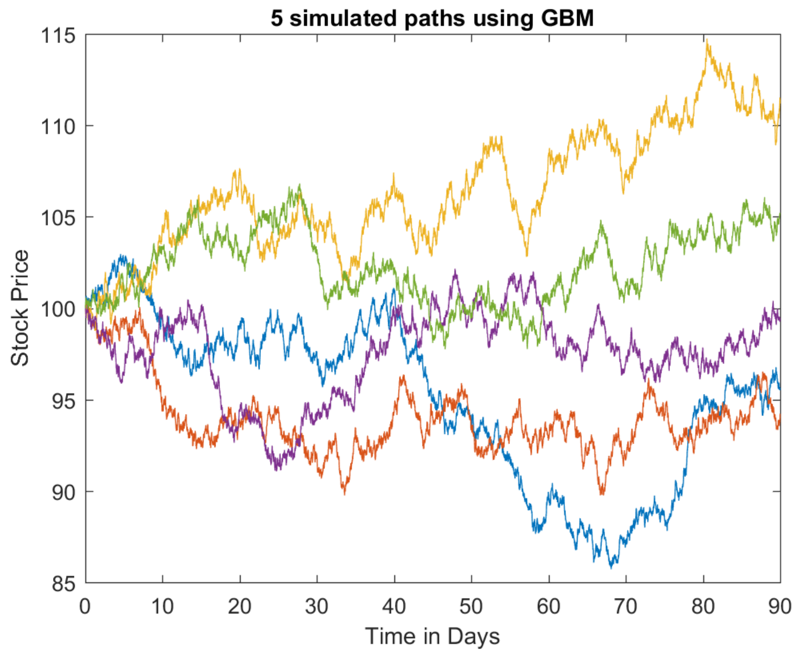
MGMT 237D:
Derivative Markets

Homework 2

Yi-Chi Chan Zhaofang Shi
Ahswin Kumar Ashok Kumar George Bonebright

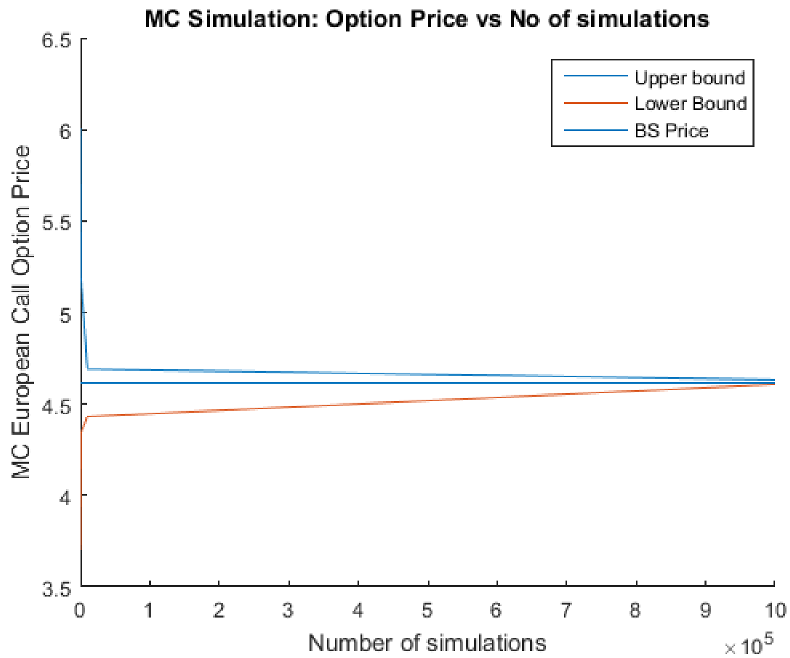
March 8, 2016

1.a. 5 Path Plot:



1.b. BS Call Price = 4.615.

1.c. Confidence Interval, Monte Carlo Simulation:

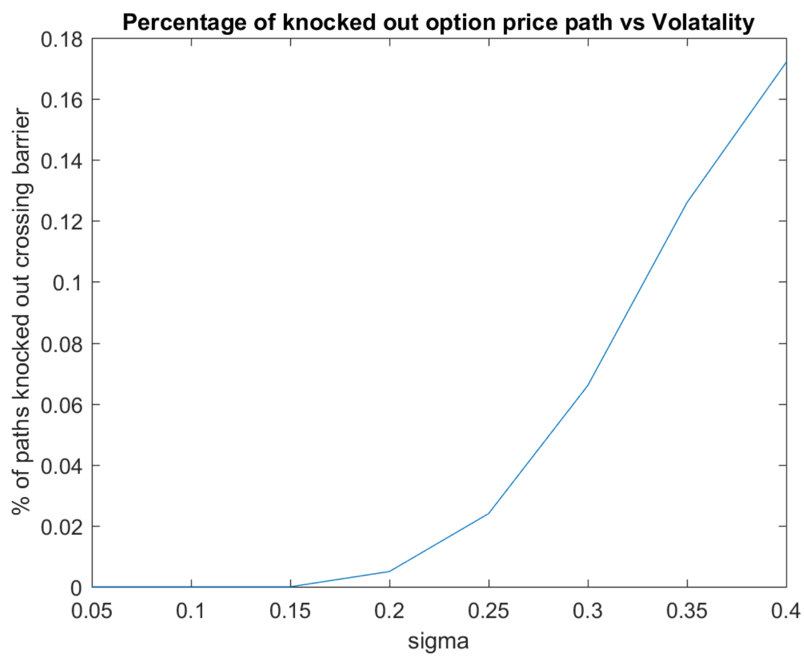


2.a. Number of simulations that crossed the barrier = 5.
Down and Put Option Estimate Price = 1.585728 with 95 perc conf (1.378485,1.792971)

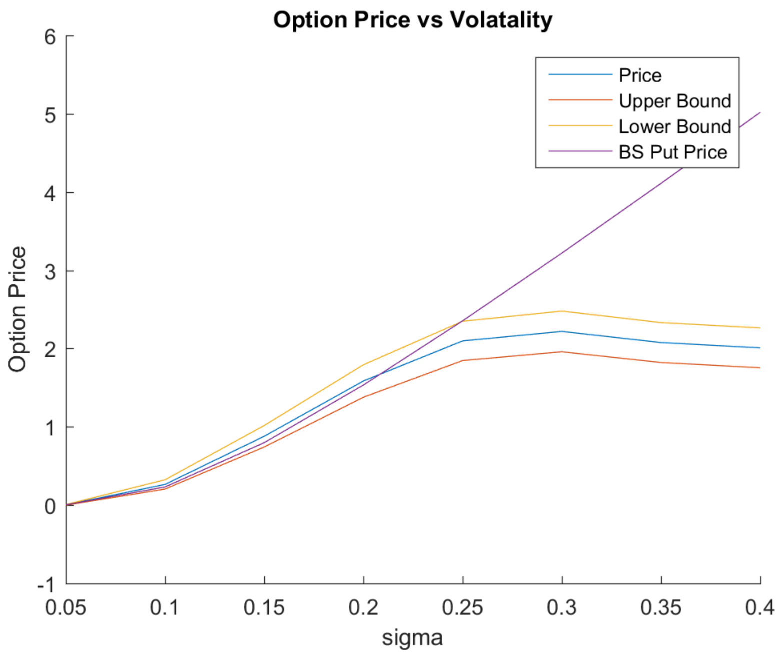
We can see from the above graph the percentage increases as sigma grows.This is due to the fact that as sigma becomes larger the stock is highly volatile. This creates more no of paths under MC simulation that crosses the barrier.

In general we expect the down and put option price to be less when compared to the Black-Scholes put price.The reason being the option is knocked out for certain number of paths (5 out of 1000). In our simulation we are getting the down and put to be higher. The reason for this is the number of simulations we are using is low. The down and put option price will be lesser than the BS Option price when we increase the simulation. The argument for this is similar to the argument that Binomial model option price tends to Black Scholes (continuous) model when we increase the number of nodes (can be visualized as number of simulations for Monte-Carlo).

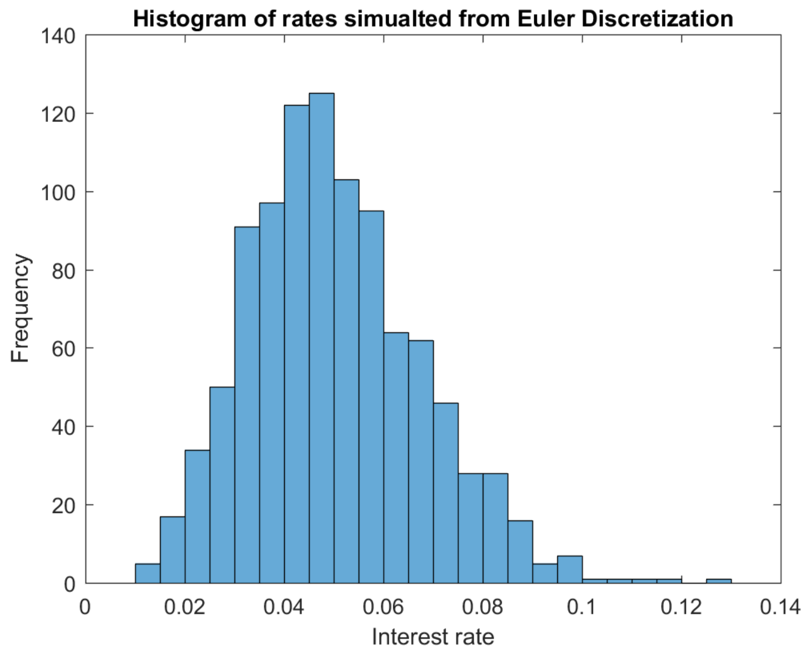
2.b. We can see from the above graph the percentage increases as sigma grows. This is due to the fact that as sigma becomes larger the stock is highly volatile. This creates more no of paths under MC simulation that crosses the barrier.



