****

**GEORGIA INSTITUTE OF TECHNOLOGY**

**Title: ISyE6785 Interim Project 2**

**Author(s): Weifeng Lyu**

**Instructor: Dr. Shijie Deng**

**TABLE OF CONTENTS**

1. **Introduction**
   1. Problem . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 3
   2. Computer Configuration . . . . . . . . . . . . . . . . . . . . . . . .. . . . 3
   3. Workflow Diagram . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 4
2. **Technology Review**
   1. Algorithm Review . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 5
   2. Algorithm Parameter . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 9
3. **Project Architecture**
   1. Python Implementation . . . . . . . . . . . . . . . . . . . . . . . . . . . . 9
   2. Result of Sample Data and Discussion. . . . . . . . . . . . . . . .22
4. **Improvement** . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 34
5. **Conclusion** . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 34
6. **Reference** . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 35

**Appendices**

1. **Execute Python Code** . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 35

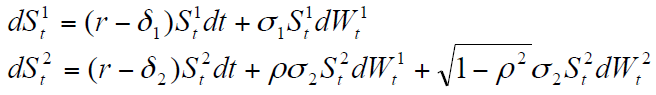
**1. Introduction**

1.1 Problem

ISyE 6785: Mini-project 2 (Due on 7/12/2018)

Note: For all the computations, please report the configuration of your computer (e.g. CPU type and speed, size of RAM) and the computing CPU time in seconds.

1. A spread call option on two assets S1, S2 with strike price K pays off max (S1 - S2 - K, 0); a spread put option on two assets S1, S2 with strike price K pays off max (K – (S1 - S2), 0). Suppose the asset price processes are given by the following correlated Geometric Brownian motions.



where r = 4.5%, δ1 = 2%, σ1 = 20%, δ2 = 0.5%, σ2 = 25%, ρ = 0.3. The current prices are S1 = $100, S2 =$95.

a. Implement a simulation approach to simulate 100 price paths of (S1, S2) from time 0 to T = 0.5 years.

b. Implement the Longstaff and Schwartz (RFS, 2001) algorithm to price a

standard American-style put option on S1 with strike price K = 90 and maturity time T = 0.5 years.

c. (Bonus question, optional) Price an American-style spread call option with strike price K = 15 and the maturity time 0.5 years.

1.2 Computer Configuration

Manufacturer: Dell

Model: Inspiron 7559 Signature Edition

Processor: Intel® Core™ i7-6700HQ CPU @ 2.60GHz 2.60GHz

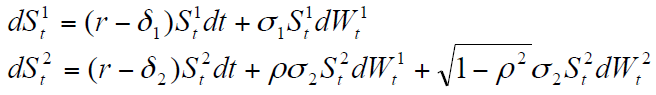
Installed Memory (RAM): 8.00GB (7.88 usable)

System Type: 64-bit Operating System, x64-based processor

**2. Technology Review**

2.1 Algorithm Review

(a) Geometric Brownian motions formula is given by the problem



(b) Antithetic variates: Equipped with a basis for evaluating potential efficiency improvements, we can now consider specific variance reduction techniques.

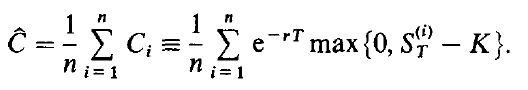
Consider the problem of computing the Black-Scholes price of a European

call option on a no-dividend stock.

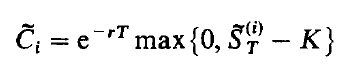


Based on n replications, an unbiased estimator of the price of an

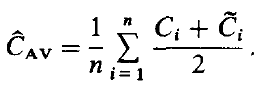
option with strike K is given by



the method of antithetic variates4 is based on the observation that if Zi has a standard normal distribution, then so does -Zi, The price obtained from (2) with Zi replaced by -Zi is thus a valid sample from the terminal stock price distribution. Similarly, each



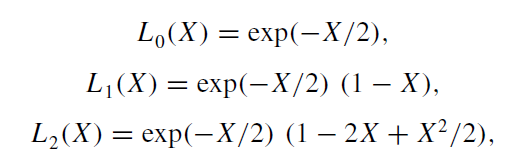
is an unbiased estimator of the option price, as is, therefore,



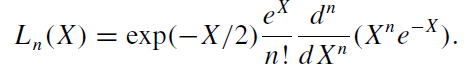
(c) The LSM approach uses least squares to approximate the conditional expectation function at t. We work backwards since the path of cash flows C(w,s;t,T) generated by the option is defined recursively.



