Assignment 2 - Finite difference methods 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 2 Group 11.

In this project we implemented both explicit and implicit Euler's method in Matlab to price an European call option under CEV-model. We studied the convergency, stability, computational cost and how the solution varies with \gamma.

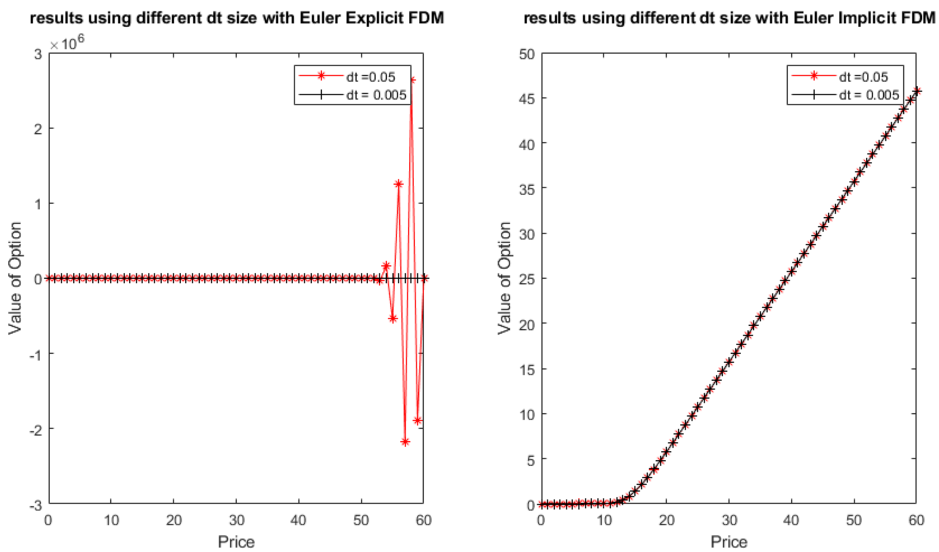
1. Stability and accuracy of Explicit and Implicit Euler's method