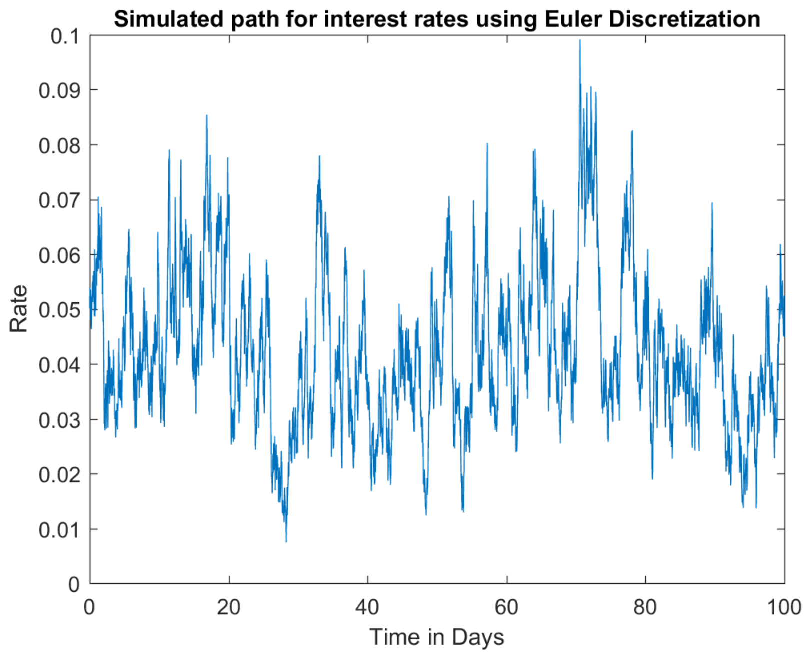
2.c. We can see that BS- option price increases as volatility increases. This is consistent with our expectation. For the down and put option there are two opposing effects due to Vol. As Vol increases the probability for option to be in money increases and at the same time the the probability to be knocked out (crossing the barrier) also increases. Thus we see the graph to change trend after sigma crosses 0.3.



3.a. Histogram:



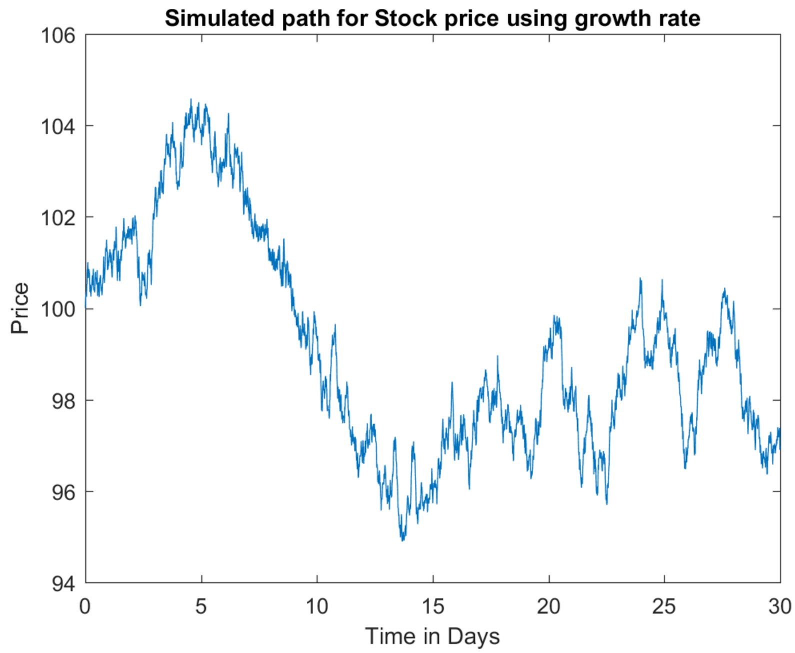
3.b. Simulated Trajectory:



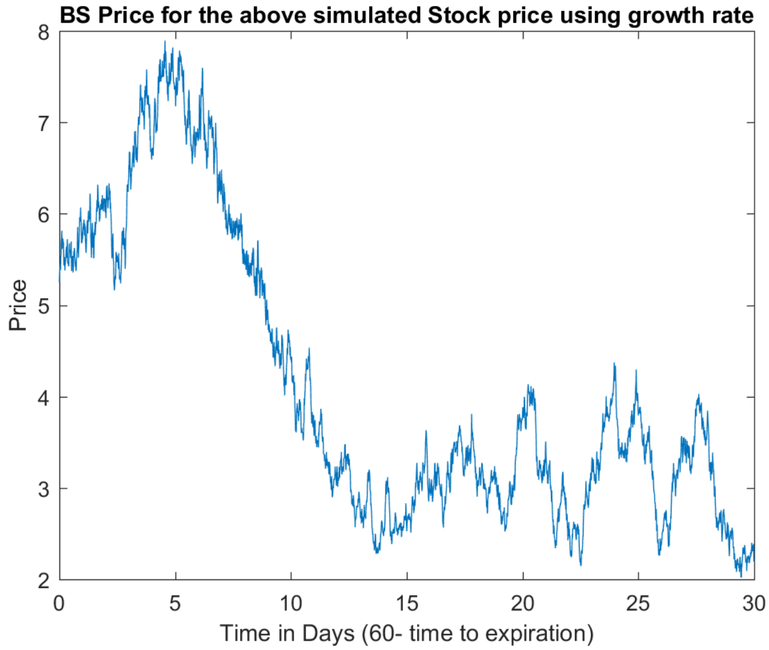
3.c. European Call Option Estimate Price = 0.687306 with 95 perc conf (0.668058,0.706553).

3.d. Option Estimate Price = 1.281788 with 95 perc conf (1.262933,1.300643).

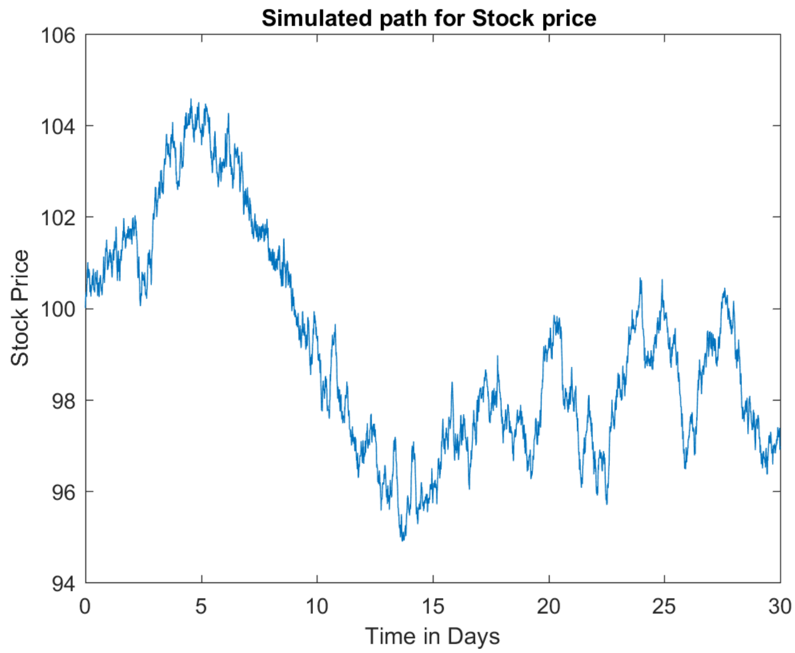
4.a. Simulated Stock Path:



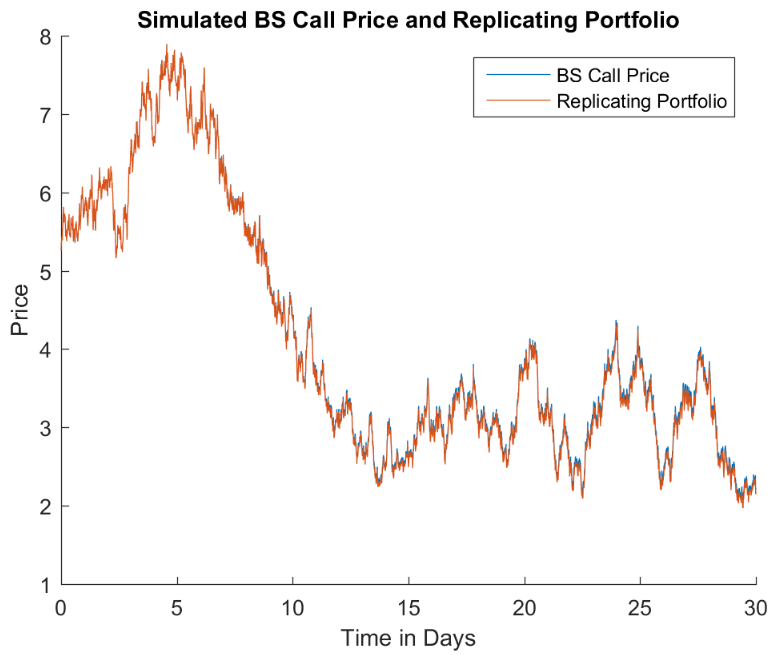
4.b. Black Scholes Price:



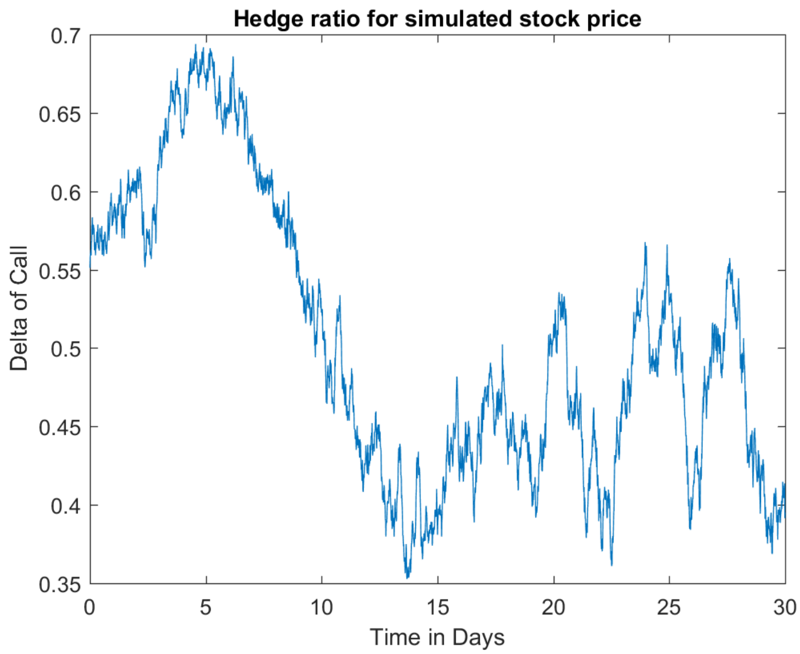
4.c. Simulated Stock Path:



