Alexander Lerma

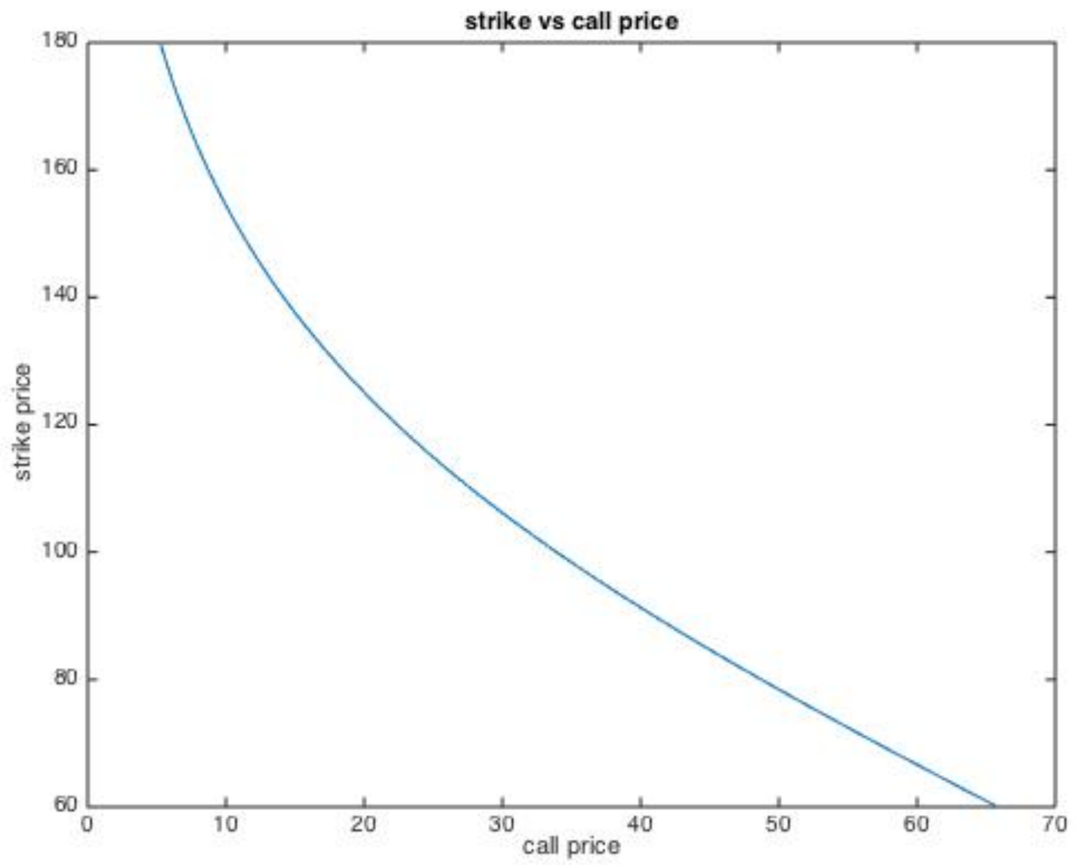
Math 485

Due: April 13, 2017

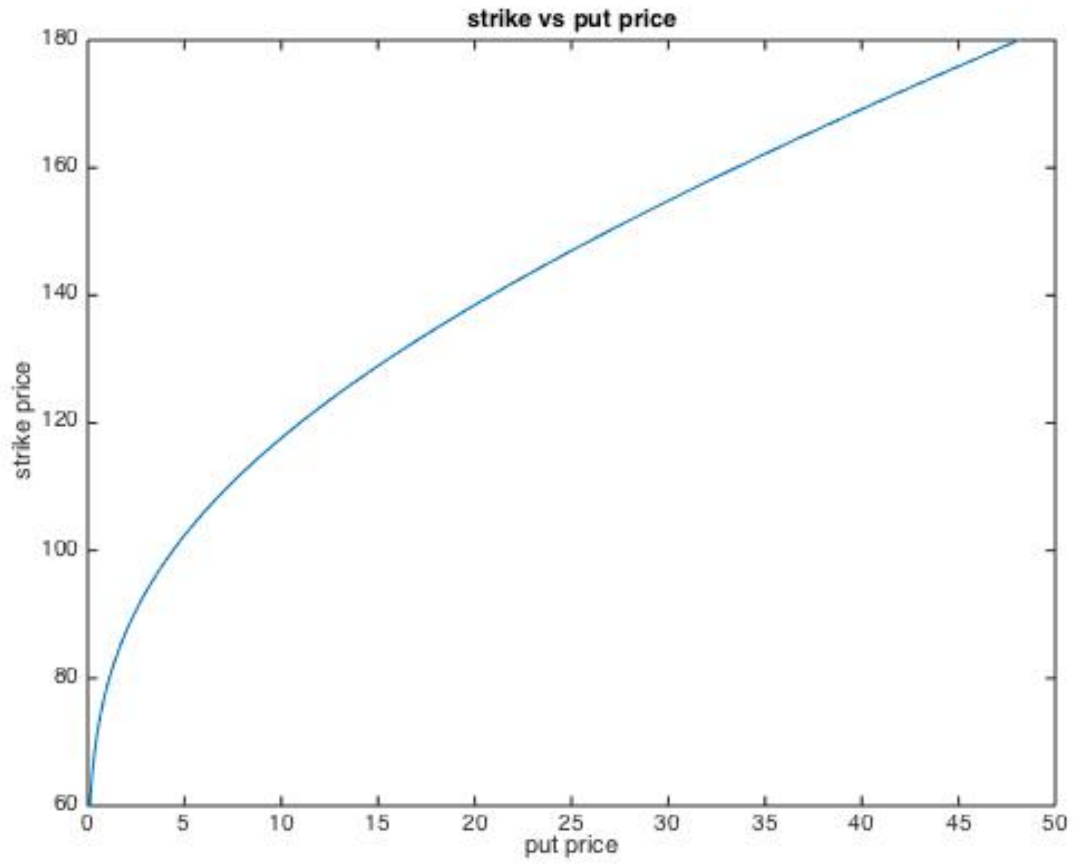
Project 2

## 1. Black-Scholes-Merton

a. Strike vs price for both set of options

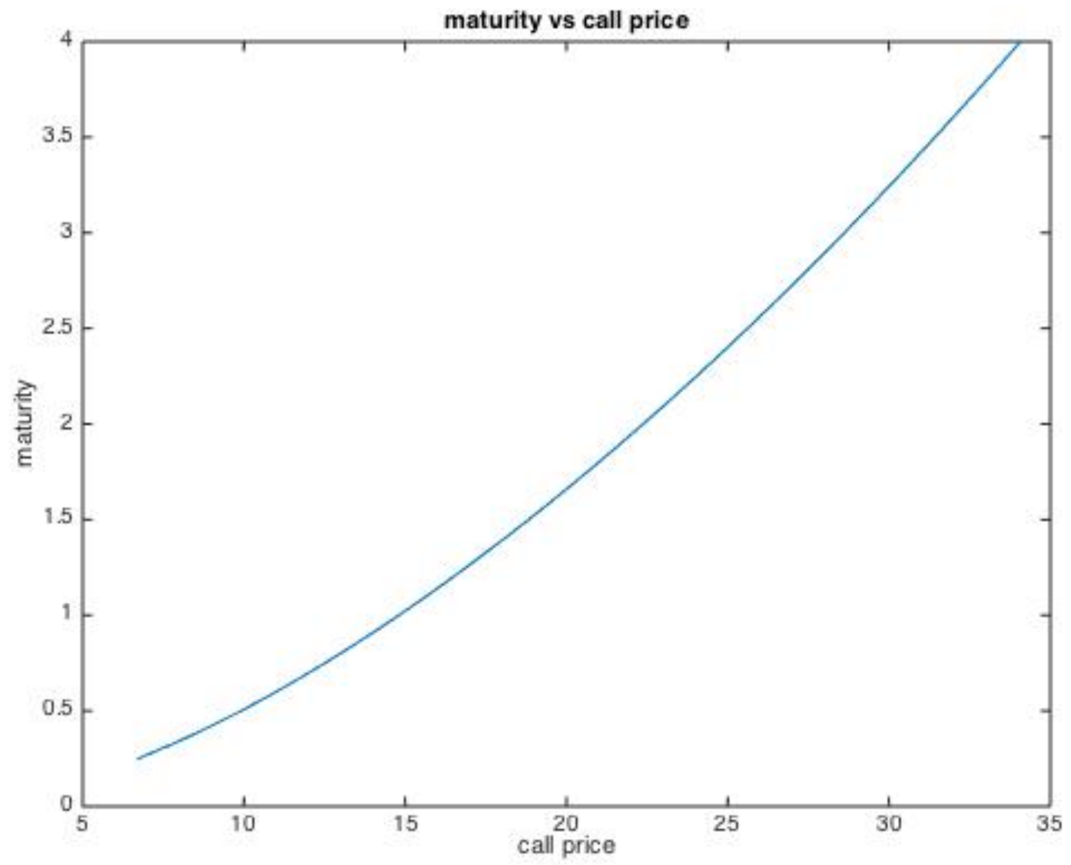


For strike vs call price we see that as call price increases, strike price decreases, they are inversely proportional. The first derivative is always negative, therefore always decreasing. The second derivative is positive, indicating the rate of growth is slowing down. This makes sense because the payoff of a call option is max(0, st – K). If we keep a fixed st, as K increases, the amount of payoff becomes less and less making the option less valuable.

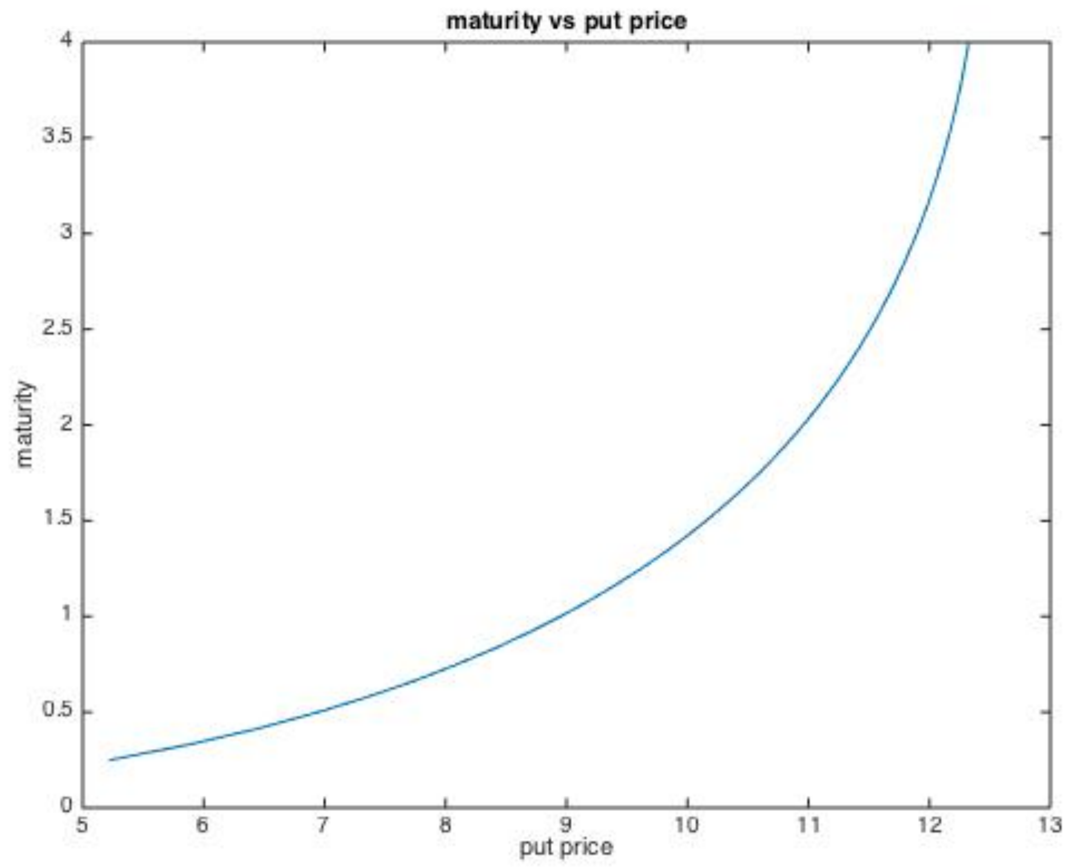