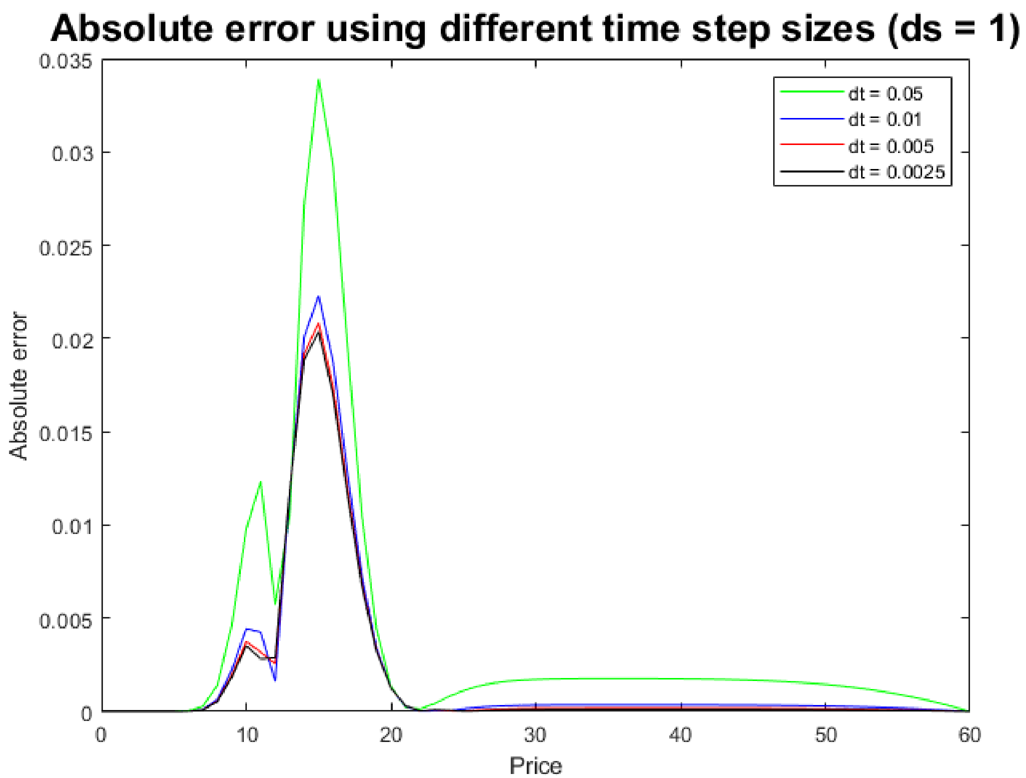
```
clear all;
close all;

K = 15;
r = 0.1;
sigma = 0.25;
T = 0.5;
gamma = 1;
Smax = 4 * K;
tn = 6; %num of points in the time grid
sn = 61; % number of points in the space (price grid)
trange = [11 51 101 151 201];
srange = [61 121 181 241 301];

% set ds = 1, and change different to different dt
% explicit Euler's method
[dt1,ds1,se1,ve1] = fdexplicit(K,r,sigma,T,trange(1),sn);
[dt2,ds2,se2,ve2] = fdexplicit(K,r,sigma,T,trange(2),sn);
[dt3,ds3,se3,ve3] = fdexplicit(K,r,sigma,T,trange(3),sn);
tic %time it to see computational cost
disp('calling explicit function with dt = 0.0033, ds = 1')
[dt4,ds4,se4,ve4] = fdexplicit(K,r,sigma,T,trange(4),sn);
toc

% implicit Euler's method
[dt1,ds1,si1,vi1] = fdimplicit(K,r,sigma,T,gamma,trange(1),sn);
[dt2,ds2,si2,vi2] = fdimplicit(K,r,sigma,T,gamma,trange(2),sn);
[dt3,ds3,si3,vi3] = fdimplicit(K,r,sigma,T,gamma,trange(3),sn);
```

```
tic
disp('calling implicit function with dt = 0.0033, ds = 1')
[dt4,ds4,si4,vi4] = fdimplicit(K,r,sigma,T,gamma,trange(4),sn);
toc
% analytical solution when gamma = 1
for i=1:sn
    exact1(i) = bsexact(sigma, r, K, T, si1(i));
    exact2(i) = bsexact(sigma, r, K, T, si2(i));
    exact3(i) = bsexact(sigma, r, K, T, si3(i));
    exact4(i) = bsexact(sigma, r, K, T, si4(i));
end
% compute the largest error
err_e1 = abs(ve1 - exact1);
err_i1 = abs(vi1 - exact1);
err_e2 = abs(ve2 - exact2);
err_i2 = abs(vi2 - exact2);
err_e3 = abs(ve3 - exact3);
err_i3 = abs(vi3 - exact3);
err_e4 = abs(ve4 - exact4);
err_i4 = abs(vi4 - exact4);


% plot 1 showing the stability of the results using explicit and
 implicit
% method
f1 = figure('position', [0,0,1000,500]);
subplot(1,2,1);
plot(se1,ve1,'r*-')
hold on
plot(se3,ve3,'k+-')
legend("dt =0.05", "dt = 0.005")
xlabel('Price');
ylabel('Value of Option');
title({'results using different dt size with Euler Explicit FDM';'
 '});

subplot(1,2,2);
plot(si1,vi1,'r*-')
hold on
plot(si3,vi3,'k+-')
legend("dt =0.05", "dt = 0.005")
xlabel('Price');
ylabel('Value of Option');
title({'results using different dt size with Euler Implicit FDM','
 '});


f2 = figure('position', [0, 0, 700, 500]);
plot(si1,err_i1,'g')
hold on
plot(si2,err_i2,'b')
plot(si3,err_i3,'r')
plot(si4,err_i4,'k')
legend("dt = 0.05", "dt = 0.01", 'dt = 0.005', 'dt = 0.0025')
```

```
xlabel('Price');
ylabel('Absolute error');
title('Absolute error using different time step sizes (ds =
 1)', 'FontSize', 18);

% We can observe that the the explicit method is unstable if dt/ds is
% large. in order to have solution that converges, we need to control
% dt/ds. for example, at dt/ds >= 0.01 the solution is unstable.
% When we decreased dt/ds to below 0.005, the solution is stable.
 However
% the implicit method is always stable and we can see the absolute
 error
% did not explode for the different dt we experimented. The
 computational
% cost is more expensive for the implicit method compare to the
 explicit
% method (with the same dt and ds), because we need to compute the
% inverse of a tridiagonal matrix.
% But the good part of choose the implicit method is that we are not
% restricted to a small dt when using a small ds, which might cause a
% problem if the T for option is large, e.g. a long term option.
```

```
calling explicit function with dt = 0.0033, ds = 1
Elapsed time is 0.001884 seconds.
calling implicit function with dt = 0.0033, ds = 1
Elapsed time is 0.003652 seconds.
```