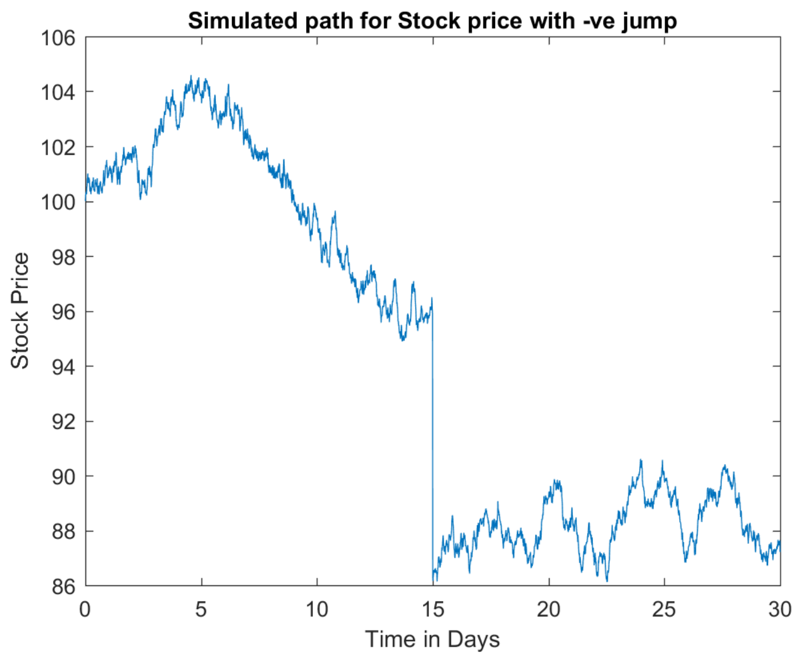
BS/Replicating Portfolio:

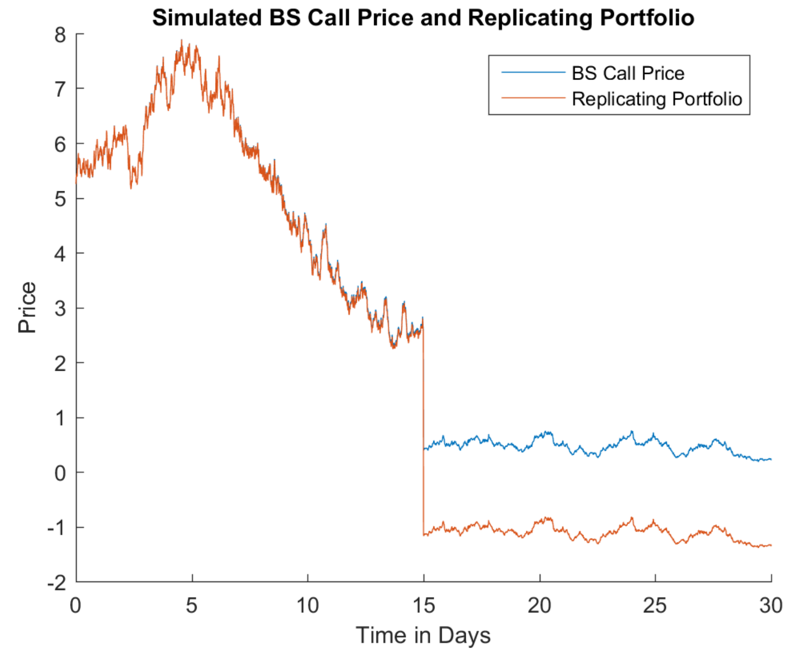


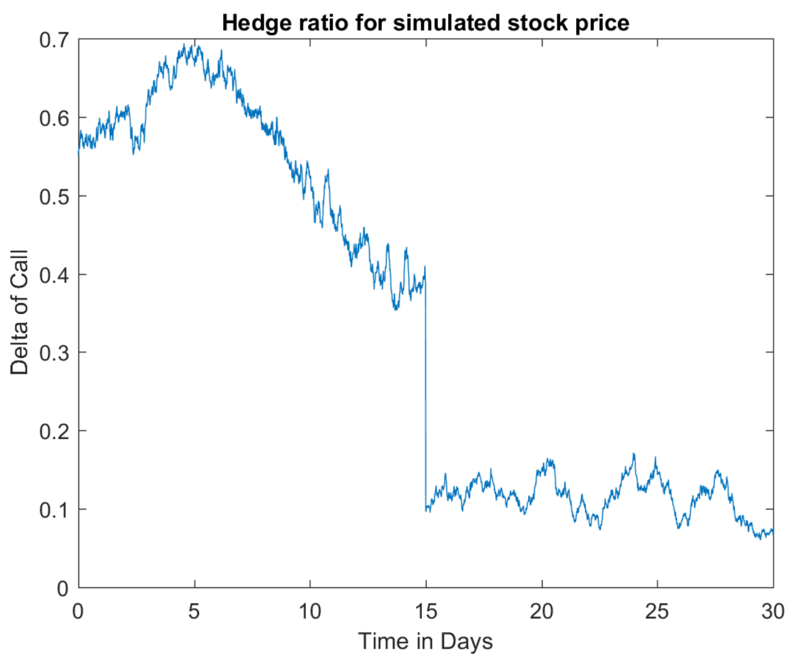
Hedging Ratio:



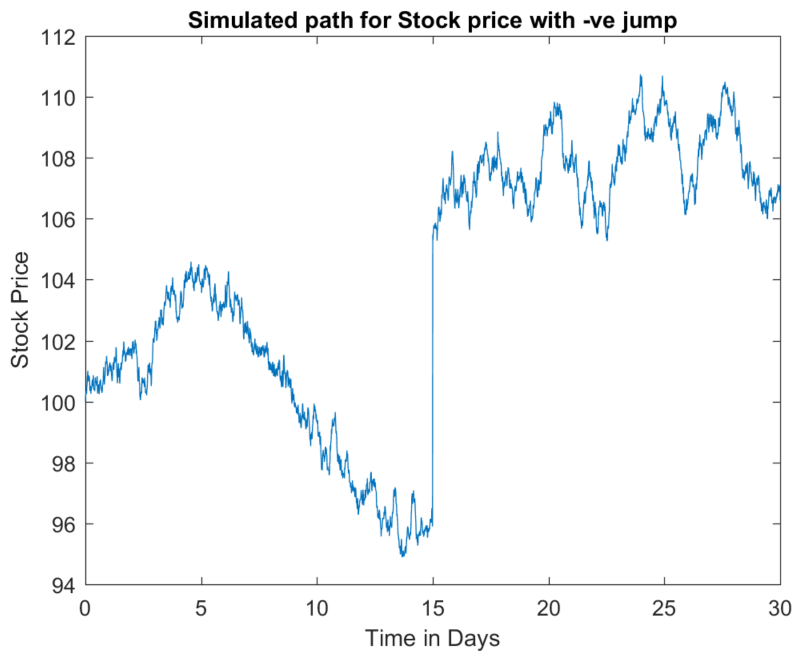
4.d. The stock price with the jump is as shown in the graph. We can see the sudden jump down in stock price causes the hedging to be ineffective at T=15 days(jump time). The replicating portfolio falls more than the call price for that moment (hedging is not continuous). From the next time(immediately after the jump is absorbed) the replicating portfolio closely follows the call price (with diff = diff in drop between call and replicating portfolio). If we try to synthesize the call by using our replicating portfolio, we will suffer loss = $\%.3f', BSCallPricedown(N/2+1) - V_T down(N/2+1))$.

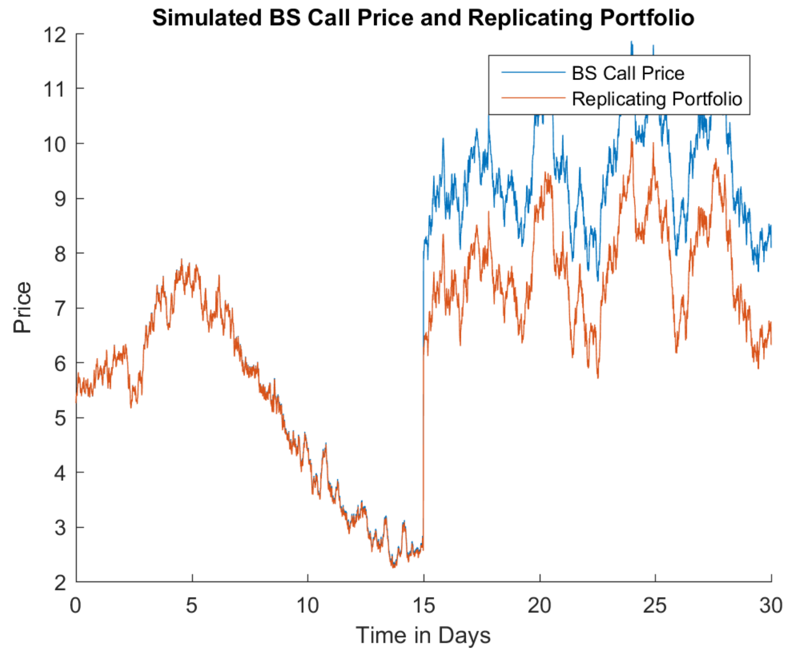


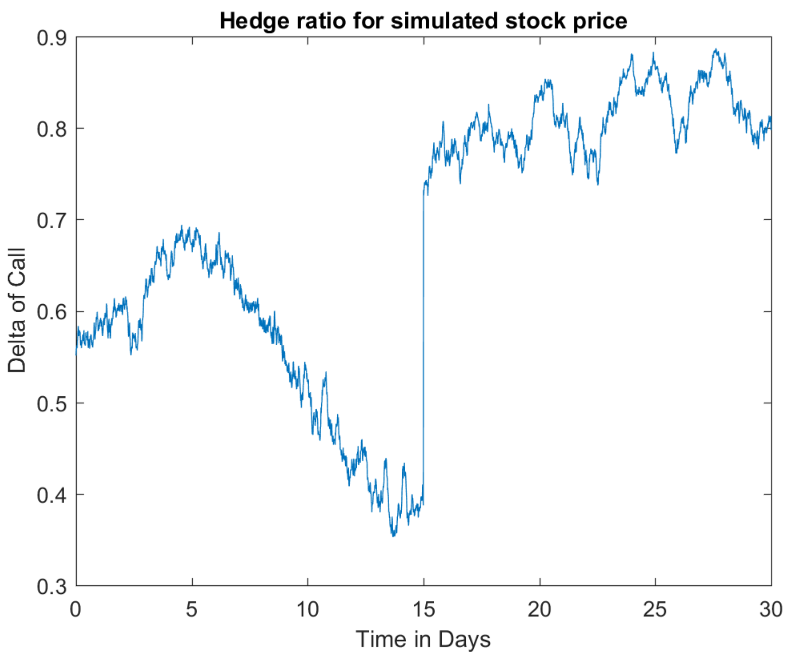

Simulated BS Call Price and Replicating Portfolio
BS Call Price
Replicating Portfolio
Price
Time in Days
8
7
6
5
4
3
2
1
0
-1
-2
0
5
10
15
20
25
30


Hedge ratio for simulated stock price
Delta of Call
Time in Days
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
0
5
10
15
20
25
30

