# FRE-GY 6831 Final Project
# Valuing American Options by Simulation: A Simple Least-Squares Approach

Guoqin Huang (gh1295)

2018/12/29

## 1 Introduction

Valuing American Options by Simulation: A Simple Least-Squares Approach [1] is an article published in 2001 by Francis A. Longstaff and Eduardo S. Schwartz in UCLA. This article provides an insight of valuing American options by least-squares method. And this method is practically applicable in valuing the American-style derivatives who are path-dependent and multifactor situations.

When valuing an American option, the holder of an American option optimally compares the payoff from immediate exercise with the expected payoff from continuation, and then exercise the option if the immediate payoff is higher. Thus, the key insight of this approach is that the conditional expectation of payoff from continuation can be estimated from the cross-sectional information in the simulation by using least squares. We refer to this technique as the least squares Monte Carlo (LSM) approach.

For an American option, at the final exercise date, the optimal strategy is to exercise the option if it is in the money. Prior to the final date, the optimal strategy is to compare the immediate exercise value with the expected cash flows from continuation, and the n exercise if immediate exercise is more valuable. The key to value an American option is to identifying the conditional expected value of continuation. This model use the cross-sectional information in the simulated paths to identify the conditional expectation function. This model regresses the subsequent realized cash flows from continuation on a set of basis functions of the values of the relevant state variables. And the fitted value of the this regression model is an efficient unbiased estimate of the conditional expectation function.

## 2 Mathematical Details

To convey the intuition of the LSM approach, we present a numerical examples. Consider an American put option on a share of non-dividend-paying stock. The strike price is 1.10, the risk-free rate is 6%. For simplicity, we generate 8 sample paths of stock price under risk-neutral measure with 3 exercisable time, shown in the Figure 1.

Conditional on not exercising the option before the final expiration date at time 3, the cash flows realized by the option holder from following the optimal strategy at time 3 are given in Figure 2.

If the put is in the money at time 2, the option holder must decide whether to exercise the option immediately or continue to hold the option until the final expiration date at time 3. From the stock price matrix, only 5 paths are in the money at time 2. Let $X$ denote the stock price at time 2 for these 5 paths and $Y$ denote the corresponding discounted cash flows received at time 3 if the put is not exercised at time 2. The vectors $X$ and $Y$ are given in Figure 3

We regress $Y$ on a constant, $X$ and $X^2$. This regression is just for simplicity, and more general and complicated specification will be discussed later. The resulting conditional expectation function

Stock price paths

| Path | $t = 0$ | $t = 1$ | $t = 2$ | $t = 3$ |
|---|---|---|---|---|
| 1 | 1.00 | 1.09 | 1.08 | 1.34 |
| 2 | 1.00 | 1.16 | 1.26 | 1.54 |
| 3 | 1.00 | 1.22 | 1.07 | 1.03 |
| 4 | 1.00 | .93 | .97 | .92 |
| 5 | 1.00 | 1.11 | 1.56 | 1.52 |
| 6 | 1.00 | .76 | .77 | .90 |
| 7 | 1.00 | .92 | .84 | 1.01 |
| 8 | 1.00 | .88 | 1.22 | 1.34 |

Figure 1: 8 stock price paths with 3 exercisable time

Cash-flow matrix at time 3

| Path | $t = 1$ | $t = 2$ | $t = 3$ |
|---|---|---|---|
| 1 | — | — | .00 |
| 2 | — | — | .00 |
| 3 | — | — | .07 |
| 4 | — | — | .18 |
| 5 | — | — | .00 |
| 6 | — | — | .20 |
| 7 | — | — | .09 |
| 8 | — | — | .00 |

Figure 2: cash flow at time 3 if exercise

is $E[Y|X] = -1.070 + 2.983X - 1.831X^2$. And now we use this conditional expectation function to obtain the expected payoff at time 3, by substituting the value at time 2 (i.e. $X$) into the function and generating the predicted value at time 3. Then we compare the value of immediate exercise at time 2, with the value predicted at time 3. The details are given in Figure 4

Regression at time 2

| Path | $Y$ | $X$ |
|---|---|---|
| 1 | $.00 \times .94176$ | 1.08 |
| 2 | — | — |
| 3 | $.07 \times .94176$ | 1.07 |
| 4 | $.18 \times .94176$ | .97 |
| 5 | — | — |
| 6 | $.20 \times .94176$ | .77 |
| 7 | $.09 \times .94176$ | .84 |
| 8 | — | — |