For strike vs put price we see that as put price increases, strike price increases, they are proportional. The first derivative is always positive, therefore always increasing. The second derivative is negative, indicating the rate of growth is slowing down. This makes sense because the payoff of a put option is max(0, K – st). If we keep a fixed st, as K increases, the amount of payoff becomes more making the option more valuable.

b. “maturity vs price” for both sets of options.

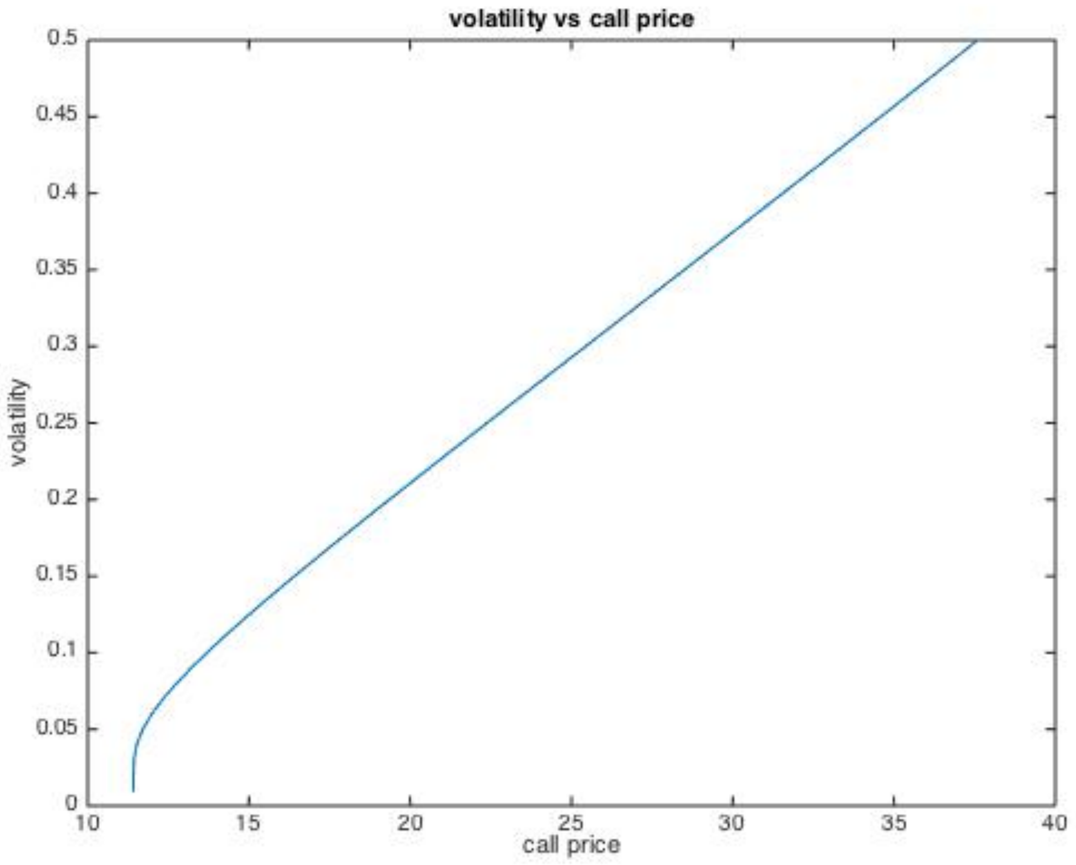


As call price increases, maturity increases.  The first derivative is always positive, therefore always increasing. The second derivative is positive, indicating the rate of growth is increasing. This makes sense because when you are further away from maturity, the option has much more time to fluctuate up or down making it more valuable. When time is very close to maturity, the stock price wont have as much room to change and the option is priced lower.