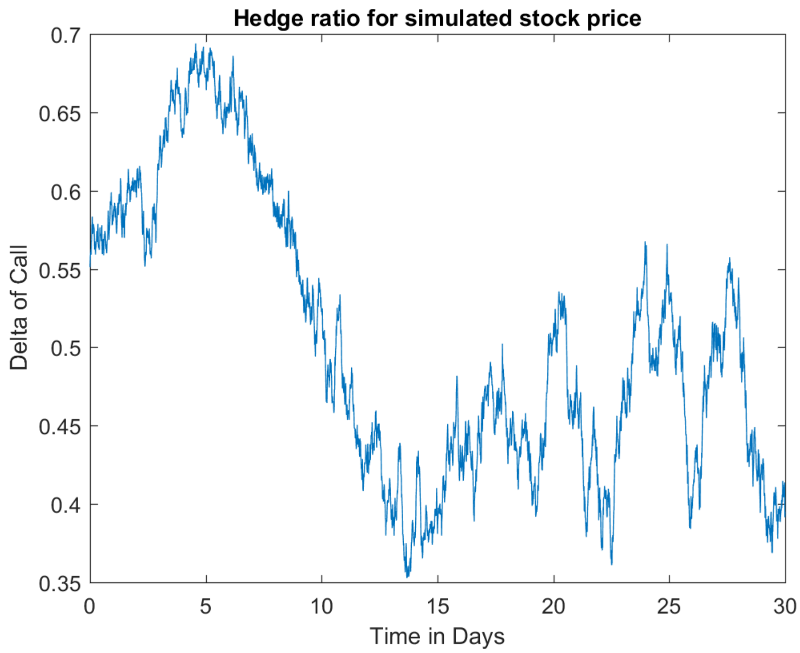
4.e. We can see similar observation. Even though stock jumps up, due to limitation of non continuous time hedging, we will not able to track the call price during the jump. The difference in Call Price is absorbed slowly and the replicating portfolio falls back again.

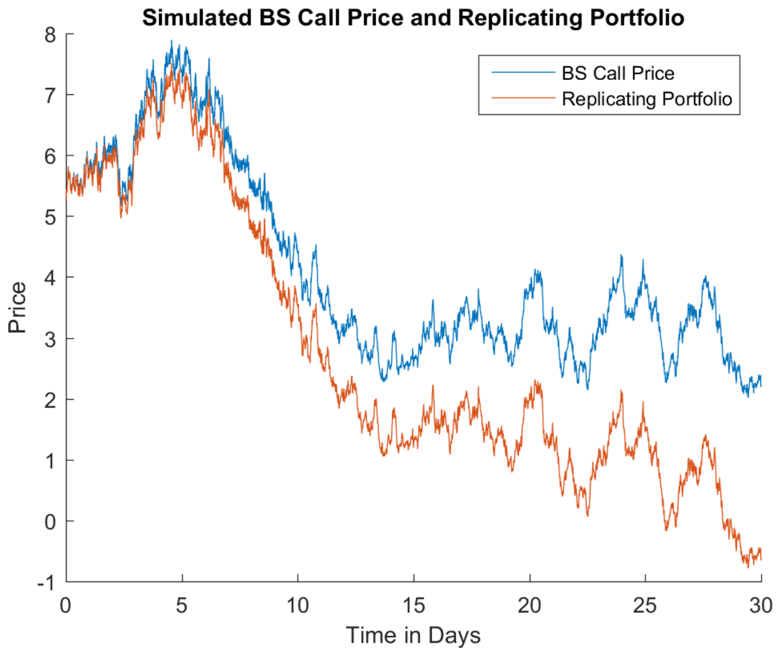



Simulated BS Call Price and Replicating Portfolio
BS Call Price
Replicating Portfolio
Price
Time in Days

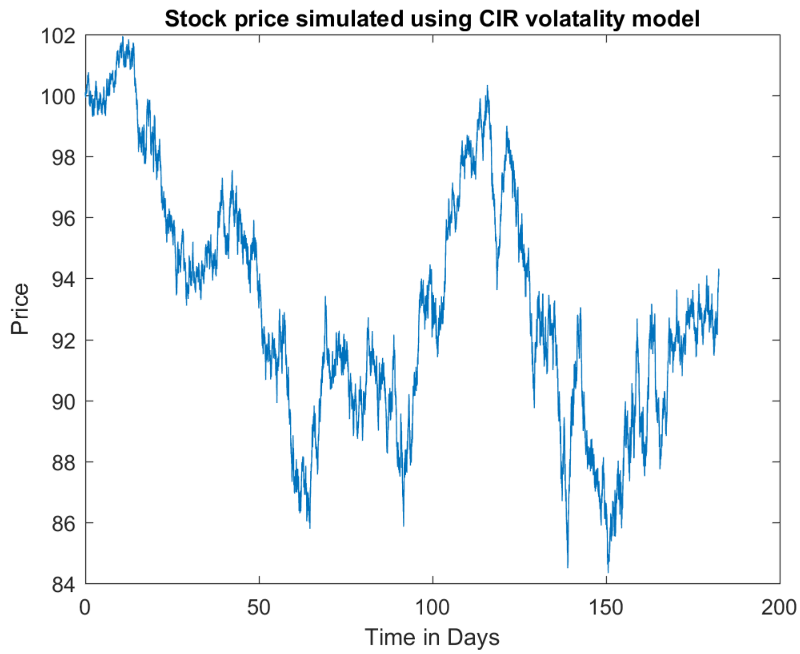

Hedge ratio for simulated stock price
Delta of Call
Time in Days

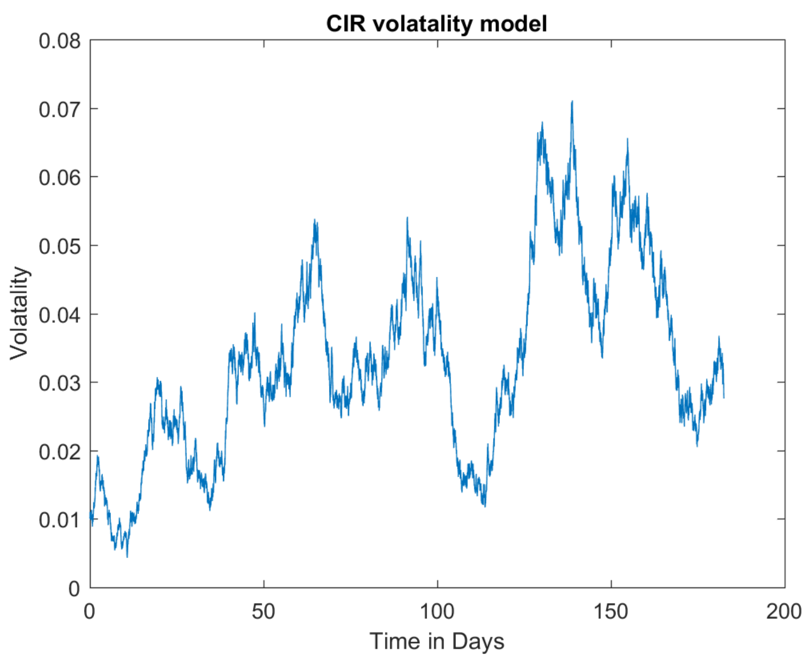
4.e. With the 20 basis points transaction cost,the replicating portfolio value diverges from the call value. This explanation falls with the fact that as time increases due to the volatile stock price movements we end up adjusting our portfolio more number of times.



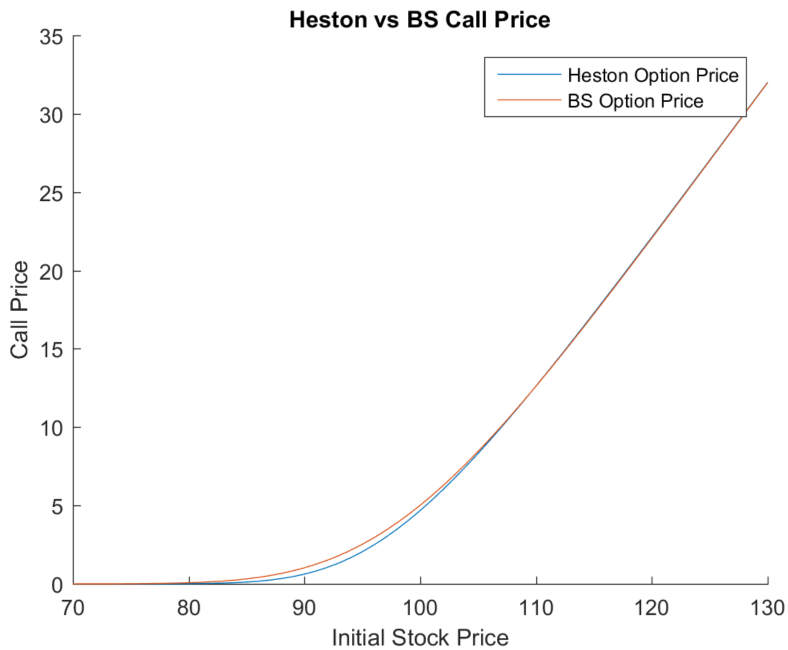

Simulated BS Call Price and Replicating Portfolio
BS Call Price
Replicating Portfolio
Price
Time in Days
8
7
6
5
4
3
2
1
0
-1
0
5
10
15
20
25
30

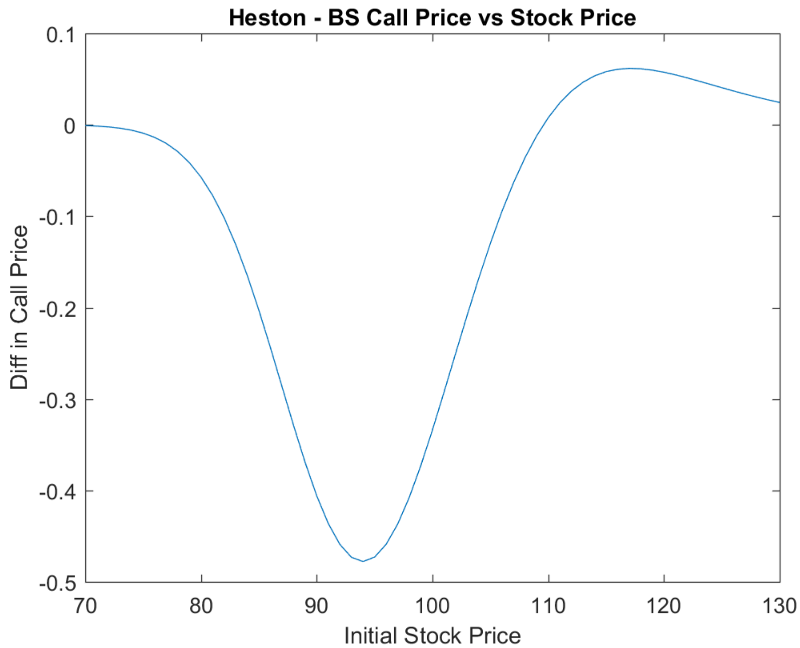
5.a. BS Call Option Price = 5.016981; Heston Call Option Price =4.683304.