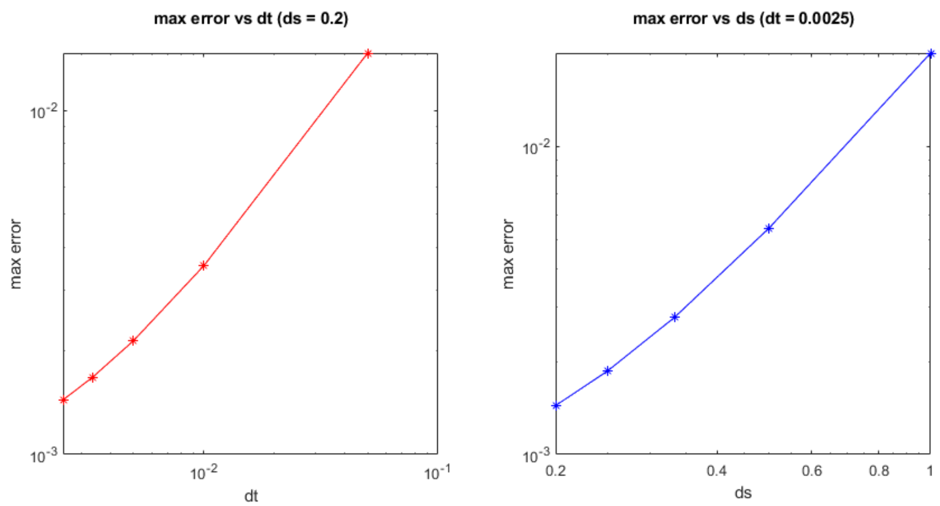


2. Accuracy using implicit method

Below, we will experiment the accuracy use only the Euler's implicit method, that the dt is not restricted to ds for the solution to converge.

```
count = 0;
for tt = trange
    exactt=[];
    count = count+1;
    [dtt,dst,sit,vi] = fdimplicit(K,r,sigma,T,gamma,tt,301);
    for i=1:length(sit)
        exactt(i) = bsexact(sigma, r, K, T, sit(i));
    end
    dtplot(count) = dtt;
    errtt(count) = max(abs(exactt - vi));
end
count = 0;
for ss = srange
    exactt=[];
    count = count+1;
    [dtt,dst,sit,vi] = fdimplicit(K,r,sigma,T,gamma,201,ss);
    for i=1:length(sit)
        exactt(i) = bsexact(sigma, r, K, T, sit(i));
    end
    dsplot(count) = dst;
    errss(count) = max(abs(exactt - vi));
end
```

```
f4 = figure('position', [0,0,1000,450]);
subplot(1,2,1);
loglog(dtplot,errtt,'r*-')
xlabel('dt');
ylabel('max error');
title({'max error vs dt (ds = 0.2)';' '});

subplot(1,2,2);
loglog(dsplot,errss,'b*-')
xlabel('ds');
ylabel('max error');
title({'max error vs ds (dt = 0.0025)';' '});

% We set the ds to 0.2 and experiment the error with different dt. the
% max error decrease linearly. The same can be observed when we set dt
 to
% 0.0025 and experiment with different ds.
```