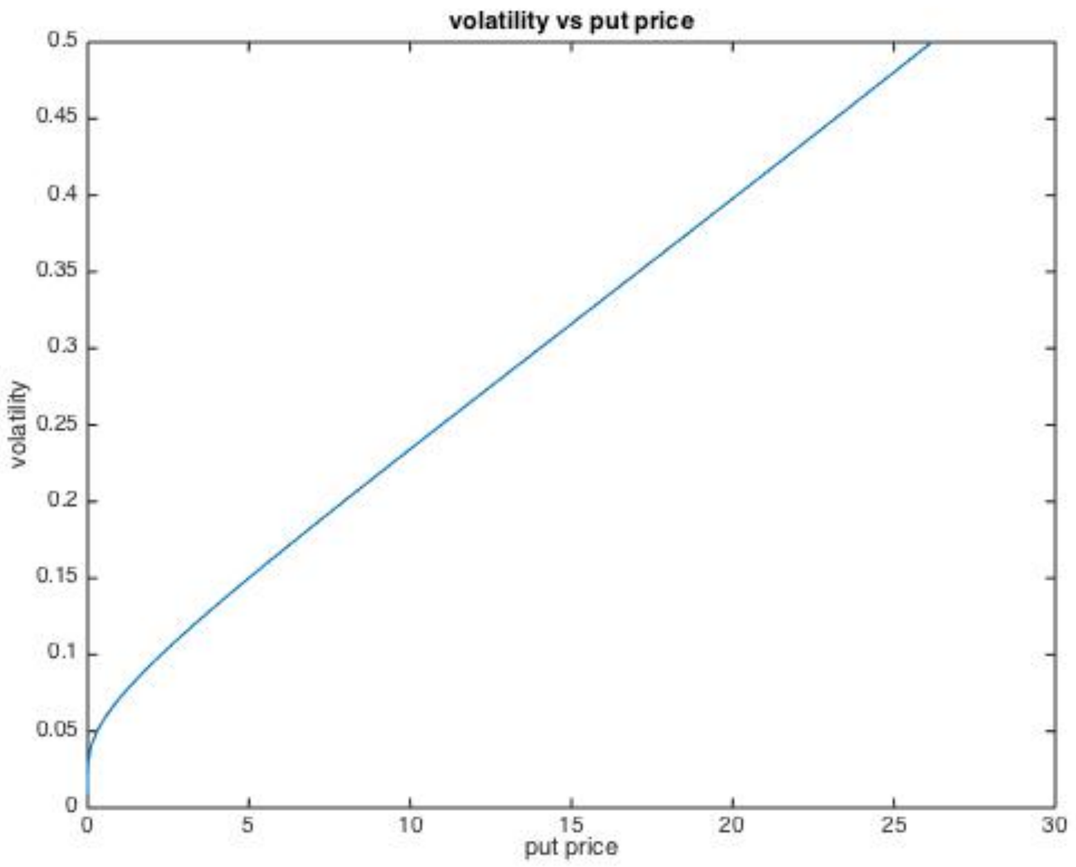


As put price increases, maturity increases. The first derivative is always positive, therefore always increasing. The second derivative is positive, indicating the rate of growth is increasing. This makes sense because when you are further away from maturity, the option has much more time to fluctuate up or down. When time is very close to maturity, the stock price will not have as much room to change and the option is priced lower.

c. “volatility vs price” for both sets of options.

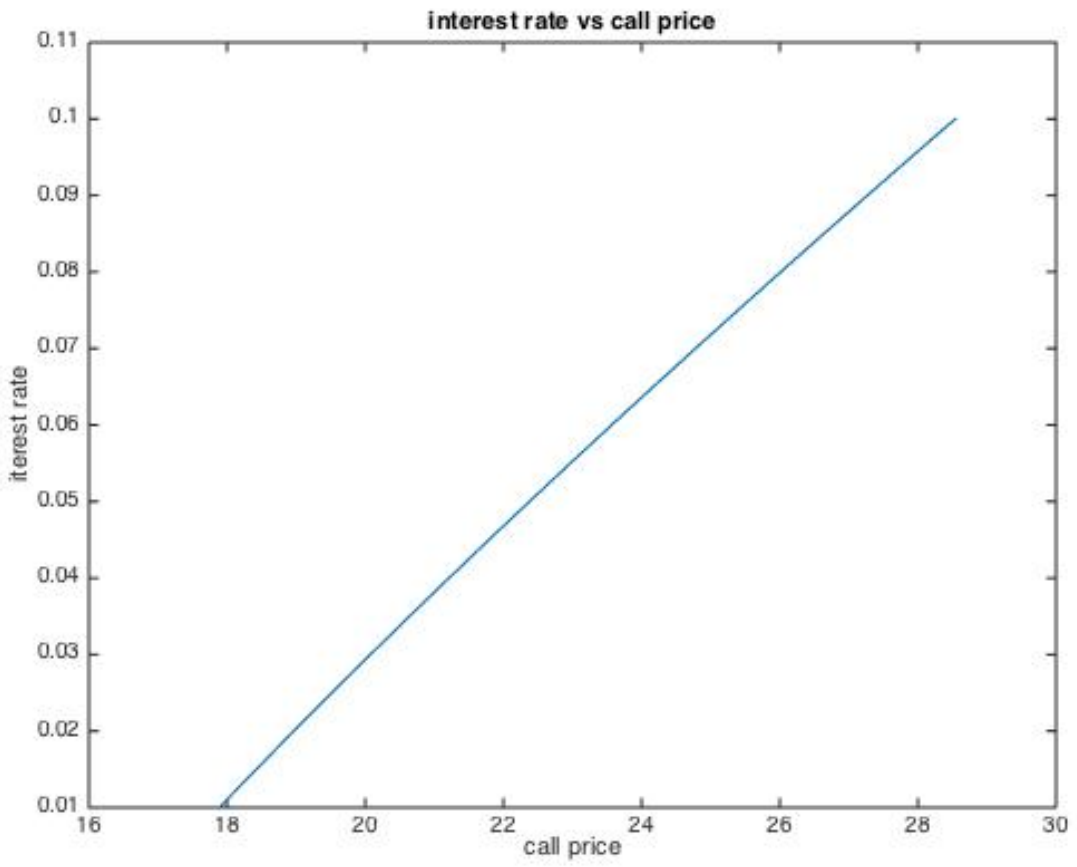


As call price increases, volatility increases. The graph is concave down, increasing and its rate of growth is slowing down. Stocks with high volatility have higher value due to the possibility of the price rapidly increasing. Stocks with low volatility will likely remain consistent and do not have a much potential gain.