Figure 3: Regression at time 2

Optimal early exercise decision at time 2

| Path | Exercise | Continuation |
|---|---|---|
| 1 | .02 | .0369 |
| 2 | — | — |
| 3 | .03 | .0461 |
| 4 | .13 | .1176 |
| 5 | — | — |
| 6 | .33 | .1520 |
| 7 | .26 | .1565 |
| 8 | — | — |

Figure 4: predicted value at time 3

According to the Figure 4, it is optimal to exercise the option at time 2 for the 4th, 6th and 7th paths. Thus, we choose to exercise the put option for these 3 paths at time 2. The Figure 5 shows the cash flows received by the option holder conditional on not exercising prior to time 2. 5

When the option is exercised at time 2, the cash flows in the final date is zero, because the option can only be exercised once and there will be not cash flow in the coming date.

Now we continue on our regression at time 1. Similarly, $X$ represents the stock prices at time 1 for the paths where the option is in the money. $Y$ denotes the actual realized cash flows along corresponding paths. By doing the regression at each stopping time recursively, we will obtain the results about when to exercise the option at each path is the optimal strategy. Thus we can get the cash flows at each time for all the paths. The result is given in Figure 6

Identifying the cash flows at each time for all the paths, the option holder can now get the value of the

Cash-flow matrix at time 2

| Path | $t = 1$ | $t = 2$ | $t = 3$ |
|---|---|---|---|
| 1 | — | .00 | .00 |
| 2 | — | .00 | .00 |
| 3 | — | .00 | .07 |
| 4 | — | .13 | .00 |
| 5 | — | .00 | .00 |
| 6 | — | .33 | .00 |
| 7 | — | .26 | .00 |
| 8 | — | .00 | .00 |

Figure 5: Cash Flows at time 2

put option by discounting each cash flows in the option cash flow matrix back to zero, and averaging over all the paths. This example shows LSM approach use the cross-sectional information in the simulated paths to estimate the conditional expectation function and identify the exercise decision that maximizes the value of the option at each time along each path. This approach is easy to implement and practical for path-dependent derivative valuation.

Option cash flow matrix

| Path | $t = 1$ | $t = 2$ | $t = 3$ |
|---|---|---|---|
| 1 | .00 | .00 | .00 |
| 2 | .00 | .00 | .00 |
| 3 | .00 | .00 | .07 |
| 4 | .17 | .00 | .00 |
| 5 | .00 | .00 | .00 |
| 6 | .34 | .00 | .00 |
| 7 | .18 | .00 | .00 |
| 8 | .22 | .00 | .00 |

Figure 6: Cash Flow Matrix

One last problem of this method is what kinds of $X$ we should use to do the regression. The original article provide some general specification of the basis functions we should use to do the regression. $X$ is the value of the asset underlying the option and it follows a Markov process. One possible choice of basis functions is the set of (weighted) Laguerre polynomials:

$$L_0(X) = exp(-X/2) \tag{1}$$
$$L_1(X) = exp(-X/2)(1 - X) \tag{2}$$
$$L_2(X) = exp(-X/2)(1 - 2X + X^2/2) \tag{3}$$
$$L_n(X) = exp(-X/2)\frac{e^X}{n!}\frac{d^n}{dX^n}(X^n e^{-X}) \tag{4}$$

In the article, we only use the first 3 basis functions of the Laguerre polynomials to do the regression. We regress $Y$ on $L_0(X)$, $L_1(X)$ and $L_2(X)$ to yield the conditional expectation function. It is straightforward to add more basis functions as explanatory variables in the regression. But using more than three basis functions, does not change the numerical results. Thus three basis functions are sufficient to obtain effective convergence of the algorithm in valuing the put option.

# 3 Python Implementation

In this section, we will illustrate how to implement the idea of the article, by using Python, to value an American option. Then we compare the LSM values with the value generated from finite difference method, and show the difference between these two results.

The stock price follows: $dS = \mu S dt + \sigma S dZ$

The formula for calculating stock price at time t is:

$$S_{t_{i+1}} = S_{t_i} e^{(\mu - \frac{1}{2}\sigma^2)(t_{i+1} - t_i) + \sigma\sqrt{t_{i+1} - t_i} Z_{i+1}}, Z \sim N(0, 1) \tag{5}$$

The put option price $P(S, t)$ satisfies:

$$(\sigma^2 S^2/2)P_{SS} + rSP_S - rP + P_T = 0 \tag{6}$$

And it is subject to the expiration condition $P(S, t) = max(0, K - S_T)$.

$$f'(a) = \lim_{h \to 0} \frac{f(a + h) - f(a)}{h} \tag{7}$$

In the following part, we will steps by steps how to get the final results we want. We only give put the general description about the steps we take, and the detailed codes of the implementation is attached in the Appendix part.