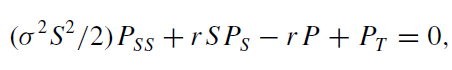
As a set of basis functions, we use a constant and the first three Laguerre polynomials as given in the followings:



Thus we regress discounted realized cash flows on a constant and three nonlinear functions of the stock price. Since we use linear regression to estimate the conditional expectation function, it is straightforward to add additional basis functions as explanatory variables in the regression as needed. Using more than three basis functions are sufficient to obtain effective convergence of the algorithm in this example.



The partial differential equation satisfied by the price P(S,t) is



2.2 Algorithm Parameter

1. Black-Scholes Model

S0 : int or float, initial asset value

K : int or float, strike

T : int or float, time to expiration as a fraction of one year

r : int or float, continuously compounded risk free rate, annualized

sigma : int or float, continuously compounded standard deviation of returns

kind : str, {'call', 'put'}, default 'call', type of option

2. Monte Carlo Methods

S0 : int or float, initial asset value

K : int or float, strike

T : int or float, time to expiration as a fraction of one year

M : int, grid or granularity for time (in number of total points)

r : int or float, continuously compounded risk free rate, annualized

i : int, number of simulations

sigma : int or float, continuously compounded standard deviation of returns

delta : int or float

rho: int or float

kind : str, {'call', 'put'}, default 'call', type of option

seed: int, random seed to generate simulations

**3. Project Architecture**

3.1 Python Implementation

**import** numpy **as** np  
**import** pandas **as** pd  
**from** matplotlib **import** pyplot **as** plt  
**import** scipy.stats  
**import** time  
**import** math  
  
**class** Option(object):  
 *"""Compute European option value, greeks, and implied volatility.  
  
 Parameters  
 ==========  
 S0 : int or float  
 initial asset value  
 K : int or float  
 strike  
 T : int or float  
 time to expiration as a fraction of one year  
 M : int  
 grid or granularity for time (in number of total points)  
 r : int or float  
 continuously compounded risk free rate, annualized  
 i : int  
 number of simulations  
 sigma : int or float  
 continuously compounded standard deviation of returns  
 delta : int or float  
 rho: int or float  
 kind : str, {'call', 'put'}, default 'call'  
 type of option  
  
 Resources  
 =========  
 http://www.thomasho.com/mainpages/?download=&act=model&file=256  
 """* **def** \_\_init\_\_(self, S0, K, T, M, r, delta, sigma, i, kind=**'call'**):  
 **if** kind.istitle():  
 kind = kind.lower()  
 **if** kind **not in** [**'call'**, **'put'**]:  
 **raise** ValueError(**'Option type must be \'call\' or \'put\''**)  
  
 self.S0 = S0  
 self.K = K  
 self.T = T  
 self.M = int(M)  
 self.r = r  
 self.delta = delta  
 self.sigma = sigma  
 self.i = int(i)  
 self.kind = kind  
 self.time\_unit = self.T / float(self.M)  
 self.discount = np.exp(-self.r \* self.time\_unit)  
  
 self.d1 = ((np.log(self.S0 / self.K)  
 + (self.r + 0.5 \* self.sigma \*\* 2) \* self.T)  
 / (self.sigma \* np.sqrt(self.T)))  
 self.d2 = ((np.log(self.S0 / self.K)  
 + (self.r - 0.5 \* self.sigma \*\* 2) \* self.T)  
 / (self.sigma \* np.sqrt(self.T)))  
  
 *# Several greeks use negated terms dependent on option type  
 # For example, delta of call is N(d1) and delta put is N(d1) - 1* self.sub = {**'call'** : [0, 1, -1], **'put'** : [-1, -1, 1]}  
  
 **def** value(self):  
 *"""Compute option value."""* **return** (self.sub[self.kind][1] \* self.S0  
 \* scipy.stats.norm.cdf(self.sub[self.kind][1] \* self.d1)  
 + self.sub[self.kind][2] \* self.K \* np.exp(-self.r \* self.T)  
 \* scipy.stats.norm.cdf(self.sub[self.kind][1] \* self.d2))  
  
 **def** AmericanPutPrice(self, seed):  
 *""" Returns Monte Carlo price matrix rows: time columns: price-path simulation """* np.random.seed(seed)  
 path = np.zeros((self.M + 1, self.i), dtype=np.float64)  
 path[0] = self.S0  
 **for** t **in** range(1, self.M + 1):  
 rand = np.random.standard\_normal(int(self.i / 2))  
 rand = np.concatenate((rand, -rand))  
 path[t] = (path[t - 1] \* np.exp((self.r - self.delta) \* self.time\_unit + self.sigma \* np.sqrt(self.time\_unit) \* rand))  
  