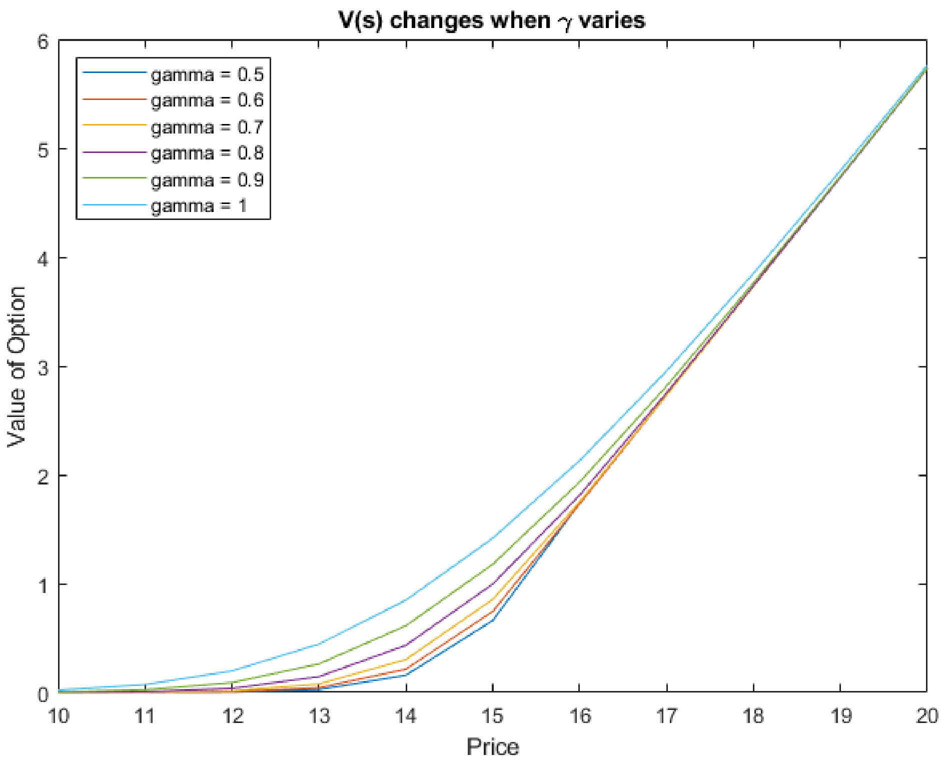


3. Experiment with different gamma

we set the gamma to between 0.5 and 1 to see how the value of option will behave.

```
count = 0;
for g = 0.5:0.1:1
    count = count + 1;
    [dt,ds,sg,vjg(count,:)] = fdimplicit(K,r,sigma,T,g,101,61);
end
f3 = figure('position', [0, 0, 700, 500]);
plot(sg,vjg)
xlim([10 20])
legend("gamma = 0.5", "gamma = 0.6", 'gamma = 0.7', 'gamma =
 0.8','gamma = 0.9','gamma = 1','Location','northwest')
xlabel('Price');
ylabel('Value of Option');
title('V(s) changes when \gamma varies');
```

```
% Here, we only plot the value of option when the strike price is between
% 10 and 20. we see that with larger \gamma, the value of option is higher.
% This is due to the fact that a larger \gamma means the stochastic part of
% the CEV model will be larger, which correcponding to a bigger volatility.
% Thus a higher value of option.
```



4. Finite difference solver called in script

```
dbtype('fdexplicit.m')
dbtype('fdimplicit.m')
```

```
function [dt,ds,s,last_v] = fdexplicit(K,r,sigma,T,tn,sn)

%% function to compute the call option price use emplicit euler
% parameter
% K = 15; strike price
% r = 0.1; risk free interest rate
% sigma = 0.25; volatility
% T = 0.5; option time
% tn: number of time points on discretization, and dt = (T - T0)/(tn-1)
```

```
    % sn: number of price points on discretizatin, and ds = (S - S0)/(sn-1)

    Smax = 4 * K; %the price range to compute
    t = linspace(0,T,tn); % time vector
    s = linspace(0,Smax,sn); % price vector
    dt = t(2)-t(1);
    ds = s(2)-s(1);
    last_v = max(s-K,0); % boundary condition, at time = T, we know the value of option
    last_v = last_v(:); %force a column vector

    for n=tn:-1:2
        v(1) = 0;
        v(sn) = Smax-K*exp(-r*(T-t(n-1)));
        for j=2:sn-1
            v(j)=last_v(j)+r*s(j)*dt/(2*ds)*(last_v(j+1)-last_v(j-1))...
                 +sigma^2*0.5*(s(j))^2*(dt/(ds^2))*(last_v(j+1)-...
                 2*last_v(j)+last_v(j-1))-dt*r*last_v(j);
        end
        last_v=v;
    end
    end
```

```
    function [dt,ds,s,vj] = fdimplicit(K,r,sigma,T,g,tn,sn)

    %% function to compute the call option price use implicit euler
    % parameter
    % K = 15; strike price
    % r = 0.1; risk free interest rate
    % sigma = 0.25; volatility
    % T = 0.5; option time
    % g = gamma, and set gamma = 1 to inspect ;
    % tn: number of time points on discretization, and dt = (T - T0)/(tn-1)
    % sn: number of price points on discretizatin, and ds = (S - S0)/(sn-1)

    Smax = 4 * K; %the price range to compute
    t = linspace(0,T,tn); % time vector
    s = linspace(0,Smax,sn); % price vector
    dt = t(2)-t(1);
    ds = s(2)-s(1);
    vj = max(s-K,0); % boundary condition, at time = T, we know the value of option
    vj = vj(:); %force a column vector
    sig2 = sigma*sigma;
    %construct the tridiagonal matrix
    a = zeros(1,sn);
    b = zeros(1,sn);
    c = zeros(1,sn);
    for i = 1:1:sn %iterate at each grid point
        a2(i) = 0.5*dt*(r*i - sig2*i^(2*g)*ds^(2*g-2));
```

```
        b2(i) = 1 + dt*(sig2*i^(2*g)*ds^(2*g-2) + r);
        c2(i) = -0.5*dt*(r*i + sig2*i^(2*g)*ds^(2*g-2));
        a(i) = 0.5*dt*(r*s(i)/ds - sig2*(s(i)^(2*g))/(ds*ds));
        b(i) = 1 + dt*(sig2*(s(i)^(2*g)/(ds*ds)) + r);
        c(i) = -0.5*dt*(r*s(i)/ds + sig2*(s(i)^(2*g)/(ds*ds)));
    end
    D = diag(a(3:end-1),-1) + diag(b(2:end-1)) + diag(c(2:end-2),1);

    for n = tn:-1:2
        v = zeros(sn,1);
        %v(1) = 0; already initialized
        v(sn,1) = Smax-K*exp(-r*(T-t(n-1)));
        f = vj(2:end-1); % take vj(2:end-1) exclude the boundary points
        f(end) = f(end) - c(end-1)*v(end); % for the v(t(i-1), s(n-1))
        v(2:end-1) = D\f;
        vj = v;
    end
    vj = vj';
end
```

Published with MATLAB® R2018a