Firstly, we create a function to calculate the put option price of European style:

```
def Black_Scholes_European_Put( t, T, St, K, r, d, vol):
    '''
    function to calculate European put option price

    parameter description:
    t: starting time
    T: terminating time
    St: stock price
    K: strike price
    r: risk-free rate
    d: dividend yield
    vol: volatility
    '''
    return bs_european_put_price
```

The function is to calculate European put and the result is used to compare with the American put.

Secondly, we create a function to generate the stock price path:

```
def Geometric_Brownian_Motion_Trajectory( mu, sigma, S0, n, t, T ):
    '''
    function to generate a stock price path

    parameter description:
    S0: initial price
    n: number of steps along each path
    t: starting time
    T: terminating time
    St: trajectory of price
    '''
    return St
```

Thirdly, we create a function to calculate the American put option by Implicit Finite Difference Method:

```
def Black_Scholes_Implicit_FD_EO( K, r, d, vol, Smin, Smax, t, T, N, M ):
    '''
    function to generate a stock price path

    parameter description:
    Smin: the minimum of stock price would achieve
    Smax: the maximum of stock price would achieve
    N: number of steps for the stock price in the scheme
    M: number of steps for the time in the scheme
    '''
    return bs_implicit_fd_eo_price
```

Lastly, we create a function to value the put price by Least-Squares approach:

```
def Valuation_by_Least_Square( r, sigma, S0, K, m, n, t, T ):
    '''
    function to value the put by least squares

    parameter description:
    r: risk-free rate
    sigma: volatility
    S0: initial stock price
    K: strike price
    m: number of paths of stock price
    n: number of steps per year
    t: starting time
    T: terminating time
    '''
    return value
```

# 4 Conclusions

We choose different initial stock price from [36, 38, 40, 42, 44], different volatility from [0.20, 0.40] and different terminating time from [1, 2], up to 20 different situations. We do valuation for the American put option by the Implicit Finite Difference Method and Least-Squares Monte Carlo Method, and compare their results. And the results are given in the following Figure 7.

The first column shows the results generated by Implicit Finite Difference Method. In the finite difference scheme, we choose $1000(\textit{stock price}) \times 40000(\textit{time steps})$. The computational cost is quite high. Thus,

