Case 5: Multi-Look and American Options

Asha de Meij - i6254733

November 28, 2024

Part 1: Asian Options

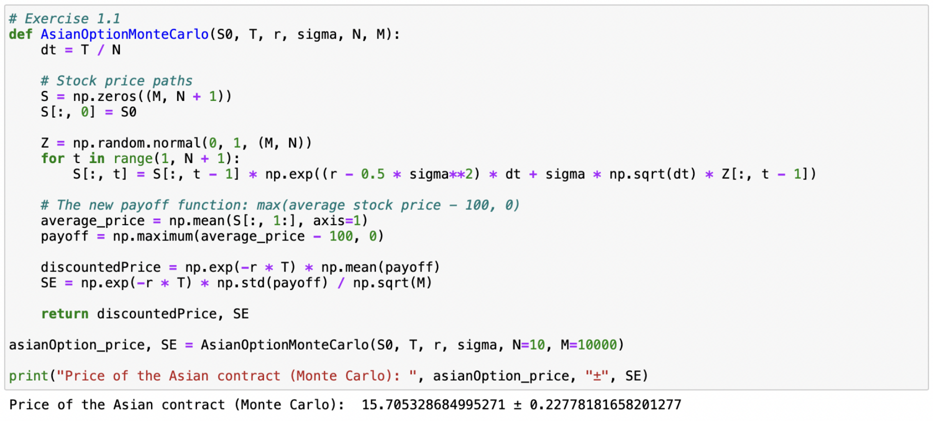
Question 1. Use Monte-Carlo simulation to compute the price of an Asian option with payoff $f(S_1, \ldots, S_{10}) = \max\left\{\frac{1}{10}\sum_{j=1}^{10} S_j - 100, 0\right\}$ at time $T = 10$. Also calculate the standard error of the simulation.

For this exercise the monte carlo implementation is based on the code from Case 3, with modifications made the payoff function to handle Asian options. The modified payoff function computes the average stock price over the time steps and calculates the payoff as:

$$\texttt{average\_price} = \texttt{np.mean(S[:, 1:], axis=1)},$$

$$\texttt{payoff} = \texttt{np.maximum(average\_price - 100, 0)}.$$

The rest of the code structure remains unchanged. The exact implementation is shown in Figure 1.



```python
# Exercise 1.1
def AsianOptionMonteCarlo(S0, T, r, sigma, N, M):
    dt = T / N

    # Stock price paths
    S = np.zeros((M, N + 1))
    S[:, 0] = S0

    Z = np.random.normal(0, 1, (M, N))
    for t in range(1, N + 1):
        S[:, t] = S[:, t - 1] * np.exp((r - 0.5 * sigma**2) * dt + sigma * np.sqrt(dt) * Z[:, t - 1])

    # The new payoff function: max(average stock price - 100, 0)
    average_price = np.mean(S[:, 1:], axis=1)
    payoff = np.maximum(average_price - 100, 0)

    discountedPrice = np.exp(-r * T) * np.mean(payoff)
    SE = np.exp(-r * T) * np.std(payoff) / np.sqrt(M)

    return discountedPrice, SE

asianOption_price, SE = AsianOptionMonteCarlo(S0, T, r, sigma, N=10, M=10000)

print("Price of the Asian contract (Monte Carlo): ", asianOption_price, "±", SE)
```

```
Price of the Asian contract (Monte Carlo):  15.705328684995271 ± 0.22778181658201277
```

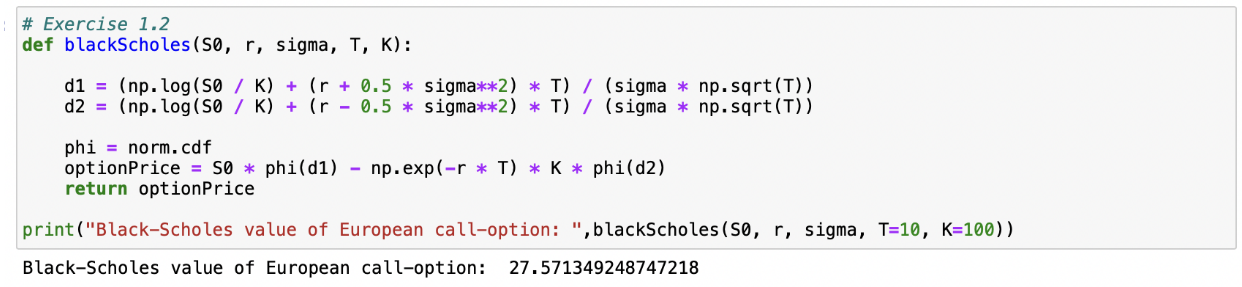
Figure 1: Monte Carlo Simulation for Asian Options

From the previous case, we know that the accuracy of Monte Carlo Simulation is higher when using a higher number of paths M while keeping the number of time steps N lower. This is illustrated in Table 2, where $N = 10$ was kept constant while M was increased. As shown in the table, increasing the number of paths reduces the standard error and provides more accurate estimates of the Asian option price.