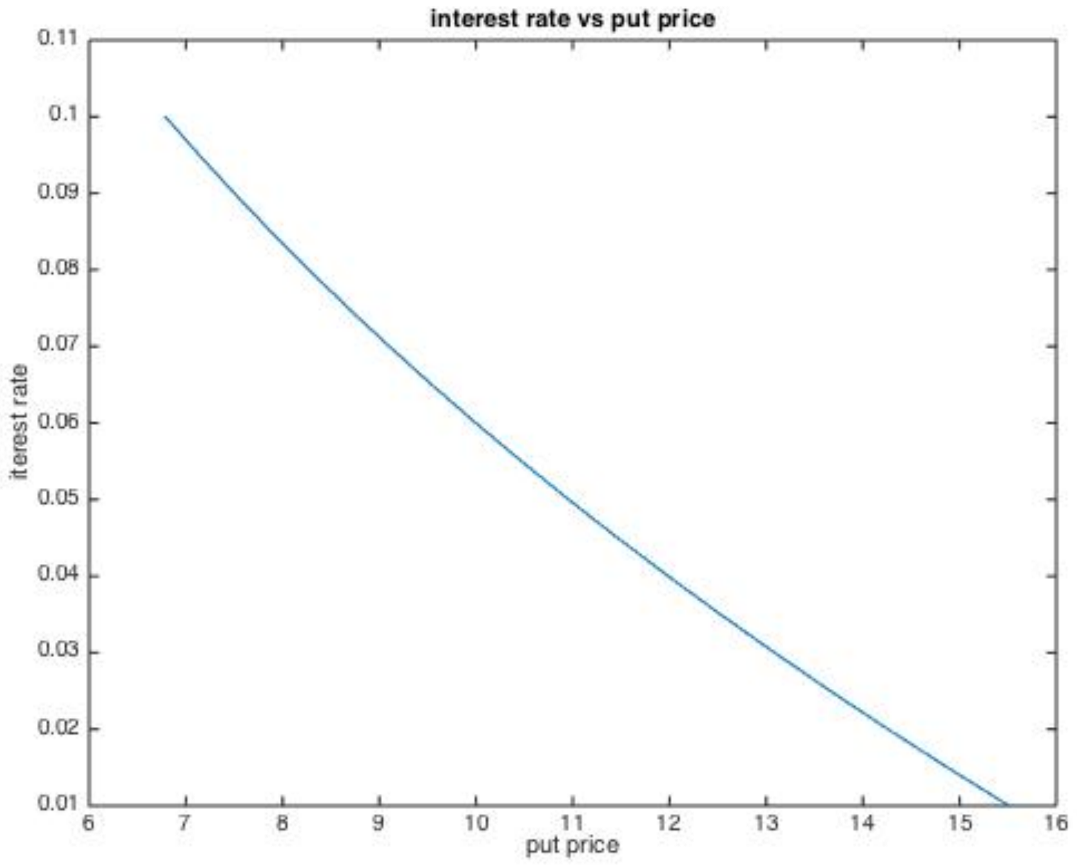


As put price increases, volatility increases. The graph is concave down, increasing and its rate of growth is slowing down. Stocks with high volatility have higher value due to the possibility of the price rapidly increasing. Stocks with low volatility will likely remain consistent and do not have a much potential gain.

d. interest rate vs price for both sets of options.



As call price increases, interest rate increases. The graph is concave down but somewhat linear, increasing, and its rate of growth is slowing down. Call options with higher interest rates yield higher payoffs.



As put price increases, interest rate decreases. The graph is concave up but somewhat linear, decreasing, and its rate of growth is slowing down. Put options with higher interest rates yield higher payoffs.

## 2. Monte Carlo (MC) Simulations

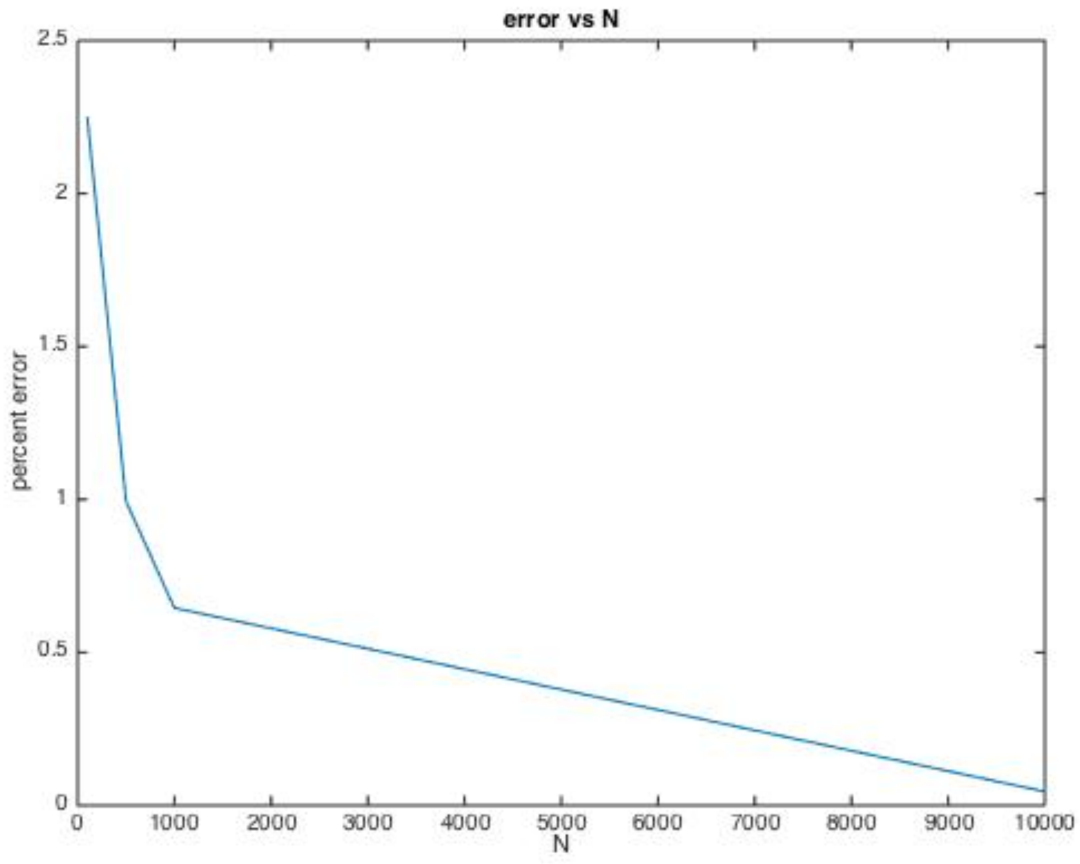
a. Using this program, compute the price of the call option K = 100, T = 1, for different number of simulations N = 100, 500, 1000, 5000, and 10000. Also compute the price of this call option by using the explicit Black-Scholes formula. Compare the obtained prices, in particular, discuss how the number of simulations affects the error.

For this particular simulation with fixed values as noted in problem:

Price of European call using explicit B-S formula

```
bs_call =

   10.8706
```

```
Price of European call using MC Simulation for n = 100
ans =

   35.3101

Price of European call using MC Simulation for n = 500
ans =

    0.0708

Price of European call using MC Simulation for n = 1000
ans =

    3.8542

Price of European call using MC Simulation for n = 10000
ans =

   11.3644
```



We see that as N increases, percent error decreases. This makes sense because there are more random variables to work with when simulating Brownian motion. This allows outliers to have less impact on pricing the data.

b. Consider the option VT = (180ST − ST2 − 7200)+ = max(0, 180ST − ST2 − 7200). Find the price of this option by MC simulations.

```
Price of european call using MC simulations with explicit payoff

ans =

  796.4799
```

**Code attached on next page for both problems.**

```
% Alexander Lerma
% Math 485

function [ call, put ] = black_scholes(st, ttm, k, r, sigma)
% compute option price using black_scholes model
    dp = compute_d(st, ttm, k, r, sigma, 1);
    dm = compute_d(st, ttm, k, r, sigma, -1);
    expon = k * exp(-r * ttm);
    call = st * cdf(dp) - expon * cdf(dm);
    ft = st - expon;
    put = call - ft;
end

function [d] = compute_d (s, ttm, k, r, sigma, factor)
% compute d plus (factor = 1) or d minus (factor = -1)
    computed = (r + factor * (sigma ^ 2) / 2) * ttm;
    d = (log(s / k) + computed) / (sigma * sqrt(ttm));
end

function [ result ] = cdf(d)
% cumulative distribution function
    f = @(x) exp(-x.^2 / 2);
    result = (1 / sqrt(2 * pi)) * integral(f,-Inf, d);
end
```