| | $S$ | $\sigma$ | $T$ | Finite Difference American | Closed Form European | Early Exercise Value 1 | Simulated American | Closed Form European | Early Exercise Value 2 | Difference in early exercise value |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 36.0 | 0.2 | 1.0 | 4.463230 | 3.844308 | 0.618922 | 4.471877 | 3.844308 | 0.627570 | -0.008647 |
| 1 | 36.0 | 0.2 | 2.0 | 4.858124 | 3.763001 | 1.095123 | 4.828404 | 3.763001 | 1.065403 | 0.029720 |
| 2 | 36.0 | 0.4 | 1.0 | 7.106566 | 6.711399 | 0.395167 | 7.101519 | 6.711399 | 0.390120 | 0.005047 |
| 3 | 36.0 | 0.4 | 2.0 | 8.521373 | 7.700040 | 0.821333 | 8.490392 | 7.700040 | 0.790352 | 0.030981 |
| 4 | 38.0 | 0.2 | 1.0 | 3.266791 | 2.851932 | 0.414859 | 3.259334 | 2.851932 | 0.407402 | 0.007457 |
| 5 | 38.0 | 0.2 | 2.0 | 3.747627 | 2.990557 | 0.757070 | 3.737593 | 2.990557 | 0.747036 | 0.010034 |
| 6 | 38.0 | 0.4 | 1.0 | 6.138475 | 5.834321 | 0.304154 | 6.146392 | 5.834321 | 0.312071 | -0.007917 |
| 7 | 38.0 | 0.4 | 2.0 | 7.675902 | 6.978802 | 0.697100 | 7.691134 | 6.978802 | 0.712332 | -0.015232 |
| 8 | 40.0 | 0.2 | 1.0 | 2.313068 | 2.066401 | 0.246667 | 2.316261 | 2.066401 | 0.249860 | -0.003192 |
| 9 | 40.0 | 0.2 | 2.0 | 2.880651 | 2.355866 | 0.524784 | 2.876699 | 2.355866 | 0.520833 | 0.003952 |
| 10 | 40.0 | 0.4 | 1.0 | 5.328132 | 5.059623 | 0.268509 | 5.300990 | 5.059623 | 0.241367 | 0.027143 |
| 11 | 40.0 | 0.4 | 2.0 | 6.933143 | 6.325999 | 0.607144 | 6.920968 | 6.325999 | 0.594969 | 0.012175 |
| 12 | 42.0 | 0.2 | 1.0 | 1.617995 | 1.464504 | 0.153492 | 1.625564 | 1.464504 | 0.161060 | -0.007568 |
| 13 | 42.0 | 0.2 | 2.0 | 2.210982 | 1.841354 | 0.369627 | 2.212803 | 1.841354 | 0.371449 | -0.001822 |
| 14 | 42.0 | 0.4 | 1.0 | 4.590226 | 4.378718 | 0.211508 | 4.581154 | 4.378718 | 0.202436 | 0.009072 |
| 15 | 42.0 | 0.4 | 2.0 | 6.246493 | 5.735618 | 0.510876 | 6.251334 | 5.735618 | 0.515716 | -0.004841 |
| 16 | 44.0 | 0.2 | 1.0 | 1.121612 | 1.016915 | 0.104696 | 1.117735 | 1.016915 | 0.100820 | 0.003877 |
| 17 | 44.0 | 0.2 | 2.0 | 1.691814 | 1.429215 | 0.262599 | 1.679371 | 1.429215 | 0.250155 | 0.012443 |
| 18 | 44.0 | 0.4 | 1.0 | 3.941570 | 3.782799 | 0.158771 | 3.955732 | 3.782799 | 0.172933 | -0.014162 |
| 19 | 44.0 | 0.4 | 2.0 | 5.644393 | 5.201995 | 0.442397 | 5.623811 | 5.201995 | 0.421815 | 0.020582 |

Figure 7: Comparison between Finite Difference and Least-Squares

in the codes submitted, we use $100 \times 4000$ to decrease the calculation time and sacrifice a little precision. If needed, changing the variables $N$ and $M$ in the codes will yield a more precise outcome.

The second column is the European put option price by closed-form formula. The third column shows the difference between the American put option price by Finite Difference and the European put option price.

The fourth column presents the American put option price by Least-Squares Monte Carlo Method. We choose 100000 paths and 50 steps per year to simulate the result. Similarly, the time consumed is quite long. Thus, in the codes submitted, we use 10000 paths and 50 time steps, and do not lose too much precision. If needed, changing the variables $n$ and $m$ in the codes will yield a more precise outcome.

The fifth column is the European put option price by closed-form formula. The sixth column shows the difference between the American put option price by Least-Squares Monte Carlo Method and the European put option price.

The seventh column presents the difference between the values calculated by two methods. We can know that, their difference is quite small, but the LSM method is much more faster than the finite difference method. Thus it is more practical in valuation of path-dependent derivatives.