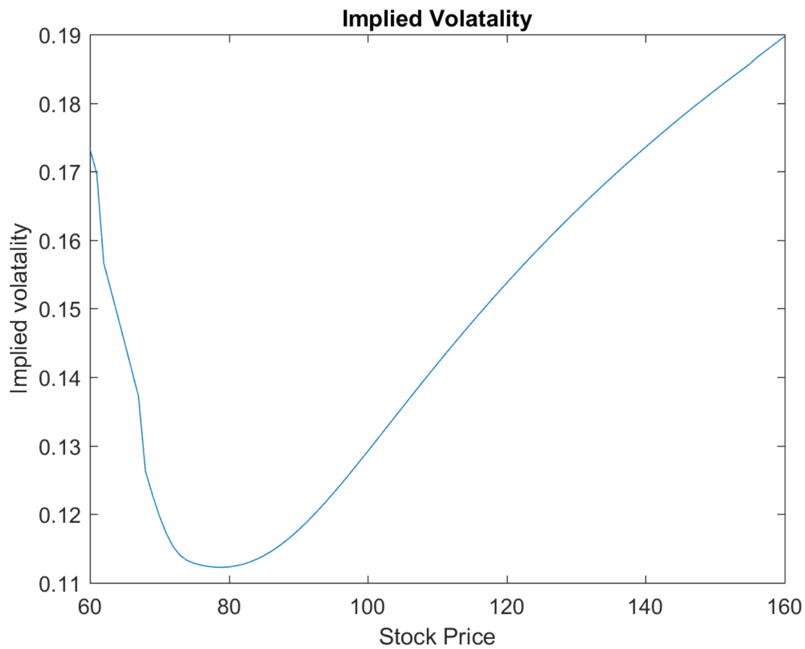


5.b. We can see from the graph, the Heston model for option price converges to BS option pricing model, for in the money call (S0 ¿ K). The reason for this is Heston model incorporates time varying volatility with negative rho. This makes out of money call options (low stock price) to have lesser probablity to be in the money when compared to in the money (for -ve rho). CIR model for vol being mean reverting.

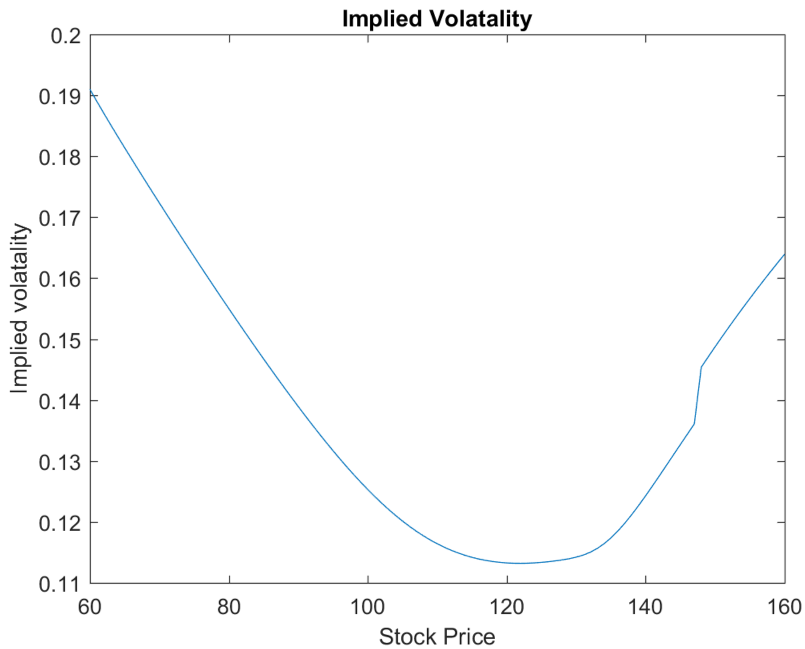



Heston - BS Call Price vs Stock Price
0.1
0
-0.1
-0.2
-0.3
-0.4
-0.5
Diff in Call Price
70
80
90
100
110
120
130
Initial Stock Price

5.c. Implied Volatility:



5.d. Implied Volatility, Rho=.5:



The volatility smile is right skewed for negative rho and left skewed for positive rho. In the plot, we used stock price from 60 to 160 to get the detail of the full vol plot. When rho ¡0, then the Heston model minimum occurs at Stock price close to K (from below), When rho ¿0, then the Heston model minimum occurs at Stock price close to K (from above).

```
%%Homework 2 Derivatives
%%by YiChi Chan, Sally , George , Ashwin Kumar
%%
%%Q1. Black-Scholes Closed Form Solution vs Monte-Carlo Simulation
%%a) Simulate 5 paths for Stock Price
clear;
S0 = 100;
Sb= -1;
N_Days = 360;
T  = 1/4;
K  = 100;
r  = 0.05;
sigma = 0.2;
div_yield = 0;
n_sims = 5;
trading_hrs = 8;
N = 12*trading_hrs*N_Days*T;
dt = (5/60)/trading_hrs/N_Days;
[St,]= GetSimulatedGBMStockPrice(S0,Sb,r,sigma,N,dt,n_sims,'N');
figure;
x = linspace(0,T*N_Days,N+1);

plot(x,St(:,:));
title('5 simulated paths using GBM');
xlabel('Time in Days') % x-axis label
ylabel('Stock Price') % y-axis label
%%b Calculate Black Scholes Call Option Price
[BS_Call_Price] = bs_call_price(S0,K,r,T,sigma,div_yield);
fprintf('BS Call Price = %0.3f \n',BS_Call_Price);

%%c Monte Carlo Simulation for Option Price
n_sims = [100,1000,1000000,1000000000];
P = zeros;
L = zeros;
U = zeros;
for i=1:length(n_sims)
    [P(i),L(i),U(i)]=MC_Option_Price_Direct(S0,K,r,sigma,T,'EC',
    n_sims(i));
end
figure; hold on;
p1 =plot(n_sims,U);
p2 =plot(n_sims,L);
p3 = refline(0,BS_Call_Price);
title('MC Simulation: Option Price vs No of simulations');
xlabel('Number of simulations') % x-axis label
ylabel('MC European Call Option Price') % y-axis label
legend([p1,p2,p3],'Upper bound','Lower Bound','BS Price');
fprintf('We can see from the above graph the length of the
    confidence interval\n decreases as the number of simulations
    increases. This satisfies with our understanding\n of standard
    error which is inversely proportional to number of simulations.
     Also the MC price\n tends towards the BlackScholes price as N
    increases\n');
%%2 Down and Out Put Option Price by MC Simulation

clear;
S0= 100;
```

```
T = 1/4;
K = 95;
N_Days = 360;
Sb = 75;
r = 0.05;
sigma = 0.2;
delta = 0;
n_sims  = 1000;
trading_hrs = 8;
N = 12*trading_hrs*N_Days*T;
dt = (5/60)/trading_hrs/N_Days;
[St, Within_Barrier,]= GetSimulatedGBMStockPrice(S0,Sb,r,sigma,N,dt
    ,n_sims,'D-O-P');

fprintf('Number of simulations that crossed the barrier = %d\n',
    n_sims-sum(Within_Barrier));
r_array = r*ones(n_sims,1);
[P1,L1,U1]=MC_Option_Price(St(:,N+1),K,r_array(:,1), T,'EP',
    Within_Barrier,n_sims);
fprintf('Down and Put Option Estimate Price = %f with 95 perc conf
    (%f,%f)\n',P1,L1,U1);


[BS_Put_Price] = bs_put_price(S0,K,r,T,sigma,delta);
fprintf('In general we expect the down and put option price to be
    less when\n compared to the Black-Scholes put price.The reason
    being the option is knocked\n out for certain number of paths
    (5 out of 1000). In our simulation we are \n getting the down
    and put to be higher. The reason for this is the number of\n
    simulations we are using is low. The down and put option price
    will be lesser\n than the BS Option price when we increase the
    simulation. The argument for this\n is simular to the argument
    that Binomial model option price tends to Black Scholes\n (
    continuos) model when we increase the number of nodes ( can be
    visualized\n  as number of simulations for Monte-Carlo');

%b & c Out Paths vs Vol
sigma = 0.05:0.05:0.40;
PercentageOutOfPaths = zeros;
P1 =zeros;
L1 = zeros;
U1 = zeros;
BS_PutPrice = zeros;
r_array = r*ones(n_sims,1);
for i=1:length(sigma)
    [St, Within_Barrier,]= GetSimulatedGBMStockPrice(S0,Sb,r,sigma(
    i),N,dt,n_sims,'D-O-P');
    PercentageOutOfPaths(i) = 1-sum(Within_Barrier)/length(
    Within_Barrier);
    [P1(i),L1(i),U1(i)]=MC_Option_Price(St(:,N+1),K,r_array(:,1),T,
    'EP',Within_Barrier,n_sims);
    BS_PutPrice(i) = bs_put_price(S0,K,r,T,sigma(i),delta);
end
figure;
plot(sigma,PercentageOutOfPaths);
title('Percentage of knocked out option price path vs Volatality');
xlabel('sigma') % x-axis label
```

```
ylabel('% of paths knocked out crossing barrier') % y-axis label

fprintf('We can see from the above graph the percentage increases as sigma grows.\n This is due to the fact that as sigma becomes larger the stock is highly volatile. This \n creates more no of paths under MC  simulation that crosses the barrier.');
figure; hold on;
p2 = plot(sigma,P1);
p3 = plot(sigma,L1);
p4 = plot(sigma,U1);
p5 = plot(sigma,BS_PutPrice);
legend([p2,p3,p4,p5],'Price','Upper Bound', 'Lower Bound','BS Put Price');
title('Option Price vs Volatality');
xlabel('sigma') % x-axis label
ylabel('Option Price'); % y-axis label
fprintf(' We can see that BS- option price increases as volatality increases. This is consistent\n with our expectation. For the down and put option there are two opposing effects due to Vol.\n As Vol increases the probability for option to be in money increases and at the same time \n the the probablity to be knocked out (crossing the barrier) also increases. Thus we see the graph to \n change trend after sigma crosses 0.3');
%%3 Exotic Options in Complicated Market Structures
clear;
r0 = 0.05;
beta = 0.05;
alpha = 0.6;
sigma11 = 0.1;
sigma12 = 0.2;
sigma21 = 0.3;
S10 = 10;
S20 = 10;
delta = 0.1;
coeff_dt(1)= alpha;
coeff_dt(2) =beta;
coeff_dw(1) = delta;
T = 1;
dt = 1/250;
N = T/dt;
n_sims = 1000;
r = zeros(n_sims,N+1);
randn('seed',0);

for i=1:n_sims
[r(i,:),] = Euler_Discretization(r0, coeff_dt, coeff_dw ,dt,N, 'R');
end
figure;
histogram(r(:,N+1));
title('Histogram of rates simualted from Euler Discretization');
xlabel('Interest rate') % x-axis label
ylabel('Frequency') % y-axis label


%3 b Simulate one trajectory
T = 100;
```

```
dt = 1/52;
N = T/dt;
n_sims = 1000;
r = zeros(n_sims,N+1);
randn('seed',0);
[r,]= Euler_Discretization(r0, coeff_dt, coeff_dw ,dt,N, 'R');
figure;
x = linspace(0,T,N+1);
plot(x,r);
title('Simulated path for interest rates using Euler Discretization
    ');
xlabel('Time in Days') % x-axis label
ylabel('Rate') % y-axis label

%3 c
T= 0.5;
K=10;
dt = 1/250;
N = T/dt;
n_sims = 10000;

coeff_dt2(1)= 1;
coeff_dw2(1) = sigma11;
coeff_dw2(2) = sigma12;

S1 = zeros(n_sims,N+1);
r = zeros(n_sims,N+1);
randn('seed',0);
for i=1:n_sims
    [r(i,:),rnd]= Euler_Discretization(r0, coeff_dt, coeff_dw ,dt,N
    , 'R');
    y= vertcat(r(i,:),rnd);
    [S1(i,:),] = Euler_Discretization(S10, coeff_dt2, coeff_dw2 ,dt
    ,N, 'S1',y);
 end
Within_Barrier = ones(n_sims,1);

[P1,L1,U1]=MC_Option_Price(S1(:,N+1),K,mean(r,2),T,'EC',
    Within_Barrier,n_sims);
% check if discounting using r_0 or r_t
fprintf('European Call Option Estimate Price = %f with 95 perc conf
     (%f,%f)\n',P1,L1,U1);

% 3 d
coeff_dw3(1) = sigma21;
S1 = zeros(n_sims,N+1);
S2 = zeros(n_sims,N+1);
r = zeros(n_sims,N+1);
randn('seed',0);
for i=1:n_sims
    [r(i,:),rnd]= Euler_Discretization(r0, coeff_dt, coeff_dw ,dt,N
    , 'R');
    y= vertcat(r(i,:),rnd);
    [S1(i,:),rnd2] = Euler_Discretization(S10, coeff_dt2, coeff_dw2
     ,dt,N, 'S1',y);
    y1= vertcat(r(i,:),S1(i,:),rnd);
```

```
    [S2(i,:),] = Euler_Discretization(S20, coeff_dt, coeff_dw3 ,dt,
    N, 'S2',y1);
end
Within_Barrier = ones(n_sims,1);
S1max = max(S1,[],2);
S2max = max(S2,[],2);
St_Calc = max(S1max,S2max);

[P1,L1,U1]=MC_Option_Price(St_Calc,K,mean(r,2),T,'EC',
    Within_Barrier,n_sims);
% check if discounting using r_0 or r_t
fprintf('Option Estimate Price = %f with 95 perc conf (%f,%f)\n',P1
    ,L1,U1);

%%4 Hedging, Large Price Movements, and Transaction Costs
%a)
clear;
S0 = 100;
Sb = -1;
N_Days = 365;
T  = 30/365;
K  = 100;
sigma = 0.3;
mu = 0.2;
yield = 0;
trading_hrs = 8;
N = 12*trading_hrs*N_Days*T;
r  = 0.05* ones(1,N+1);
dt = (5/60)/trading_hrs/N_Days;
randn('seed',0);
[S_T, Within_Barrier,]= GetSimulatedGBMStockPrice(S0,Sb,mu,sigma,N,
    dt,1,'P');

figure;
x = linspace(0,(N)/(trading_hrs*12),N+1);
plot(x,S_T);
title('Simulated path for Stock price using growth rate');
xlabel('Time in Days') % x-axis label
ylabel('Price') % y-axis label
%b)
T_array = 2*T:-(1/(365*trading_hrs*12)):T;