 **""" Returns the inner-value of American Option """  
 if** self.kind == **'call'**:  
 payoff = np.maximum(path - self.K, np.zeros((self.M + 1, self.i), dtype=np.float64))  
 **else**:  
 payoff = np.maximum(self.K - path, np.zeros((self.M + 1, self.i), dtype=np.float64))  
  
 value = np.zeros\_like(payoff)  
 value[-1] = payoff[-1]  
 **for** t **in** range(self.M - 1, 0, -1):  
 regression = np.polyfit(path[t], value[t + 1] \* self.discount, 5)  
 continuation\_value = np.polyval(regression, path[t])  
 value[t] = np.where(payoff[t] > continuation\_value, payoff[t], value[t + 1] \* self.discount)  
  
 **return** np.sum(value[1] \* self.discount) / float(self.i)  
  
*# This is a function simulating the price path for a Geometric Brownian Motion price model***def** gen\_paths(S0\_1, S0\_2, r, delta\_1, delta\_2, sigma\_1, sigma\_2, rho, T, M, I):  
 dt = float(T) / M  
 path\_1 = np.zeros((M + 1, I), np.float64)  
 path\_2 = np.zeros((M + 1, I), np.float64)  
 path\_1[0] = S0\_1  
 path\_2[0] = S0\_2  
 **for** t **in** range(1, M + 1):  
 rand\_1 = np.random.standard\_normal(I)  
 rand\_2 = np.random.standard\_normal(I)  
 path\_1[t] = path\_1[t - 1] \* np.exp((r - delta\_1) \* dt +  
 sigma\_1 \* np.sqrt(dt) \* rand\_1)  
 path\_2[t] = path\_2[t - 1] \* np.exp((r - delta\_2) \* dt +  
 rho \* sigma\_2 \* np.sqrt(dt) \* rand\_1 +  
 np.sqrt(1-rho\*\*2) \* sigma\_2 \* np.sqrt(dt) \* rand\_2)  
 **return** [path\_1,path\_2]  
  
**def** gen\_paths\_antithetic(S0\_1, S0\_2, r, delta\_1, delta\_2, sigma\_1, sigma\_2, rho, T, M, I):  
 dt = float(T) / M  
 path\_1 = np.zeros((2\*M + 1, I), np.float64)  
 path\_2 = np.zeros((2\*M + 1, I), np.float64)  
 path\_1[0] = S0\_1  
 path\_2[0] = S0\_2  
 **for** t **in** range(1, M + 1):  
 rand\_1 = np.random.standard\_normal(I)  
 rand\_2 = np.random.standard\_normal(I)  
 rand\_anti\_1 = -1.0 \* rand\_1 *# antithetic variates* rand\_anti\_2 = -1.0 \* rand\_2 *# antithetic variates* path\_1[2 \* t] = path\_1[2 \* (t - 1)] \* np.exp((r - delta\_1) \* dt +  
 sigma\_1 \* np.sqrt(dt) \* rand\_1)  
 path\_1[2 \* t - 1] = path\_1[max(2 \* t - 3, 0)] \* np.exp((r - delta\_1) \* dt +  
 sigma\_1 \* np.sqrt(dt) \* rand\_anti\_1)  
 path\_2[2\*t] = path\_2[2\*(t - 1)] \* np.exp((r - delta\_2) \* dt +  
 rho \* sigma\_2 \* np.sqrt(dt) \* rand\_1 +  
 np.sqrt(1 - rho \*\* 2) \* sigma\_2 \* np.sqrt(dt) \* rand\_2)  
 path\_2[2\*t-1] = path\_2[max(2\*t - 3, 0)] \* np.exp((r - delta\_2) \* dt +  
 rho \* sigma\_2 \* np.sqrt(dt) \* rand\_anti\_1 +  
 np.sqrt(1 - rho \*\* 2) \* sigma\_2 \* np.sqrt(dt) \* rand\_anti\_2)  
 **return** [path\_1,path\_2]  
  
**def** hist\_comp(dist1, dist2, lgnd, bin\_num):  
 hist\_start = min(min(dist1), min(dist2))  
 hist\_end = max(max(dist1), max(dist2))  
 bin\_vec = np.linspace(hist\_start, hist\_end, bin\_num)  
 plt.hist([dist1, dist2], color=[**'r'**,**'g'**], label=[lgnd[0],lgnd[1]], alpha=0.8, bins=bin\_vec)  
 plt.legend(loc=**'upper right'**)  
 plt.show()  
  
*#1*S0\_1 = 100.0  
S0\_2 = 95.0  
K = 90.0  
r = 0.045  
delta\_1 = 0.02  
sigma\_1 = 0.2  
delta\_2 = 0.005  
sigma\_2 = 0.25  
rho = 0.3  
T = 0.5  
M = 200 *#252*i = 100  
discount\_factor = np.exp(-r \* T)  
  
seed = 84  
*## Closed-form option price*start\_time = time.time()  
call\_option\_1 = Option(S0\_1, K, T, M, r, delta\_1, sigma\_1, i, **'call'**)  
call\_option\_2 = Option(S0\_2, K, T, M, r, delta\_2, sigma\_2, i, **'call'**)  
print(**'B-S price for Asset 1: %f, time used: %f.'** % (call\_option\_1.value(), time.time()-start\_time))  
print(**'B-S price for Asset 2: %f, time used: %f.'** % (call\_option\_2.value(), time.time()-start\_time))  
  