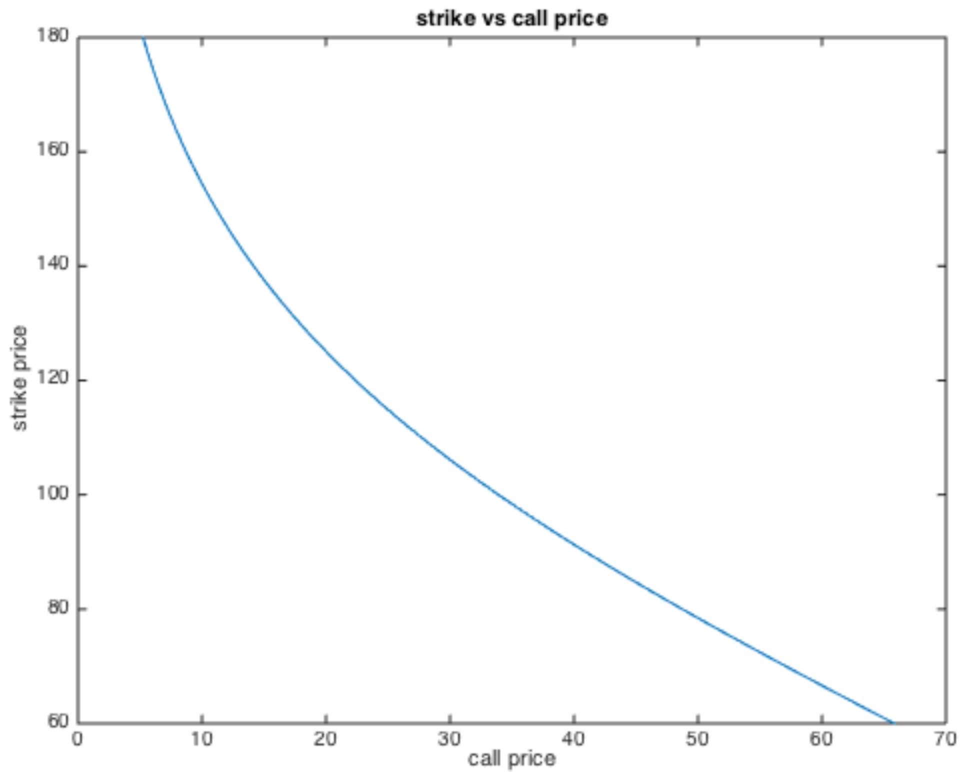
~~*Error using black_scholes (line 6)*~~
~~*Not enough input arguments.*~~

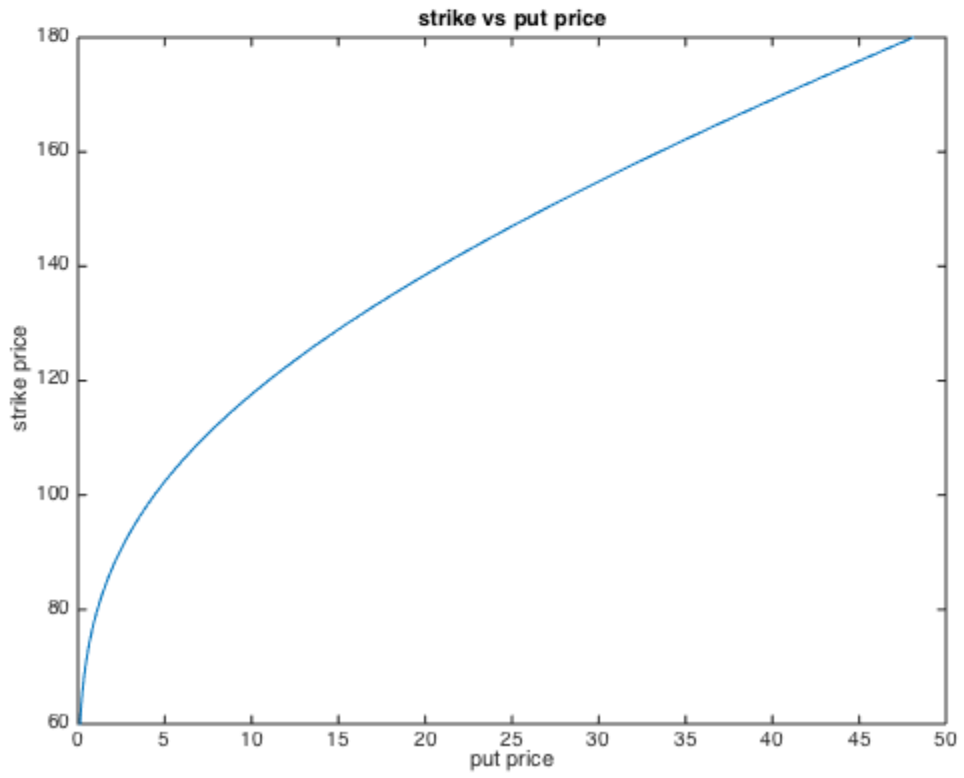
*Published with MATLAB® R2014b*

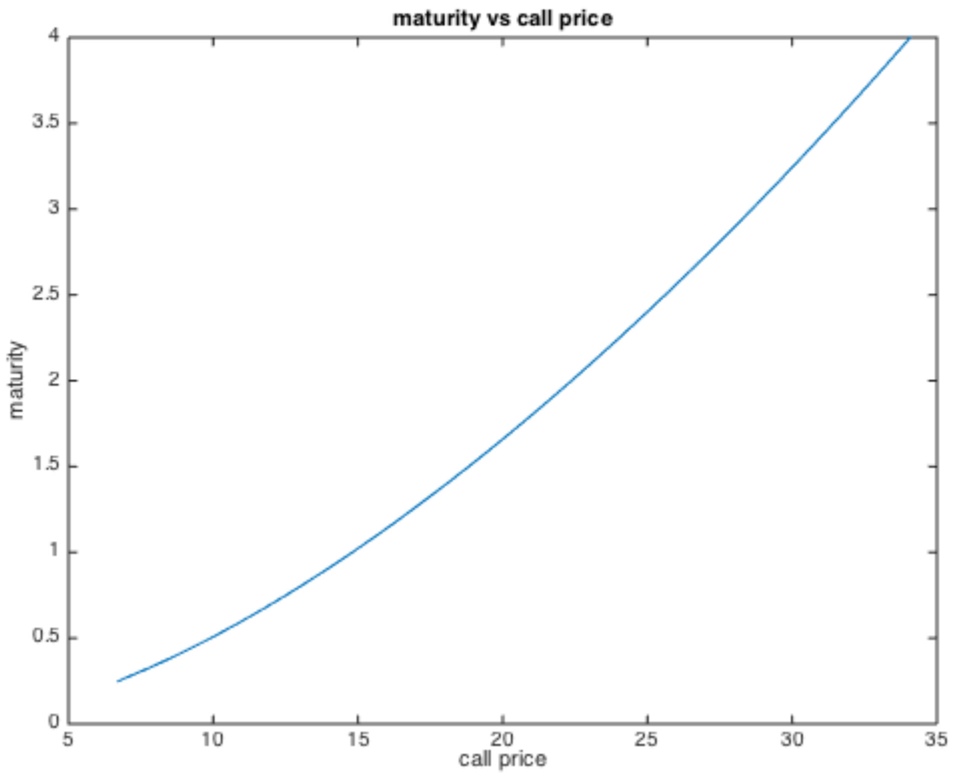
```
% Alexander Lerma
% Math 485

% 1. Write a code (preferable in Matlab) that will compute the price of a
% call option, and a put option, in the Black-Scholes-Merton setup. Attach
% a printed version of the code at the end of the report. Using these code,
% take fixed S0 = 120, and compute the prices for calls and puts as
% follows.

% (a) Fix T = 2, r = 0.05, ? = 0.25, and vary the strike K ? [60,180] with
% step size 2. Plot the graph ?strike vs price? for both sets of options.

s0 = 120;
T = 2;
r = 0.05;
sigma = 0.25;
K = 60:2:180;
calls = NaN([1 length(K)]);
puts = NaN([1 length(K)]);
i = 1;
for k = K
    [calls(i), puts(i)] = black_scholes(s0, T, k, r, sigma);
    i = i + 1;
end

figure('Name', 'strike vs call price')
plot(calls, K)
title('strike vs call price');
xlabel('call price');
ylabel('strike price');


figure('Name', 'strike vs put price')
plot(puts, K)
title('strike vs put price');
xlabel('put price');
ylabel('strike price');

% (b) Fix K = 120, r = 0.05, ? = 0.25, and vary the maturity T ? [0.25, 4]
% with step size 1/12. Plot the graph ?maturity vs price? for both sets of
% options.

K = 120;
r = 0.05;
sigma = 0.25;
T = 0.25:1/12:4;
calls = NaN([1 length(T)]);
puts = NaN([1 length(T)]);
i = 1;
for t = T
    [calls(i), puts(i)] = black_scholes(s0, t, K, r, sigma);
    i = i + 1;
```