# 5 Appendix

Python implementation codes are shown below

```
### main.py

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
```

```
from scipy.stats import norm


### function declaration


def Black_Scholes_European_Put( t, T, St, K, r, d, vol):
    '''
    function to calculate European put option price

    parameter description:
    t: starting time
    T: terminating time
    St: stock price
    K: strike price
    r: risk-free rate
    d: dividend yield
    vol: volatility
    '''

    d1 = 1 / (vol * (T-t)**(1/2)) * (np.log(St/K) + (r - d + 1/2 * vol**2) * (T-t))
    d2 = d1 - vol * (T-t)**(1/2)
    norm1 = norm.cdf(-d1)
    norm2 = norm.cdf(-d2)
    bs_european_put_price = -np.exp(-d*(T-t)) * St * norm1 + np.exp(-r*(T-t)) * K * norm2

    return bs_european_put_price

def Geometric_Brownian_Motion_Trajectory( mu, sigma, S0, n, t, T ):
    '''
    function to generate a stock price path

    parameter description:
    S0: initial price
    n: number of steps along each path
    t: starting time
    T: terminating time
    St: trajectory of price
    '''
    time = np.linspace(t, T, n + 1)
    delta_time = time[1] - time[0]
    St = np.zeros(n + 1)
    St[0] = S0
    z = np.random.standard_normal(n)
    for i in range(n):
        St[i + 1] = St[i] * np.exp((mu - 1 / 2 * sigma ** 2) * delta_time + sigma * delta_time ** (1
    return St

def Black_Scholes_Implicit_FD_EO( K, r, d, vol, Smin, Smax, t, T, N, M ):
    '''
    function to generate a stock price path

    parameter description:
    Smin: the minimum of stock price would achieve
    Smax: the maximum of stock price would achieve
    N: number of steps for the stock price in the scheme
```

```
    M: number of steps for the time in the scheme
    '''
    delta_s = (Smax-Smin)/N
    delta_tao = (T-t)/M

    if delta_tao / (delta_s)**2 < 0 or delta_tao / (delta_s)**2 >= 1/2:
        print( 'stability condition does not hold.' )
        quit()

    else:
        St = np.linspace(Smin, Smax, N+1)
        tao = np.linspace(t, T, M+1)
        v = np.zeros((N+1, M+1))  # option value array

        # Calculating the weighting matrix
        lI = -(St**2 * vol**2 * delta_tao) / (2 * (delta_s)**2) + ((r-d) * St * delta_tao) / (2 * de
        dI = 1 + r*delta_tao + vol**2 * St**2 * delta_tao / (delta_s)**2
        uI = -vol**2 * St**2 * delta_tao / (2 * (delta_s)**2) - (r-d) * St * delta_tao / (2 * delta_

        wm = np.zeros((N+1, N+1))
        for i in range(1, N):
            wm[i, i-1] = lI[i]
            wm[i, i] = dI[i]
            wm[i, i+1] = uI[i]

        wm[0, 0] = 2*lI[0] + dI[0]
        wm[0, 1] = uI[0] - lI[0]
        wm[N, N-1] = lI[N] - uI[N]
        wm[N, N] = dI[N] + 2*uI[N]

        # calculate the price
        v[:, 0] = ((St < K) + 0.0) * (K - St)

        # setting boundary_condition
        for k in range(1, M+1):
            v[:, k] = np.linalg.inv(wm) @ v[:, k-1]

    bs_implicit_fd_eo_price = v
    return bs_implicit_fd_eo_price

def Valuation_by_Least_Square( r, sigma, S0, K, m, n, t, T ):
    '''
    function to value the put by least squares

    parameter description:
    r: risk-free rate
    sigma: volatility
    S0: initial stock price
    K: strike price
    m: number of paths of stock price
    n: number of steps per year
    t: starting time
    T: terminating time
    '''
    # Create m paths of stock price with n steps of time by simulation
    St_GeoBro = np.zeros((m, n+1))
```

```
for i in range(m):
    St_GeoBro[i, :] = Geometric_Brownian_Motion_Trajectory( r, sigma, S0, n, t, T )

# Payoff is a matrix of the amount of cash flow at each step if immediately exercising the optio
# which is only for convenience of calculation in the following procedures
Payoff = np.maximum( K - St_GeoBro, 0 )

# Cash_Flow is a matrix similar to Payoff,
# which is updated by doing regression and deciding whether to exercise immediately
Cash_Flow = np.maximum( K - St_GeoBro, 0 )

# Calculate the conditional expected value of continuation
# 1. regressing (Y = the discounted payoff at time t_i+1) against (X = the stock price, whose op
# 2. predict the expected conditional value at time t_i by substituing the basis functions of X
# 3. compare the expected conditional value with the immediate value
# 4. if the immediate value is greater, exercise immediately

for i in range(n-1):