K_array = K*ones(1,N+1);
[BSCallPrice] = bs_call_price(S_T,K,r,T_array,sigma,yield);
V_0 = BSCallPrice(1);
figure;
x = linspace(0,(N)/(trading_hrs*12),N+1);
plot(x,BSCallPrice);
title('BS Price for the above simulated Stock price using growth
    rate');
xlabel('Time in Days (60- time to expiration)') % x-axis label
ylabel('Price') % y-axis label

%c)
t_cost= 0;
d1 = getd1(S_T,K,r,T_array,sigma);
[V_T,~,Call_Delta_T]= Get_ReplicatingPortfolio(V_0,S_T,r,dt,d1,N,
    t_cost,'C');
```

```
x = linspace(0,(N)/(trading_hrs*12),N+1);
figure;
plot(x, S_T);
title('Simulated path for Stock price ');
xlabel('Time in Days') % x-axis label
ylabel('Stock Price') % y-axis label

figure;hold on;
p1 = plot(x, BSCallPrice);
p2= plot(x,V_T);
title('Simulated BS Call Price and Replicating Portfolio');
xlabel('Time in Days') % x-axis label
ylabel('Price') % y-axis label
legend([p1,p2],'BS Call Price', 'Replicating Portfolio');
figure;
plot(x, Call_Delta_T);
title('Hedge ratio for simulated stock price');
xlabel('Time in Days') % x-axis label
ylabel('Delta of Call') % y-axis label


%d)
t_cost = 0;
jump_down_v = zeros(1,N+1);
jump_down_v(1,N/2) = -0.1;

randn('seed',0);
[S_T_down_jump,~,~]= GetSimulatedGBMStockPrice(S0,Sb,mu,sigma,N,dt
    ,1,'P',jump_down_v);
[BSCallPrice_down] = bs_call_price(S_T_down_jump,K,r,T_array,sigma,
    yield);
d1_down = getd1(S_T_down_jump,K,r,T_array,sigma);
V_0_down = BSCallPrice_down(1);
[V_T_down,~,Call_Delta_T_down]= Get_ReplicatingPortfolio(V_0_down,
    S_T_down_jump,r,dt,d1_down,N,t_cost,'C');
figure;
plot(x, S_T_down_jump);
title('Simulated path for Stock price with -ve jump');
xlabel('Time in Days') % x-axis label
ylabel('Stock Price') % y-axis label

figure;hold on;
p1 = plot(x, BSCallPrice_down);
p2= plot(x,V_T_down);
title('Simulated BS Call Price and Replicating Portfolio');
xlabel('Time in Days') % x-axis label
ylabel('Price') % y-axis label
legend([p1,p2],'BS Call Price', 'Replicating Portfolio');
figure;
plot(x, Call_Delta_T_down);
title('Hedge ratio for simulated stock price');
xlabel('Time in Days') % x-axis label
ylabel('Delta of Call') % y-axis label
fprintf('The stock price with the jump is as shown in the graph. We
     can see the sudden\n jump down in stock price causes the
    hedging to be ineffective at T=15 days(jump time). The
    replicating\n portfolio falls more than the call price for that
```

```
moment (hedging is not continuos). From the next time(immediately after\n the jump is absorbed )the replicating portfolio closely follows the call price (with diff = diff in drop between call and replicating portfolio). If we try to synthesize\n the call by using our replicating portfolio, we will suffer loss = %.3f\n' ,BSCallPrice_down(N/2+1)-V_T_down(N/2+1));

%e)
t_cost=0;
jump_up_v = zeros(1,N+1);
jump_up_v(1,N/2) = 0.1;
randn('seed',0);
[S_T_up_jump,~,~]= GetSimulatedGBMStockPrice(S0,Sb,mu,sigma,N,dt,1,'P',jump_up_v);
[BSCallPrice_up] = bs_call_price(S_T_up_jump,K,r,T_array,sigma,yield);
d1_up = getd1(S_T_up_jump,K,r,T_array,sigma);

V_0_up = BSCallPrice_up(1);
[V_T_up,~,Call_Delta_T_up]= Get_ReplicatingPortfolio(V_0_up,S_T_up_jump,r,dt,d1_up,N,t_cost,'C');
figure;
plot(x, S_T_up_jump);
title('Simulated path for Stock price with -ve jump');
xlabel('Time in Days') % x-axis label
ylabel('Stock Price') % y-axis label

figure;hold on;
p1 = plot(x, BSCallPrice_up);
p2= plot(x,V_T_up);
title('Simulated BS Call Price and Replicating Portfolio');
xlabel('Time in Days') % x-axis label
ylabel('Price') % y-axis label
legend([p1,p2],'BS Call Price', 'Replicating Portfolio');
figure;
plot(x, Call_Delta_T_up);
title('Hedge ratio for simulated stock price');
xlabel('Time in Days') % x-axis label
ylabel('Delta of Call') % y-axis label
fprintf('We can see similar observation. Even though stock\n jumps up, due to limitation of non\n continuos time hedging, we will not able to track the call price \n during the jump.\n The differnce in Call Price is abosrbed slowly and the replicating portfolio falls back again.\n');
%f)
t_cost =20;
[V_T,B_T,Call_Delta_T]= Get_ReplicatingPortfolio(V_0,S_T,r,dt,d1,N,t_cost,'C');
figure;hold on;
p1 = plot(x, BSCallPrice);
p2 = plot(x,V_T);
title('Simulated BS Call Price and Replicating Portfolio');
xlabel('Time in Days') % x-axis label
ylabel('Price') % y-axis label
legend([p1,p2],'BS Call Price', 'Replicating Portfolio');
```

```
sprintf('With the 20 basis points transaction cost,the replicating portfolio value diverges from \n the call value. This explanation falls with the fact that\n as time increases due to the volatile stock price movements\n we end up adjusting our portfolio more number of times\n' );
figure;
plot(x, Call_Delta_T);
title('Hedge ratio for simulated stock price');
xlabel('Time in Days') % x-axis label
ylabel('Delta of Call') % y-axis label
%5 Heston model

%a)
clear;
S0= 100;
v_0 = 0.01;
K= 100;
r= 0.04;
T=0.5;
trading_hrs = 8;
%dt is 5 minute interval
N = 12*trading_hrs*365*T ;
dt = 1/N;
lambda=0;
rho = -0.5;
kappa = 6;
theta = 0.02;
div_yield = 0;

sigma= 0.3;
% Assuming mu = 0.2
mu = 0.2;
randn('seed',0);
[S_T,v_T] = CIRProcess(S0,v_0,mu,sigma,rho,kappa, theta,N,dt);
figure;
x = linspace(0,(N)/(trading_hrs*12),N+1);
plot(x,S_T);
title('Stock price simulated using CIR volatality model');
xlabel('Time in Days') % x-axis label
ylabel('Price') % y-axis label
figure;
plot(x,v_T);
title('CIR volatality model');
xlabel('Time in Days') % x-axis label
ylabel('Volatality') % y-axis label
T_array = T:-(1/(365*trading_hrs*12)):0;
BSCall_Price = bs_call_price(S_T(1,1),K,r,T_array(1,1),sqrt(theta),div_yield);


HestonCallPrice = HestonModel(S_T(1,1),K,v_T(1,1),r,rho,sigma,kappa,lambda,theta,T_array(1,1));
fprintf('BS Call Option Price = %f Heston Call Option Price =%f\n',BSCall_Price,HestonCallPrice);


%b)
```

```
S01 = 70;
S02 = 130;
S = S01:1:S02;
H_Call = zeros;
BS_Call = zeros;
for i=1:length(S)
    H_Call(i) = HestonModel(S(1,i),K,v_T(1,1),r,rho,sigma,kappa,
    lambda,theta,T);
    BS_Call(i) = bs_call_price(S(1,i),K,r,T,sqrt(theta),div_yield);
end
figure;hold on;
p1 = plot(S,H_Call);
p2=plot(S,BS_Call);
title('Heston vs BS Call Price');
xlabel('Initial Stock Price') % x-axis label
ylabel('Call Price') % y-axis label
legend([p1,p2],'Heston Option Price','BS Option Price');
figure;
plot(S,H_Call- BS_Call);
title('Heston - BS Call Price vs Stock Price');
xlabel('Initial Stock Price') % x-axis label
ylabel('Diff in Call Price') % y-axis label


sprintf('We can see from the graph, the Heston model for option
    price converges to BS option pricing model, for in the money
    call (S0 > K)\n. The reason for this is Heston model
    incorporates time varying volatility with negative rho. This
    makes out of money\n call options (low stock price) to have
    lesser probablity to be in the money when compared\n to in the
    money (for -ve rho). CIR model for vol being mean reverting.\n'
    );
%c)
S01 = 60;
S02 = 160;
S = S01:1:S02;
rho =-0.5;
implied_vol = zeros(1,length(S));
for i=1:length(S)
    H_Call(i) = HestonModel(S(1,i),K,v_T(1,1),r,rho,sigma,kappa,
    lambda,theta,T);
    implied_vol(i)= getHS_Call_Implied_Vol(S(1,i),K,r,T,div_yield,
    H_Call(i),sigma);
end

figure;
plot(S,implied_vol);
title('Implied Volatality');
xlabel('Stock Price') % x-axis label
ylabel('Implied volatality') % y-axis label
%d
rho=0.5;
implied_vol = zeros(1,length(S));
for i=1:length(S)
    H_Call(i) = HestonModel(S(1,i),K,v_T(1,1),r,rho,sigma,kappa,
    lambda,theta,T);
```

```
    implied_vol(i)= getHS_Call_Implied_Vol(S(1,i),K,r,T,div_yield,H_Call(i),sigma);
end

figure;
plot(S,implied_vol);
title('Implied Volatality');
xlabel('Stock Price') % x-axis label
ylabel('Implied volatality') % y-axis label

fprintf('The volatality smile is right skewed for negative rho and left skewed for positive rho.\n
In the plot, we used stock price from 60 to 160 to get the detail of the full vol plot.\n
When rho <0, then the Heston model minimum occurs at Stcok price close to K (from below),\n
When rho >0, then the Heston model minimum occurs at Stock price close to K (from above)');
```

