Published with MATLAB® R2018a