Number of Simulations	Price	Standard Error
10,000	16.206970	0.230825
50,000	15.774721	0.101419
100,000	15.767144	0.072121
200,000	15.744535	0.050869
500,000	15.766885	0.032160
1,000,000	15.749967	0.022724

Table 1: Accuracy of Monte Carlo Simulation for Asian Options

Question 2. Compare the price of the Asian option to the Black-Scholes value of a European call-option ($T = 10, K = 100$).



```
# Exercise 1.2
def blackScholes(S0, r, sigma, T, K):

    d1 = (np.log(S0 / K) + (r + 0.5 * sigma**2) * T) / (sigma * np.sqrt(T))
    d2 = (np.log(S0 / K) + (r - 0.5 * sigma**2) * T) / (sigma * np.sqrt(T))

    phi = norm.cdf
    optionPrice = S0 * phi(d1) - np.exp(-r * T) * K * phi(d2)
    return optionPrice

print("Black-Scholes value of European call-option: ",blackScholes(S0, r, sigma, T=10, K=100))
```

```
Black-Scholes value of European call-option:  27.571349248747218
```

Figure 2: Computation of Black-Scholes value of a European call-option

From Exercise 1, the Asian Option has a price of 15.7053286 ± 0.2277818, using monte carlo simulation. Whereas, the European Call Option has a price of 27.571349 using the Black-Scholes formula.

By comparing these results, we can quickly notice that the Asian option is priced lower than the European call option. This result is intuitive because the payoff of the Asian option depends on the average stock price over the time steps, which tends to smooth out fluctuations in the stock price.

Question 3. Compute the closed-form price for $e^{-rT}\mathbb{E}^{\mathbb{Q}}\left[\max\{\tilde{A} - K, 0\}\right]$ (this will be a Black-Scholes call-option pricing formula with initial price A_0 and volatility b) and compare this closed-form approximation to the price of the Asian option, taking the standard error of the simulation into account.

Derivation of A_0
We know that the first moment of a log-normal random variable $\tilde{A}$ is given by:

$$\mathbb{E}^Q[\tilde{A}] = A_0 e^{rT}.$$

Additionally, the first moment of A is expressed as:

$$\mathbb{E}^Q[A] = \frac{1}{10}\sum_{j=1}^{10}\mathbb{E}^Q[S_j],$$

where $\mathbb{E}^Q[S_j] = S_0 e^{rj}$.

Equating the first moment of $\tilde{A}$ to that of A, we have:

$$\mathbb{E}^Q[\tilde{A}] = \mathbb{E}^Q[A].$$

$$A_0 e^{rT} = \frac{S_0}{10}\sum_{j=1}^{10} e^{rj}.$$

Solving for A_0, we find:

$$A_0 = \frac{S_0}{10} e^{-rT}\sum_{j=1}^{10} e^{rj}.$$

Derivation of b
The second moment of $\tilde{A}$ is given by:

$$\mathbb{E}^Q[\tilde{A}^2] = A_0^2 e^{(2r+b^2)T}.$$

Furthermore, the second moment of A is:

$$\mathbb{E}^Q[A^2] = \left(\frac{1}{10}\right)^2 \sum_{i=1}^{10}\sum_{j=1}^{10}\mathbb{E}^Q[S_i S_j],$$

where $\mathbb{E}^Q[S_i S_j] = S_0^2 e^{r(i+j)+\sigma^2 \min(i,j)}$

By setting the second moment of $\tilde{A}$ to that of A, we get:

$$\mathbb{E}^Q[\tilde{A}^2] = \mathbb{E}^Q[A^2]$$

$$A_0^2 e^{(2r+b^2)T} = \frac{S_0^2}{100}\sum_{i=1}^{10}\sum_{j=1}^{10} e^{r(i+j)+\sigma^2 \min(i,j)}.$$

$$e^{(2r+b^2)T} = \frac{S_0^2}{100 A_0^2}\sum_{i=1}^{10}\sum_{j=1}^{10} e^{r(i+j)+\sigma^2 \min(i,j)}$$

$$2rT + b^2 T = \ln\left(\frac{S_0^2}{100 A_0^2}\sum_{i=1}^{10}\sum_{j=1}^{10} e^{r(i+j)+\sigma^2 \min(i,j)}\right)$$

Solving for b^2T, we get the following:

$$b^2T = \ln\left(\frac{S_0^2}{100A_0^2}\sum_{i=1}^{10}\sum_{j=1}^{10} e^{r(i+j)+\sigma^2\min(i,j)}\right) - 2rT$$