*## Set seed for a random number generator*start\_time = time.time()  
np.random.seed(seed)  
[path1,path2] = gen\_paths(S0\_1, S0\_2, r, delta\_1, delta\_2, sigma\_1, sigma\_2, rho, T, M, i)  
duration = time.time()-start\_time  
  
*## Plot all sample paths*pd.DataFrame(path1).plot()  
plt.xlabel(**'time'**)  
plt.ylabel(**'price'**)  
plt.title(**"Simulate 100 price paths of S1"**)  
pd.DataFrame(path2).plot()  
plt.xlabel(**'time'**)  
plt.ylabel(**'price'**)  
plt.title(**"Simulate 100 price paths of S2"**)  
plt.show()  
  
  
*# Compute the value of a Call option*CallPayoffAverage\_1 = np.average(np.maximum(0, path1[-1] - K))  
CallPayoff\_1 = discount\_factor \* CallPayoffAverage\_1  
print(**'MC estimator for Asset 1: %f, time used: %f.'** % (CallPayoff\_1, duration))  
CallPayoffAverage\_2 = np.average(np.maximum(0, path2[-1] - K))  
CallPayoff\_2 = discount\_factor \* CallPayoffAverage\_2  
print(**'MC estimator for Asset 2: %f, time used: %f.'** % (CallPayoff\_2, duration))  
  
*## Antithetic variate estimator*start\_time = time.time()  
[path\_1\_anti,path\_2\_anti] = gen\_paths\_antithetic(S0\_1, S0\_2, r, delta\_1, delta\_2, sigma\_1, sigma\_2, rho, T, M, i)  
CallPayoffAverage\_1 = np.average(np.maximum(0, path\_1\_anti[-1] - K))  
CallPayoffAverage\_tilda\_1 = np.average(np.maximum(0, path\_1\_anti[-2] - K))  
CallPayoff\_anti\_1 = discount\_factor \* (CallPayoffAverage\_1+CallPayoffAverage\_tilda\_1)/2.0  
CallPayoffAverage\_2 = np.average(np.maximum(0, path\_2\_anti[-1] - K))  
CallPayoffAverage\_tilda\_2 = np.average(np.maximum(0, path\_2\_anti[-2] - K))  
CallPayoff\_anti\_2 = discount\_factor \* (CallPayoffAverage\_2+CallPayoffAverage\_tilda\_2)/2.0  
mcav\_time = time.time() - start\_time  
print(**'Antithetic variate estimator for path1: %f, Antithetic variate time used: %f.'** % (CallPayoff\_anti\_1,mcav\_time))  
print(**'Antithetic variate estimator for path2: %f, Antithetic variate time used: %f.'** % (CallPayoff\_anti\_2,mcav\_time))  
  
M = 100 *# number of Monte Carlo estimators*MC\_vec\_1 = []  
MCAV\_vec\_1 = []  
MC\_vec\_2 = []  
MCAV\_vec\_2 = []  
best\_j1 = 0  
best\_j2 = 0  
best\_diff = math.inf  
  
*#test different seed for Monte Carlo estimators***for** j **in** range(M):  
 np.random.seed(j+1)  
 [path1,path2] = gen\_paths(S0\_1, S0\_2, r, delta\_1, delta\_2, sigma\_1, sigma\_2, rho, T, M, i)  
 CallPayoffAverage\_1 = np.average(np.maximum(0, path1[-1] - K))  
 CallPayoff\_1 = discount\_factor \* CallPayoffAverage\_1  
 CallPayoffAverage\_2 = np.average(np.maximum(0, path2[-1] - K))  
 CallPayoff\_2 = discount\_factor \* CallPayoffAverage\_2  
 MC\_vec\_1.append(CallPayoff\_1)  
 MC\_vec\_2.append(CallPayoff\_2)  
 [path\_1\_anti, path\_2\_anti] = gen\_paths\_antithetic(S0\_1, S0\_2, r, delta\_1, delta\_2, sigma\_1, sigma\_2, rho, T, M, i)  
 CallPayoffAverage\_1 = np.average(np.maximum(0, path\_1\_anti[-1] - K))  
 CallPayoffAverage\_tilda\_1 = np.average(np.maximum(0, path\_1\_anti[-2] - K))  
 CallPayoff\_anti\_1 = discount\_factor \* (CallPayoffAverage\_1 + CallPayoffAverage\_tilda\_1) / 2.0  
 CallPayoffAverage\_2 = np.average(np.maximum(0, path\_1\_anti[-1] - K))  
 CallPayoffAverage\_tilda\_2 = np.average(np.maximum(0, path\_2\_anti[-2] - K))  
 CallPayoff\_anti\_2 = discount\_factor \* (CallPayoffAverage\_2 + CallPayoffAverage\_tilda\_2) / 2.0  
 MCAV\_vec\_1.append(CallPayoff\_anti\_1)  
 MCAV\_vec\_2.append(CallPayoff\_anti\_2)  
 diff1 = abs(CallPayoff\_anti\_1 - call\_option\_1.value())  
 diff2 = abs(CallPayoff\_anti\_2 - call\_option\_2.value())  
 **if** diff1 + diff2 < best\_diff:  
 best\_diff = diff1 + diff2  
 best\_j = j  
  