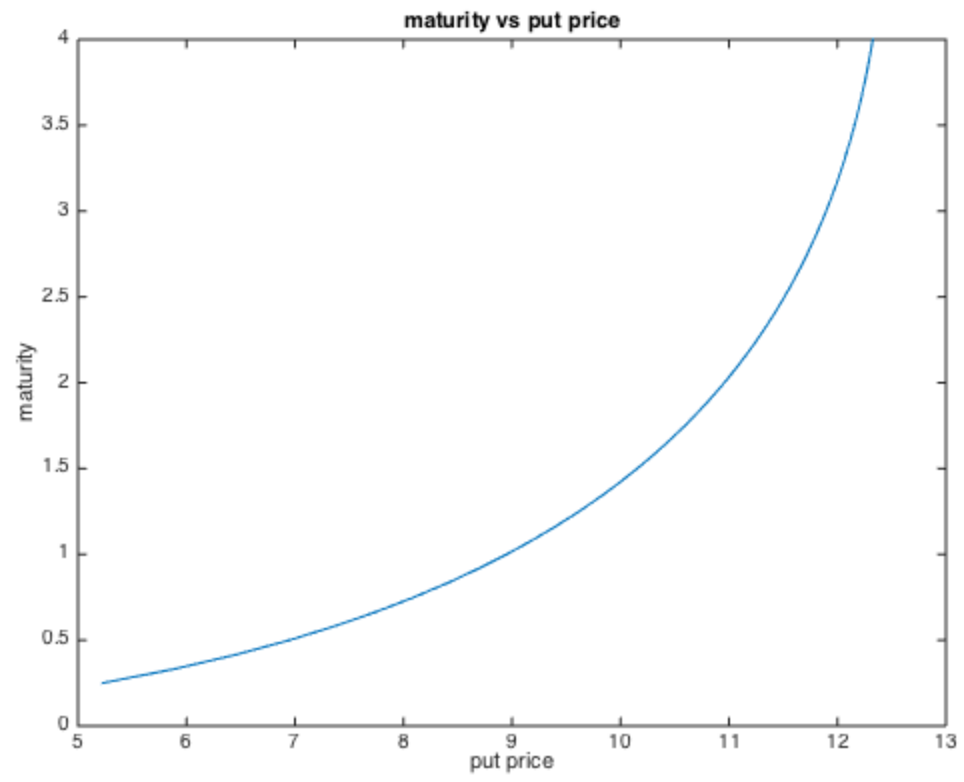
```
end

figure('Name', 'maturity vs call price')
plot(calls, T)
title('maturity vs call price');
xlabel('call price');
ylabel('maturity');

figure('Name', 'maturity vs put price')
plot(puts, T)
title('maturity vs put price');
xlabel('put price');
ylabel('maturity');

% (c) Fix K = 120, T = 2, r = 0.05, and vary the volatility ? ? [0.01, 0.5]
% with step size 0.01. Plot the graph ?volatility vs price? for both sets
% of options.

K = 120;
r = 0.05;
T = 2;
sigma = 0.01:0.01:0.5;
calls = NaN([1 length(sigma)]);
puts = NaN([1 length(sigma)]);
i = 1;
for sgma = sigma
    [calls(i), puts(i)] = black_scholes(s0, T, K, r, sgma);
    i = i + 1;
end

figure('Name', 'volatility vs call price')
plot(calls, sigma)
title('volatility vs call price');
xlabel('call price');
ylabel('volatility');

figure('Name', 'volatility vs put price')
plot(puts, sigma)
title('volatility vs put price');
xlabel('put price');
ylabel('volatility');


% (d) Fix K = 120, T = 2, ? = 0.25, and vary the interest rate r ? [0.01,
% 0.1] with step size 0.005. Plot the graph ?interest rate vs price? for
% both sets of options.

K = 120;
T = 2;
sigma = 0.25;
r = 0.01: 0.005: 0.1;
calls = NaN([1 length(r)]);
puts = NaN([1 length(r)]);
i = 1;
```

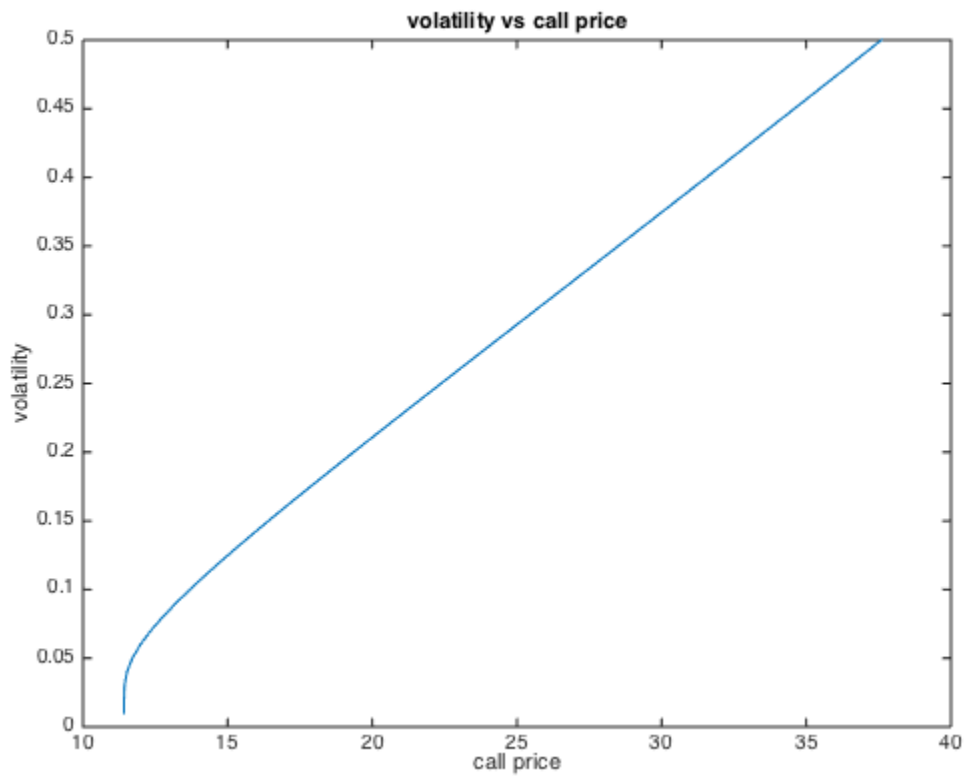
```
for rate = r
    [calls(i), puts(i)] = black_scholes(s0, T, K, rate, sigma);
    i = i + 1;
end

figure('Name', 'interest rate vs call price')
plot(calls, r)
title('interest rate vs call price');
xlabel('call price');
ylabel('iterest rate');


figure('Name', 'interest rate vs put price')
plot(puts, r)
title('interest rate vs put price');
xlabel('put price');
ylabel('iterest rate');
```