Functions:

```
function [ bs_call_price] = bs_call_price(Stock_price,Strike_price,rate,T,sigma,yield)
%Returns the BS Call Option price using the given paremeters
d1 = getd1(Stock_price, Strike_price, rate,T, sigma);
d2 = getd2(Stock_price, Strike_price, rate,T, sigma);
bs_call_price = Stock_price.*exp(-yield.*T).*normcdf(d1) - Strike_price.*exp(-rate.*T).*normcdf(d2);
end
```

```
function [ put_price ] = put_price(Stock_price,Strike_price, rate,T,sigma,yield)
%Returns the BS Put Option price using the given paremeters
%   Detailed explanation goes here
put_price = Strike_price.*exp(-rate.*T).*normcdf(-getd2(Stock_price, Strike_price, rate,T, sigma)) -Stock_price.*exp(-yield.*T).*normcdf(-getd1(Stock_price, Strike_price, rate,T, sigma));
end
```

```
function [ S_T,v_T ] = CIRProcess( S0,v_T0,mu,sigma,rho,kappa,theta,T,dt )
% Returns the stock price and volatility simulated using CIR process
S_T = zeros(1,T+1);
v_T = zeros(1,T+1);
v_T(1) = v_T0;
S_T(1) = S0;
for i=2:T+1
    dz1 = randn;
    n2 = randn;
    dz2 = (rho)*dz1 + sqrt(1-rho^2)*n2;
    S_T(i) = S_T(i-1)+ S_T(i-1)* ( mu*dt + sqrt(v_T(i-1)*dt)*dz1 );
    v_T(i) = v_T(i-1) + kappa*(theta- v_T(i-1))*dt + sigma*dz2*sqrt(v_T(i-1)*dt);
```

```
end
```

```
function [ x ,rnd_gen] = Euler_Discretization( x_0,coeff_dt,
    coeff_dw ,dt,T,type,y,jump_perc)
% Returns the simulated values of x using the Euler Discretization
    method.
% The process uses the coefficients of dt, dw ,x0, dependent
    variables y and jump points
% Type can be
% a)R - Euler Discretization process to simulate rate movements
% b)S1/S2/S - Euler Discretization process to simulate Stock Price
% movements
    if (~exist('y', 'var'))
        y = zeros(1,T+1);
    end
    if (~exist('jump_perc', 'var'))
         jump_perc = zeros(1,T+1);
    end
    x = zeros;
    x(1) = x_0;
    rnd_gen = zeros(1,T);
    for i=2:T+1
        if jump_perc(i)== 0
            [sigma,rnd_gen(:,i)]= get_sigma_x(coeff_dw,x(i-1),type,
    y(:,i));
            x(i) = x(i-1)+ dt*get_mu_x(coeff_dt,x(i-1),type,y(:,i))
     + sqrt(dt)*sigma;
        else
            x(i) = x(i-1)*(1+jump_perc(1,i));
        end
    end
end
```

```
function [ c ] = get_mu_x( coeff_dt,x,type,y)
% Returns the coefficient of dt in Euler discretization for
    different
% simulations. Type can be
% a)R - Euler Discretization process to simulate rate movements
% b)S1/S2/S - Euler Discretization process to simulate Stock Price
% movements
switch type
    % Call
    case 'R'
         alpha = coeff_dt(1);
         beta = coeff_dt(2);
         c= alpha*(beta - x);
    %European Put
    case 'S1'
        c = y(1)*x;
    case 'S2'
        c = y(1)*x;
    case 'S'
        c = coeff_dt(1);
    otherwise
        c = 0;
```

```
end

end
```

```
function [ PayOff ] = get_PayOff(StockPrice,K,optionType)
% Returns the payoff for different types of options

switch optionType
    %European Call
    case 'EC'
        PayOff = max(StockPrice- K,0);
    %European Put
    case 'EP'
        PayOff = max(K -StockPrice,0);
end
end
```

```
function [ V_T,B_T,Delta_T ] = Get_ReplicatingPortfolio(V0,S_T,r,dt,d1,N,cost,type)
% Return the replicating portfolio for the simulated stock price
% V_T - Value of the replicating portfolio
% B_T - Vaule in bond
% Delta_T - Number of shares
V_T = zeros(1,N+1);
B_T = zeros(1,N+1);
switch(type)
    case 'C'
        Delta_T = normcdf(d1);
    case 'P'
        Delta_T = normcdf(d1)-1;
    otherwise
        warning('Unexpected option type.')
end
V_T(1) = V0;
B_T(1) = V_T(1)- Delta_T(1)*S_T(1);

for i=2:N+1
    delta_change = abs(Delta_T(i)-Delta_T(i-1));
    V_T(i) = Delta_T(i-1)*S_T(i) + B_T(i-1)*exp(r(1)*dt) - delta_change*S_T(i)*cost/10^4;
    B_T(i)=  V_T(i) - Delta_T(i)*S_T(i);
end

end
```

```
function [ c ,rnd_gen] = get_sigma_x( coeff_dw,x,type,y)
% Returns the coefficient of dw and random number used to generate it
% in Euler discretization for different simulations.
% Type can be
% a)R - Euler Discretization process to simulate rate movements
% b)S1/S2/S - Euler Discretization process to simulate Stock Price
% movements
```

```
rnd_gen = zeros;
switch type
    % Call
    case 'R'
         delta = coeff_dw(1);
         c= delta*sqrt(x)*randn;
         rnd_gen = randn;
    %European Put
    case 'S1'
        sigma11 = coeff_dw(1);
        sigma12 = coeff_dw(2);
        %rnd_gen(1) = randn;
        rnd_gen = randn;
        c = sigma11*sqrt(x)*y(2) + sigma12*x*rnd_gen;
    case 'S2'
        sigma21 = coeff_dw(1);
        S1 = y(2);
        c = sigma21*(S1-x)*y(3);
    case 'S'
        c = coeff_dw(1)*randn;
    otherwise
        c = 0;
end

end
```

```
function [ A_j, B_j ] = getA_Bj(u,r,rho,sigma,kappa,lambda,theta, t
    , j )
%UNTITLED4 Summary of this function goes here
%   Detailed explanation goes here
b_j = kappa + lambda -(j==1)*rho*sigma;
u_j = (j==1)*1/2 - (j==2)*1/2;

z1 = complex(-b_j,rho*sigma*u);
z2 = complex(-u.^2, 2*u_j*u);

d_j = sqrt(z1.^2- (sigma^2)*z2);


g_j = (-z1+ d_j)./(-z1 - d_j);

z3 = complex(0,r*u*t);
z5 = (kappa*theta/(sigma^2))*((d_j-z1)*t -2*log( (1- exp(d_j*t).*
    g_j)./(1-g_j) ));


A_j= z3 + z5;
B_j = 1/(sigma^2)*(d_j-z1).*(1- exp(d_j*t))./(1-g_j.*exp(d_j*t));
end
```

```
function [ d1 ] = getd1( Stock_price, Strike_price, rate,T, sigma)
%Returns the d1 of B-S option pricing formula
d1 = (log(Stock_price/Strike_price) + (rate + (sigma*sigma)*0.5).*T
    )./(sigma.*sqrt(T));
end
```

```
function [ d2 ] = getd2( Stock_price, Strike_price, rate,T, sigma)
%Returns the d2 of B-S option pricing formula
d2 = getd1(Stock_price, Strike_price, rate,T, sigma)- sigma.*sqrt(T);
end
```

```
function [ implied_vol ] = getHS_Call_Implied_Vol( S,K,r,T,yield,CallPrice,guess)
% Return the implied vol of BS Option Price using Heston Model's price

result = @(x) bs_call_price(S,K,r,T,x,yield) - CallPrice;
implied_vol = fsolve(result,guess);
end
```

```
function [ real_P_j] = getReal_Pj(u,S_T,K,v_T,r,rho,sigma,kappa,lambda,theta, t, j )
 x_T = log(S_T);
 [A_j,B_j]=getA_Bj(u,r,rho,sigma,kappa,lambda,theta,t,j);

 z1 = complex(0,u*x_T);
 phi_j = exp(A_j + B_j.*v_T + z1);

 z2 = complex(0,-u*log(K));
 z3 = complex(0,u);

 real_P_j =  real(exp(z2).*phi_j./z3);
 %real_P_j(isnan(real_P_j)) = 0 ;

 end
```

```
function [ St, Knocked_Out,Knocked_In ] = GetSimulatedGBMStockPrice(S0,Sb,r,sigma,N,dt,n_sims,optionType,jump_perc)
% Returns the "n_sims" number of Stock Prices simulated using geometric
%brownian motion and the boolean array of paths in which the stock prices
%got knocked_out/knocked_in. The function takes the type of crossing
%(crossing from below or above) and the jump points as parameters

switch optionType
    % Up and Out Call
    case 'U-O-C'
        c = -1;
    % Up and In Call
    case 'U-I-C'
        c= 1;
    % Down and Out Put
    case 'D-O-P'
        c = 1;
    % Down and In Put
    case 'D-I-P'
```

```
    otherwise
        c = 0;
end

if (~exist('jump_perc', 'var'))
    jump_perc = zeros(1,N+1);
end

a1 = r-sigma^2/2;
St = zeros(n_sims,N+1);
Knocked_Out = ones(n_sims,1);
Knocked_In = zeros(n_sims,1);

randn('seed',0);
for i=1:n_sims
    St(i,1)=S0;
    notknocked = 1;
    for j=1:N
        if jump_perc(1,j)== 0
            St(i,j+1) = St(i,j)*(exp(a1*dt + sigma*sqrt(dt)*randn))
    ;
        else
            St(i,j+1) = St(i,j)*(1+jump_perc(1,j));
        end
        if( ((St(i,j+1)-Sb)*c <0) && (notknocked))
            Knocked_Out(i)=0;
            Knocked_In(i)=1;
            notknocked = 0;
        end
    end
end
end
```

```
function [call_prices, std_errs] = Heston(S0, r, V0, eta, theta,
    kappa, strike, T, M, N)
%   Compute European call option price using the Heston model and a
%   conditional Monte-Carlo method
%
%       [call_prices, std_errs] = Heston(S0, r, V0, eta, theta,
    kappa,
%       strike, T, M, N)
%
%
    ***********************************************************************
% ACKNOWLEDGMENTS:
% Thanks to Roger Lee for his MSFM course at the University of
    Chicago
%
%
    ***********************************************************************
%   INPUTS:
%
%   S0      - Current price of the underlying asset.
%
```

```
%   r        - Annualized continuously compounded risk-free rate of return
%              over the life of the option, expressed as a positive decimal
%              number.
%
%        Heston Parameters:
%
%   V0     - Current variance of the underlying asset
%
%   eta    - volatility of volatility
%
%   theta  - long-term mean
%
%   kappa  - rate of mean-reversion
%
%
%   strike      - Vector of strike prices of the option
%
%   T           - Time to expiration of the option, expressed in years.
%
%   N           - Number of time steps per path
%
%   M           - Number of paths (Monte-Carlo simulations)
%
%
%   OUTPUTS:
%
%   call_prices      - Prices (i.e., value) of a vector of European call options.
%
%   std_err          - Standard deviation of the error due to the Monte-Carlo
%                      simulation:
%                      (std_err = std(sample)/sqrt(length(sample)))
%
%
% ************************************************************************
%
%   Example:
%
%     S0 = 100;
%     r = 0.02;
%     V0 = 0.04;
%     eta = 0.7;
%     theta = 0.06;
%     kappa = 1.5;
%     strike = 85:5:115;
%     T = 0.25;
%
%     M = 2000;  % Number of paths.
%     N = 250;   % Number of time steps per path
%
%       [call_prices, std_errs] = Heston(S0, r, V0, eta, theta,
```

```
    kappa,
%       strike, T, M, N)
%
% call_prices =
%
%   15.9804   11.4069    7.2125    3.9295    2.1213    1.2922    0.8625
%
%
% std_errs =
%
%    0.0198    0.0263    0.0329    0.0367    0.0357    0.0315    0.0268
%
%
% ***********************************************************************
% Rodolphe Sitter - MSFM Student - The University of Chicago
% November 2009
% ***********************************************************************


% Memory allocation for the variance paths
V = [V0*ones(M,1), zeros(M,N)];
Vneg = [V0*ones(M,1), zeros(M,N)]; % Antithetic variate for Monte-Carlo

% Normal random variables sample needed: M trajectories of N time steps
W = randn(M,N);

% Time step
dt = T/N;

% Simulation of N-step trajectories for the Variance of the underlying asset
for i = 1:N

    V(:,i+1) = V(:,i) + kappa*(theta-V(:,i))*dt+eta*sqrt(V(:,i)).*W(:,i)*sqrt(dt);
    % We don't want to variance to be negative
    V(:,i+1) = V(:,i+1).*(V(:,i+1)>0);

        % Antithetic variates
        Vneg(:,i+1) = Vneg(:,i)+kappa*(theta-Vneg(:,i))*dt - eta*sqrt(Vneg(:,i)).*W(:,i)*sqrt(dt);
        % We don't want to variance to be negative
        Vneg(:,i+1) = Vneg(:,i+1).*(Vneg(:,i+1)>0);
end

% The implied variance is equal to the time averaged realized variance
% We use numerical integration (trapezoidal rule) to compute it:
ImpVol = sqrt((1/2*V(:,1) + 1/2*V(:,end) + sum(V(:,2:end-1),2))*dt/
```

```
T);

% Antithetic variates
ImpVolneg = sqrt((1/2*Vneg(:,1) + 1/2*Vneg(:,end) + sum(Vneg(:,2:end-1),2))*dt/T);



% Computation of Heston call prices using Antithetic Variates and the
% Black-Scholes formula with the time averaged realized variance

std_errs = nan(length(strike),1);  % Memory allocation
call_prices = nan(length(strike),1);

for j=1:length(strike)

    % Antithetic variates
    Sample = (BS(S0,0,strike(j),T,r,r,ImpVol) + BS(S0,0,strike(j),T,r,r,ImpVolneg))/2;

    % Standard deviation of the error
    std_errs(j) = std(Sample)/sqrt(M);

    call_prices(j) = mean(Sample);

end

% Plot the Heston volatility smile (use of blsimpv from the financial toolbox)
% ************************************************************************
% Comment this section of code if you don't want to output the plot *******

% Computation of the Black-Scholes implied volatilities (financial toolbox)
IV = blsimpv(S0, strike, r, T, call_prices', 3);

% Computation of forward log-moneyness from strikes for plot
F = S0*exp(r*T);
moneyness = log(F./strike);

figure;
set(gca,'Fontsize',12,'FontWeight','Bold','LineWidth',2);
plot(moneyness,IV,'-r+','linewidth',2)
grid on; axis tight;
xlabel('Log-Moneyness','interpreter','latex','FontSize',16);
ylabel('Implied Volatility$~\sigma_{imp}$','interpreter','latex',...
    'FontSize',16);
title('HESTON Model - Volatility Skew','interpreter','latex','FontSize',18)
fprintf('\n')
% ************************************************************************
```

```
% Black-Scholes Price function

function Call = BS(S0, t, strike, T, Rgrow, Rdisc, sigma)

F = S0.*exp(Rgrow.*T);

d1 = log(F./strike)./(sigma.*sqrt(T-t))+sigma.*sqrt(T)/2;
d2 = log(F./strike)./(sigma.*sqrt(T-t))-sigma.*sqrt(T)/2;

Call = exp(-Rdisc.*T).*(F.*normcdf(d1) - strike.*normcdf(d2));
```

```
function [ CallPrice ] = HestonModel( S_T,K,v_T,r,rho,sigma,kappa,
    lambda,theta,t)
% Return the Call Price using Heston Model.

    I1 = integral(@(u) getReal_Pj(u,S_T,K,v_T,r,rho,sigma,kappa,
    lambda,theta, t,1),0,500);
    I2 = integral(@(u) getReal_Pj(u,S_T,K,v_T,r,rho,sigma,kappa,
    lambda,theta, t,2),0,500);
    CallPrice = S_T*(0.5+ I1/pi) - K*exp(-r*t)*(0.5+I2/pi);
end
```

```
function [Option_Price,lower_bound, upper_bound] = MC_Option_Price(
    S_T,K,r,T,optionType,Within_Barrier,n_sims)
% Returns the bound of the option price calculated using the given
    PayOff
% function and the simulated stock price path
OptionPayOff = zeros(n_sims,1);
for i=1:n_sims
    OptionPayOff(i,1) = Within_Barrier(i)*get_PayOff(S_T(i),K,
    optionType).*exp(-r(i)*T);
end
Option_Price = mean(OptionPayOff);
lower_bound = Option_Price - 1.96*std(OptionPayOff)/sqrt(n_sims);
upper_bound = Option_Price + 1.96*std(OptionPayOff)/sqrt(n_sims);
end
```

```
function [Option_Price,lower_bound, upper_bound] =
    MC_Option_Price_Direct(S0,K,r,sigma,T,optionType,n_sims)
% Returns the bound of the option price calculated using the given
    PayOff
% function and the initial value of stock price
randn('seed',0);
S_T = zeros;
OptionPayOff = zeros;
for i=1:n_sims
    S_T(i) = S0*exp((r-sigma^2/2)*T + sigma*sqrt(T)*randn);
    OptionPayOff(i) = get_PayOff(S_T(i),K,optionType)*exp(-r*T);
end
Option_Price = mean(OptionPayOff);
```

```
lower_bound = Option_Price - 1.96*std(OptionPayOff)/sqrt(n_sims);
upper_bound = Option_Price + 1.96*std(OptionPayOff)/sqrt(n_sims);
end
```