print(**'Best Seed for Asset S1 and S2:'**, best\_j)  
  
MC\_mean\_1 = np.average(MC\_vec\_1)  
MC\_std\_1 = np.sqrt(np.var(MC\_vec\_1))  
MC\_mean\_2 = np.average(MC\_vec\_2)  
MC\_std\_2 = np.sqrt(np.var(MC\_vec\_2))  
  
print(**'Naive MC estimator for Asset 1 mean: %f, standard dev: %f.'** % (MC\_mean\_1, MC\_std\_1))  
print(**'Antithetic Variates MC estimator for Asset 1 mean: %f, standard dev: %f.'** % (np.average(MCAV\_vec\_1), np.sqrt(np.var(MCAV\_vec\_1))))  
  
print(**'Naive MC estimator for Asset 2 mean: %f, standard dev: %f.'** % (MC\_mean\_2, MC\_std\_2))  
print(**'Antithetic Variates MC estimator for Asset 2 mean: %f, standard dev: %f.'** % (np.average(MCAV\_vec\_2), np.sqrt(np.var(MCAV\_vec\_2))))  
  
*### Plot the histogram of the Monte Carlo estimators and the Antithetic Variate estimators*hist\_start = min(min(MC\_vec\_1), min(MCAV\_vec\_1))  
hist\_end = max(max(MC\_vec\_1), max(MCAV\_vec\_1))  
bin\_num = 40  
bin\_vec = np.linspace(hist\_start, hist\_end, bin\_num)  
plt.hist([MC\_vec\_1, MCAV\_vec\_1], color=[**'r'**,**'g'**], label=[**'MC'**,**'MCAV'**], alpha=0.8, bins=bin\_vec)  
plt.xlabel(**'price'**)  
plt.ylabel(**'number of sample'**)  
plt.legend(loc=**'upper right'**)  
plt.title(**"Price Sample Distribution of Naive MC estimator Vs. Antithetic variate MC estimator of S1"**)  
plt.show()  
hist\_start = min(min(MC\_vec\_2), min(MCAV\_vec\_2))  
hist\_end = max(max(MC\_vec\_2), max(MCAV\_vec\_2))  
bin\_num = 40  
bin\_vec = np.linspace(hist\_start, hist\_end, bin\_num)  
plt.hist([MC\_vec\_2, MCAV\_vec\_2], color=[**'r'**,**'g'**], label=[**'MC'**,**'MCAV'**], alpha=0.8, bins=bin\_vec)  
plt.xlabel(**'price'**)  
plt.ylabel(**'number of sample'**)  
plt.legend(loc=**'upper right'**)  
plt.title(**"Price Sample Distribution of Naive MC estimator Vs. Antithetic variate MC estimator of S2"**)  
plt.show()  
  
*#2 Price American Put Option*put\_option\_1 = Option(S0\_1, K, T, M, r, delta\_1, sigma\_1, i, **'put'**)  
mc\_price = []  
bs\_price = []  
best\_diff = math.inf  
**for** j **in** range(M):  
 mc\_put\_option\_1 = Option(S0\_1, K, T, M, r, delta\_1, sigma\_1, i, **'put'**).AmericanPutPrice(j)  
 mc\_price.append(mc\_put\_option\_1)  
 bs\_price.append(put\_option\_1.value())  
 diff = abs(mc\_put\_option\_1 - put\_option\_1.value())  
 **if** diff < best\_diff:  
 best\_diff = diff  
 best\_seed = j  
  
plt.plot(mc\_price,label=**'MC'**)  
plt.plot(bs\_price,label=**'BS'**)  
plt.xlabel(**'seed'**)  
plt.ylabel(**'price'**)  
plt.legend(loc=**'upper right'**)  
plt.title(**"Black Scholes Price Vs. Monte Carlo Price Simulation in Different Seed"**)  
plt.show()  
  
print(**'Best Seed of Put Option for Asset S1:'**, best\_seed)  
seed = best\_seed  
print(**'American Put Option Price (MC) for S1: %f'** % Option(S0\_1, K, T, M, r, delta\_1, sigma\_1, i, **'put'**).AmericanPutPrice(seed))  
print(**'American Put Option Price (BS) for S1: %f'** % put\_option\_1.value())  
  