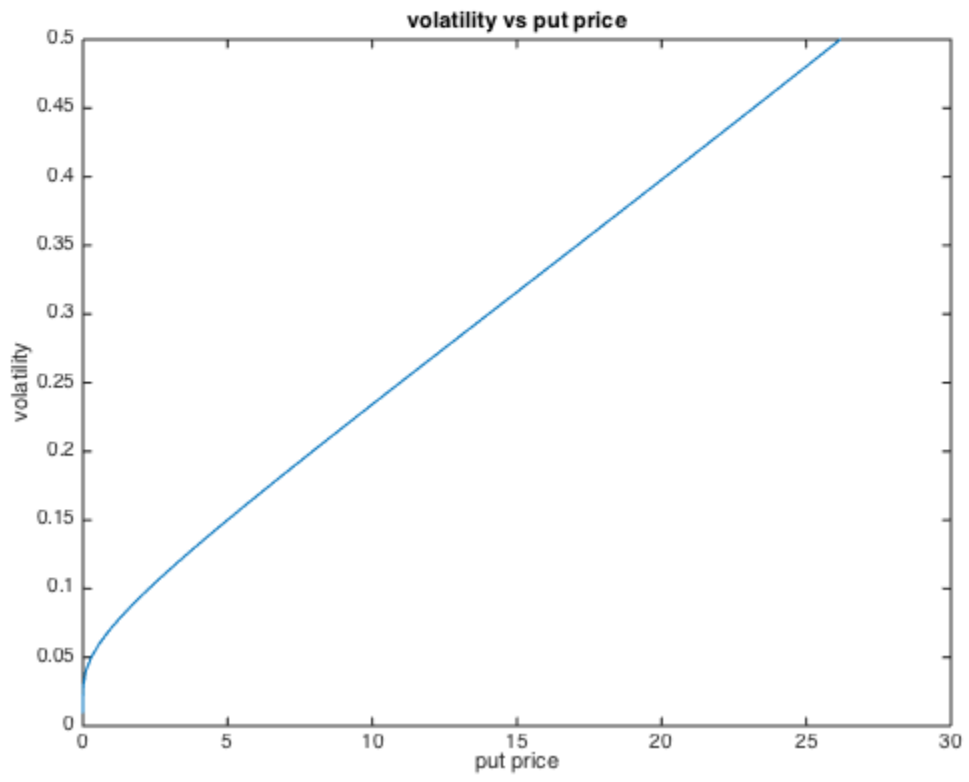


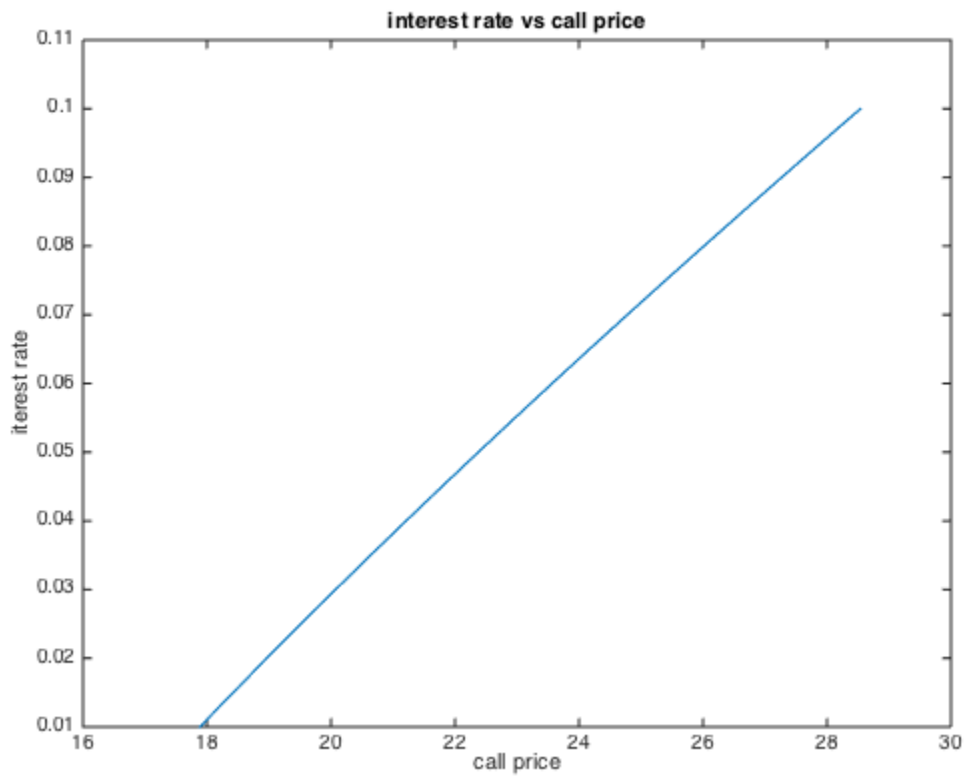

strike vs put price
strike price
put price
180
160
140
120
100
80
60
0
5
10
15
20
25
30
35
40
45
50

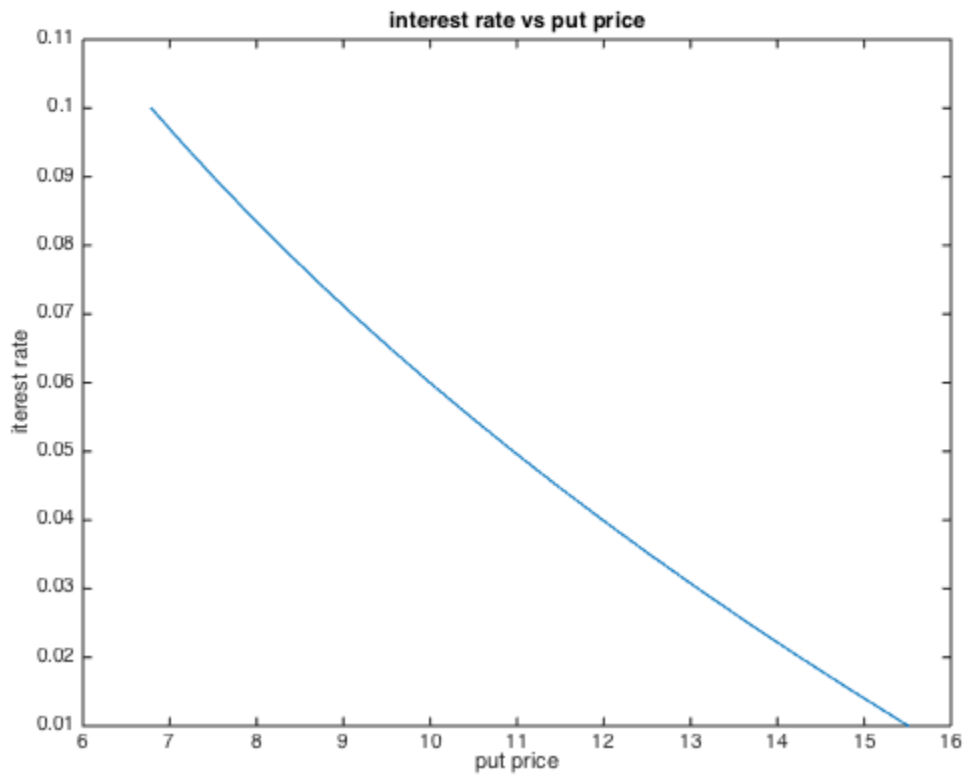

maturity vs call price
maturity
call price
4
3.5
3
2.5
2
1.5
1
0.5
0
5
10
15
20
25
30
35


maturity vs put price
maturity
put price
4
3.5
3
2.5
2
1.5
1
0.5
0
5
6
7
8
9
10
11
12
13


volatility vs call price
volatility
call price
0.5
0.45
0.4
0.35
0.3
0.25
0.2
0.15
0.1
0.05
0
10
15
20
25
30
35
40


volatility vs put price
volatility
put price
0.5
0.45
0.4
0.35
0.3
0.25
0.2
0.15
0.1
0.05
0
5
10
15
20
25
30


interest rate vs call price
iterest rate
call price
0.11
0.1
0.09
0.08
0.07
0.06
0.05
0.04
0.03
0.02
0.01
16
18
20
22
24
26
28
30