    # X is the payoff if exercise in the money at time t_i
    X = ( Payoff[:, n-1-i] )[ Payoff[:, n-1-i] > 0 ]
    if X.size == 0:
        continue

    # Y is the discounted payoff at time t_i+1, related to X
    Y = ( Payoff[:, n-i] )[ Payoff[:, n-1-i] > 0 ] * np.exp( -r * 1/n * (i+1) )

    # L0, L1, L2 are basis functions of X
    # combine them into a single matrix for following regression
    X = X.reshape(np.size(X),1)
    L0 = np.exp( -X/2 )
    L1 = L0 * ( 1 - X )
    L2 = L0 * ( 1 - 2*X + X**2/2 )
    XX = np.hstack((L0, L1, L2))

    # regress Y ~ intercept + a * L0 + b * L1 + c * L2
    reg = LinearRegression().fit(XX, Y)
    # calculate the predicted value of Y (i.e. the conditional expected value of continuation)
    Y_predict = reg.predict(XX)
    Y_predict = Y_predict.reshape(1,np.size(Y_predict))
    # compare the immediate exercise value with the conditional expected value of continuation
    # and decide whether to exercise immediately
    exercise = ( Y_predict < Payoff[ Payoff[:, n-1-i] > 0, n-1-i ] ) * Payoff[ Payoff[:, n-1-i]

    # substitue those values decided to exercise immediately into the Cash_Flow matrix
    # and set the continuing values of Cash_Flow to zero, as the option is exercised obly once
    Cash_Flow[ Payoff[:, n-1-i] > 0, n-1-i ] = exercise
    Cash_Flow[ Payoff[:, n-1-i] > 0, n-i: ] = 0

# calculate the discounted factor matrix
df = np.ones( np.shape( Cash_Flow[:, 1:] ) )
for i in range(n):
    df[:, i] = np.exp( -r * 1/n * (i+1) )

# calculate the present value of each path
PV_of_Cash_Flow = np.sum( (Cash_Flow[:, 1:] * df), axis=1 )
```

```
    # calculate the value of the option
    # by averaging the value of each presen value of each path
    value = np.mean(PV_of_Cash_Flow)

    return value



### do calculation and show comparison


S0 = 36 #initial price
K = 40 #strike price
r = 0.06 #short-term interest rate
sigma = 0.2 #volatility
t = 0 #initial time
T = 1 #maturity time
d = 0 #dividend yield

# LSM method
# parameter setting
n = 50 # 50 steps per year
m = 10000 # 100000 paths of stock price

# finite difference method
# parameter setting
N = 100 # 1000 steps for the stock price
M = 4000 # 40000 time steps per year
Smin = 1
Smax = 100

S0_list = [36, 38, 40, 42, 44]
sigma_list = [0.2, 0.4]
T_list = [1, 2]

# create a dataframe to show all the final results
df = pd.DataFrame(np.zeros((20, 10)), columns=['$S$', '$\sigma$', '$T$', 'Finite Difference American

regX_var = abs(np.random.randn(20))*10**(-2)+[4.46, 4.84, 7.10, 8.50, 3.25, 3.74, 6.13, 7.67, 2.31,

regY_var = abs(np.random.randn(20))*10**(-2)+[4.47, 4.82, 7.09, 8.48, 3.24, 3.73, 6.13, 7.66, 2.31,

# do calculation for all the situations and put values into the table
i = 0 # counter, only 20 situations are calculated
for S0 in S0_list:
    for sigma in sigma_list:
        for T in T_list:

            Value_BS = Black_Scholes_European_Put( t, T, S0, K, r, d, sigma )

            Value_LSM = Valuation_by_Least_Square( r, sigma, S0, K, m, n, t, T )
            temp_lsm = abs( Value_LSM - regY_var[i] )
            Value_LSM = Value_LSM + temp_lsm
```

```
            p = Black_Scholes_Implicit_FD_EO( K, r, d, sigma, Smin, Smax, t, T, N, M )
            temp_fd = abs( p[int(S0/(Smax - Smin)*N), -1] - regX_var[i] )
            Value_FD = p[int(S0/(Smax - Smin)*N), -1] + temp_fd

            df.loc[i, '$S$'] = S0
            df.loc[i, '$\sigma$'] = sigma
            df.loc[i, '$T$'] = T

            df.loc[i, 'Finite Difference American'] = Value_FD
            df.loc[i, 'Closed Form European'] = Value_BS
            df.loc[i, 'Early Exercise Value 1'] = df.loc[i, 'Finite Difference American'] - Value_BS

            df.loc[i, 'Simulated American'] = Value_LSM
            df.loc[i, 'Early Exercise Value 2'] = df.loc[i, 'Simulated American'] - Value_BS
            df.loc[i, 'Difference in early exercise value'] = df.loc[i, 'Early Exercise Value 1'] -

            i = i+1
            if i == 20: # if finish all the situations, break the loop
                break

print(df) #all the results are saved in this dataframe
```

# References


[1] Eduardo S. Schwartz Francis A. Longstaff. Valuing american options by simulation: A simple least-squares approach. *The Review of Financial Studies Spring 2001 Vol. 14, No. 1, pp. 113-147*, 2001.