*#3 Price American Call Option*K = 15  
call\_option\_1 = Option(S0\_1, K, T, M, r, delta\_1, sigma\_1, i, **'call'**)  
mc\_price = []  
bs\_price = []  
best\_diff = math.inf  
**for** j **in** range(M):  
 mc\_call\_option\_1 = Option(S0\_1, K, T, M, r, delta\_1, sigma\_1, i, **'call'**).AmericanPutPrice(j)  
 mc\_price.append(mc\_call\_option\_1)  
 bs\_price.append(call\_option\_1.value())  
 diff = abs(mc\_call\_option\_1 - call\_option\_1.value())  
 **if** diff < best\_diff:  
 best\_diff = diff  
 best\_seed = j  
  
plt.plot(mc\_price,label=**'MC'**)  
plt.plot(bs\_price,label=**'BS'**)  
plt.xlabel(**'seed'**)  
plt.ylabel(**'price'**)  
plt.legend(loc=**'upper right'**)  
plt.title(**"Black Scholes Price Vs. Monte Carlo Price Simulation in Different Seed"**)  
plt.show()  
  
print(**'Best Seed of Call Option for Asset S1:'**, best\_seed)  
seed = best\_seed  
print (**'American Call Option Price (MC): '**, Option(S0\_1, K, T, M, r, delta\_1, sigma\_1, i, **'call'**).AmericanPutPrice(seed))  
print (**'American Call Option Price (BS): '**, Option(S0\_1, K, T, M, r, delta\_1, sigma\_1, i, **'call'**).value())

3.2 Result of Sample Data and Discussion

a. Implement a simulation approach to simulate 100 price paths of (S1, S2) from time 0 to T = 0.5 years.

B-S price for Asset 1: 13.322705, time used: 0.000000.

B-S price for Asset 2: 10.538943, time used: 0.000000.

MC estimator for Asset 1: 12.967337, time used: 0.005231.

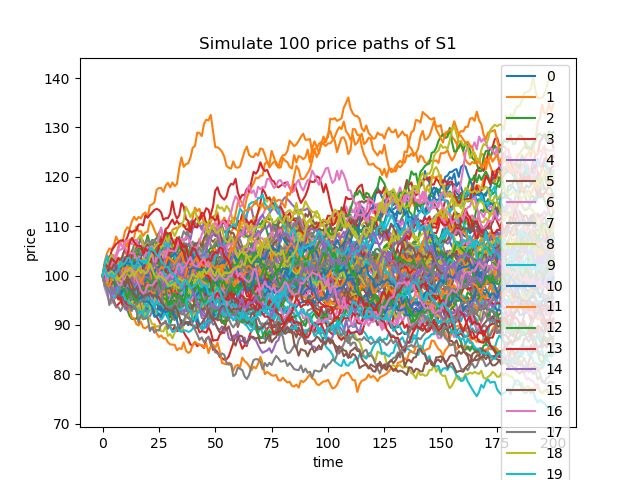
MC estimator for Asset 2: 9.597588, time used: 0.005231.

Antithetic variate estimator for path1: 13.028943, Antithetic variate time used: 0.013088.

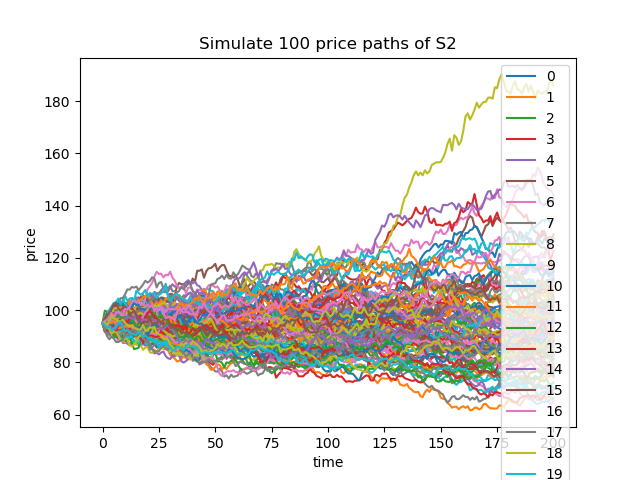
Antithetic variate estimator for path2: 10.863420, Antithetic variate time used: 0.013088.