*Published with MATLAB® R2014b*

```
% Alexander Lerma
% Math 485

function [ call ] = monte_carlo( st, dt, k, r, sigma, n, payoff )
%SIMULATE_BM Compute n paths of brownian motion to price a call option
% param payoff:a lambda function, default to payoff of a call option if not
% set
    if nargin < 7
        payoff = @(st) max(0, st - k);
    end

    stock_prices = NaN([1, n + 1]); % need room for st, n + 1
    stock_prices(1) = st;

    payoffs = NaN([1, n + 1]);

    i = 2;
    expon = exp((r - (sigma ^ 2) / 2) * dt);
    rvs = randn([1, n]);
    for rv = rvs
        brownian = exp(sigma * sqrt(dt) * rv);
        stock_prices(i) = stock_prices(i-1) * expon * brownian;
        payoffs(i) = payoff(stock_prices(i));
        i = i + 1;
    end

    payoffs(1) = mean(payoffs(1, 2:length(payoffs)));
    call = payoffs(1);
end
```

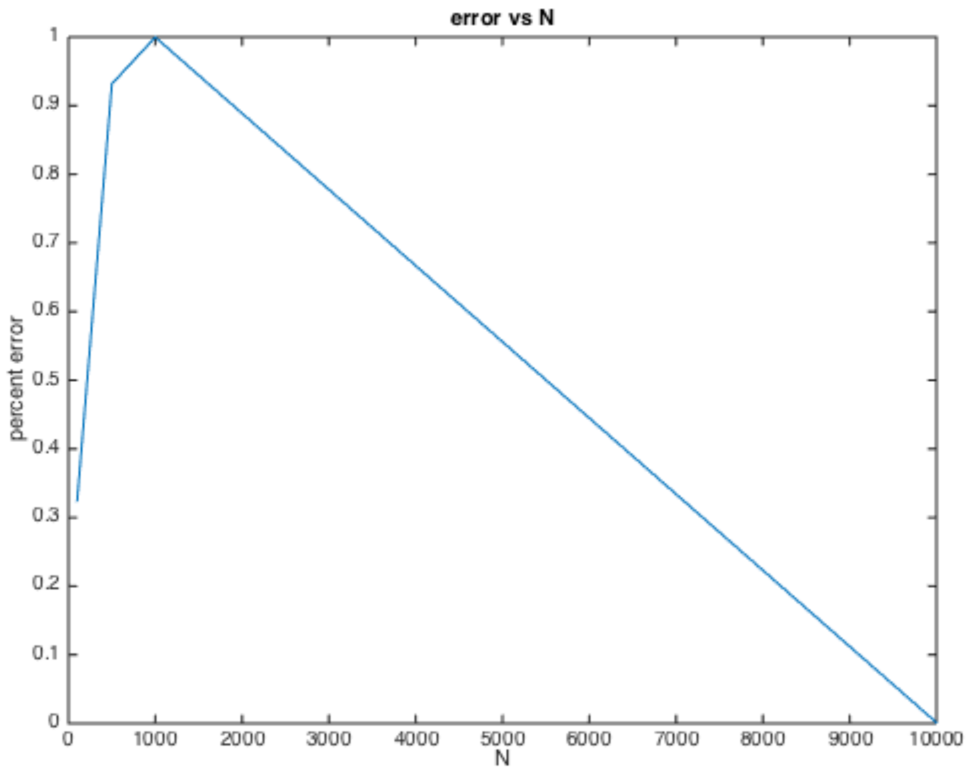
~~*Error using monte_carlo (line 10)*~~
~~*Not enough input arguments.*~~

*Published with MATLAB® R2014b*

```
% Alexander Lerma
% Math 485

% 2. Monte Carlo (MC) Simulations. Implement (in Matlab) the Monte Carlo
% technique for computing prices of a generic European option, under the
% Black-Scholes-Merton model. Attach a printed version of the code to the
% report.

% Throughout this problem fix the model parameters
% r=0.02,?=0.25,and S0 =100.

% Step 1. Consider a given option V , with the
% payoff being an explicit function of ST (for simplicity). Use the scheme
% we discussed in class to simulate one path of the Geometric Brownian
% motion.

% Step 2. Simulate many paths, e.g., N = 10, 000. Generally
% speaking, the order of error is O(1/N ). Use a small enough time step
% size (e.g., ?t = 0.01).

% Step 3. For each path, compute the payoff vn of
% the option V , and store it in a vector.

% Step 4. Compute the average of vn? s: V0 = 1 (v1 + · · · + vN ) . N In view of
% MC method, the number V0 is an approximation of the true value of the
% price V0. The more path you simulate, and smaller time step you take, the
% closer to the true price you get.


% (a) Using this program, compute the
% price of the call option K = 100, T = 1, for different number of
% simulations N = 100, 500, 1000, 5000, and 10000. Also compute the price
% of this call option by using the explicit Black-Scholes formula. Compare
% the obtained prices, in particular, discuss how the number of simulations
% affects the error.


% fixed variables
r = 0.02;
sigma = 0.25;
s0 = 100;


K = 100;
T = 1;
N = [100, 500, 1000, 10000];

disp('Price of European call using explicit B-S formula')
[bs_call] = black_scholes(s0, T, K, r, sigma);
bs_call

errors = NaN([1, length(N)]);
```

```
mc_results = NaN([1, length(N)]);
i = 1;
for n = N
    fprintf('Price of European call using MC Simulation for n = %.0f', n)
    dt = 1 / n;
    mc_results(i) = monte_carlo(s0, dt, K, r, sigma, n);
    errors(i) = abs(mc_results(i) - bs_call) / bs_call;
    mc_results(i)
    i = i + 1;
end

figure('Name', 'error vs N');
plot(N, errors)
title('error vs N');
xlabel('N');
ylabel('percent error');




% (b) Consider the option VT = (180ST ? ST2 ? 7200)+ =
% max(0, 180ST ? ST2 ? 7200). Find the price of this option by MC
% simulations.

N = 10000;
payoff = @(st) max(0, (180 * st) - (st ^ 2) - 7200);

disp('Price of european call using MC simulations with explicit payoff')
monte_carlo(s0, T / N, K, r, sigma, N, payoff)
```

```
Price of European call using explicit B-S formula

bs_call =

   10.8706

Price of European call using MC Simulation for n = 100
ans =

    7.3424

Price of European call using MC Simulation for n = 500
ans =

    0.7415

Price of European call using MC Simulation for n = 1000
ans =

    0.0033

Price of European call using MC Simulation for n = 10000
ans =
   10.8589

Price of european call using MC simulations with explicit payoff

ans =

  840.2041
```



*Published with MATLAB® R2014b*