Best Seed for Asset S1 and S2: 84

price paths of S1:



price paths of S2:



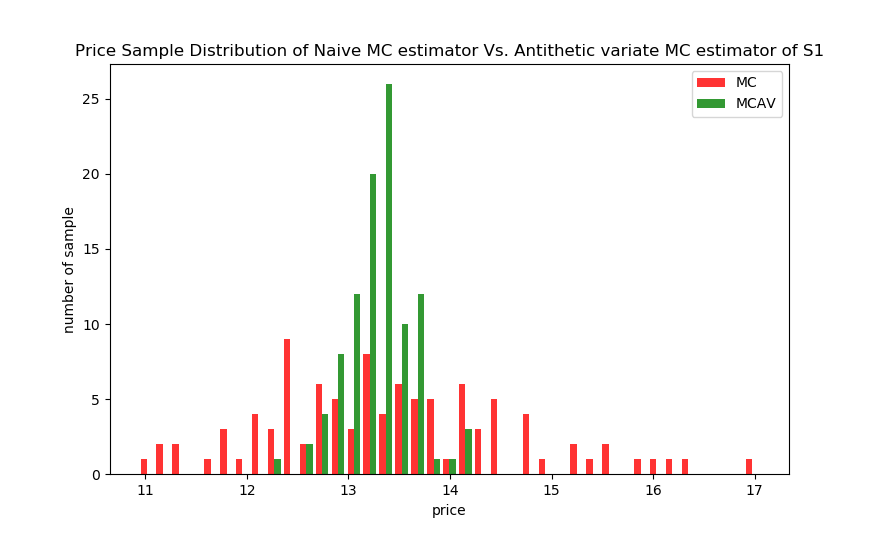
Naive MC estimator for Asset 1 mean: 13.429590, standard dev: 1.221526.

Antithetic Variates MC estimator for Asset 1 mean: 13.295097, standard dev: 0.330655.

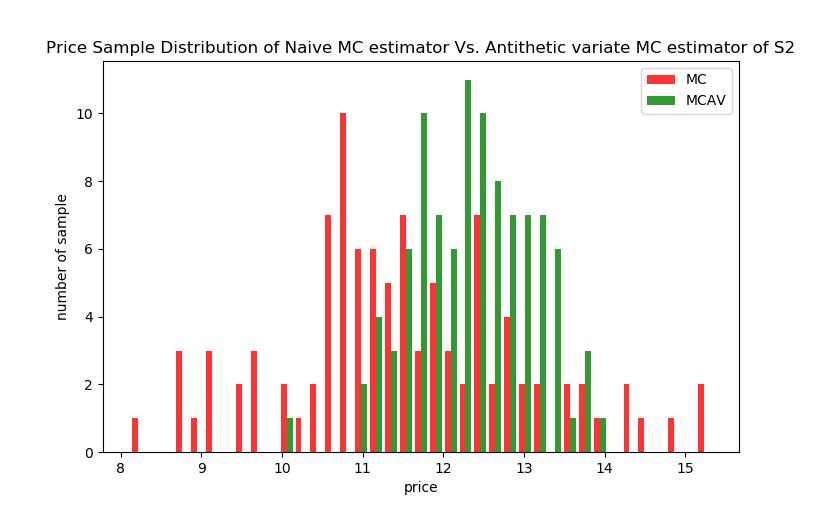
Naive MC estimator for Asset 2 mean: 11.517873, standard dev: 1.455184.

Antithetic Variates MC estimator for Asset 2 mean: 12.360179, standard dev: 0.742329.

Price Sample Distribution of Naïve Monte Carlo estimator Vs. Antithetic variate Monte Carlo estimator of S1:



Price Sample Distribution of Naïve Monte Carlo estimator Vs. Antithetic variate Monte Carlo estimator of S2:



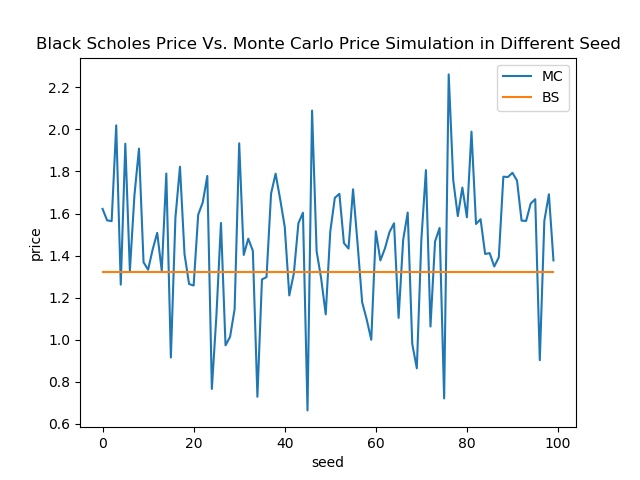
b. Implement the Longstaff and Schwartz (RFS, 2001) algorithm to price a

standard American-style put option on S1 with strike price K = 90 and maturity time T = 0.5 years.

Best Seed of Put Option for Asset S1: 42

American Put Option Price (MC) for S1: 1.314627

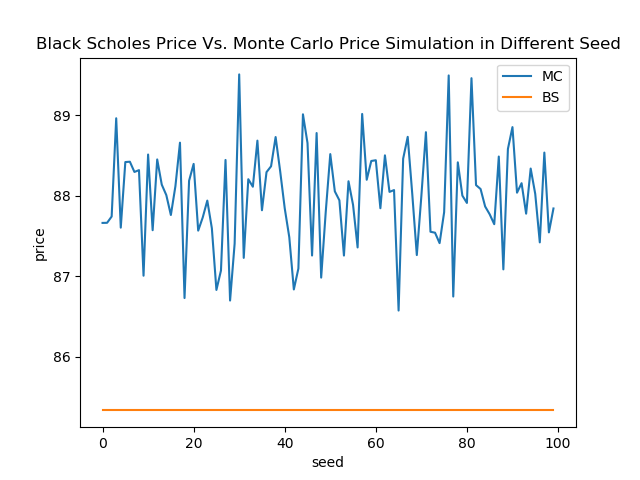
American Put Option Price (BS) for S1: 1.320316



c. (Bonus question, optional) Price an American-style spread call option with strike price K = 15 and the maturity time 0.5 years.

Best Seed of Call Option for Asset S1: 65

American Call Option Price (MC): 86.57488792405864

American Call Option Price (BS): 85.33373144209996

**4. Improvement**

In this project, the algorithm used for calculating the call option in Trinomial Tree should be optimized. Somehow, when the time step becomes larger, the accuracy does not increase. Instead, it drops. This error could be due to non-linearity error. Currently, its takes too much resource and CPU time to compute in large number of time steps. Delta and Gamma results still need more improvement to obtain closer experiment data to the paper. Noticing the result of Black-Scholes Analytic Value is slightly different than theoretical values, it needs more improvement to close the numeric gap and reach greater accuracy. We rely on the paper’s formula [1] to compute Black-Scholes Analytic value, the computation algorithm might be incorrectly introducing or missing some important parameters. However, the computer configuration, algorithm defects could contribute to those imperfect results.

**5. Conclusions**

In this project, I learned how to implement Binomial Tree Model, Trinomial Tree Model, Adaptive Mesh Method in computing European Call Option based on different parameters: number of time steps, barrier option, current price level, mesh level. All the implementations are based on solid understandings of European call option financial theorem.

Computing methods are derived from risk-neutral probability setup and parameters such as current pricing, strike pricing, alpha, sigma, time length, risk-free interest rate, dividend rate.

The Trinomial Tree down-and-in, down-and-out algorithms are relatively expensive when it constructs large numbers of periods, requiring the complexity of n^2 to generate paths to reach maturity prices. This project implementation truncates time step from requirements and perform reasonable computational results as the paper states and Black-Scholes model. The computing methods share common characteristics between Binomial Tree and Trinomial Tree model.

When the computing method comes to adaptive mesh, it gets complicated because the algorithm needs to control mesh level and store many lists of data for further processing. The first mesh computes the values from top to down to barrier option price level, the second, third, fourth, fifth mesh computes deeper near the barrier option price level to obtain more precise call option value. The advantage of Restricted Trinomial Model is that this model can determine the number of step to perform and obtain optimal call option value. The advantage of Adaptive Mesh Method is to perform much higher efficiency than Restricted Trinomial Model.

Delta and Gamma computation introduces much performance improve for adaptive mesh method not limited to save computation time, but also the accuracy.

Most importantly, this project involves significant amount of mathematics logic and formula to construct the model using Python, taking the implementation enhances my understandings of the algorithm. It takes me to learn many powerful python library such as not only numpy, matplotlib, but also iteratortools, which establishes pricing path movement across the current price to maturity. I believe this project experience is a valuable add-on to my programming skill, financial knowledge about European call option, and the implementation of mathematics model.

Reference:

[1] “Mark Rubinstein", “Implied Binomial Trees”, The Journal of Finance, Vol. 49, No.3, Papers and Proceedings Fifty-Fourth Annual Meeting of the American Finance Association, Boston, Massachusetts, January 3-5, 1994 (Jul., 1994), 771-818

[2] “Phelim Boyle”\*\*, Mark Broadieb, Paul Glassermanb”, Monte Carlo methods for security pricing, “School of Accountancy, University of Waterloo. Waterloo. Ont. Canada NZL 3GI Graduaie School of Business, Columbia University, New Yorh, NY, 10027, USA

[3] “Francis A. Longstaff, Eduardo S.Schwartz”, “Valuing American Options by Simulation: A Simple Least-Squares Approach”, UCLA

**Appendices**

Code:

The python program code file has been attached with the submission. With Pycharm and libraries installed, the python code will be executable.