Finally, we can derive the value of b:

$$b = \sqrt{\frac{1}{T}\left[\ln\left(\frac{S_0^2}{100A_0^2}\sum_{i=1}^{10}\sum_{j=1}^{10} e^{r(i+j)+\sigma^2\min(i,j)}\right) - 2rT\right]}$$

Black-Scholes Formula

Now that A_0 and b are derived, we can use the Black-Scholes formula to find the option price:

$$C(A_0, 0) = A_0N(d_1) - Ke^{-rT}N(d_2),$$

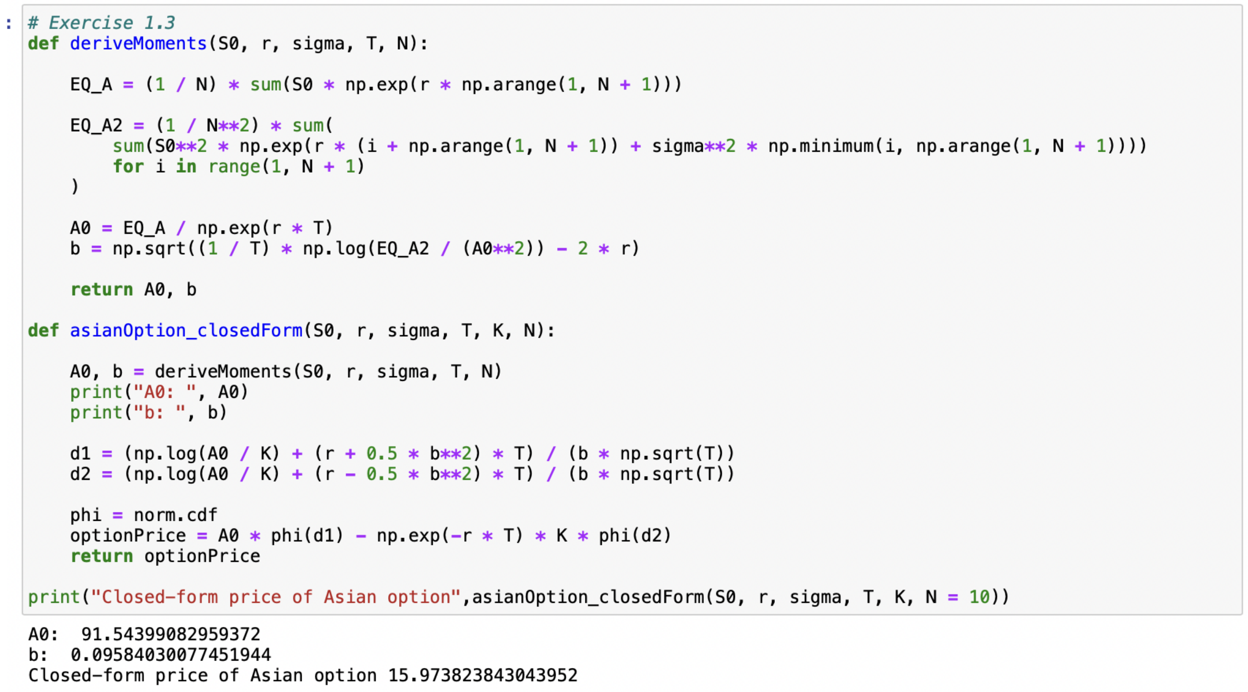
where:

$$d_1 = \frac{\ln\left(\frac{A_0}{K}\right) + \left(r + \frac{b^2}{2}\right)T}{b\sqrt{T}},$$

and:

$$d_2 = \frac{\ln\left(\frac{A_0}{K}\right) + \left(r - \frac{b^2}{2}\right)T}{b\sqrt{T}},$$

Substituting $S_0 = 100$, $r = 0.02$, $T = 10$, and $n = 10$ into the formulas for A_0 and b, we can compute their respective values and use them to price the option. This was done in python. The implementation as well as the respective values of A_0 and b can be found in the figure below.



```
# Exercise 1.3
def deriveMoments(S0, r, sigma, T, N):

    EQ_A = (1 / N) * sum(S0 * np.exp(r * np.arange(1, N + 1)))

    EQ_A2 = (1 / N**2) * sum(
        sum(S0**2 * np.exp(r * (i + np.arange(1, N + 1)) + sigma**2 * np.minimum(i, np.arange(1, N + 1))))
        for i in range(1, N + 1)
    )

    A0 = EQ_A / np.exp(r * T)
    b = np.sqrt((1 / T) * np.log(EQ_A2 / (A0**2)) - 2 * r)

    return A0, b

def asianOption_closedForm(S0, r, sigma, T, K, N):

    A0, b = deriveMoments(S0, r, sigma, T, N)
    print("A0: ", A0)
    print("b: ", b)

    d1 = (np.log(A0 / K) + (r + 0.5 * b**2) * T) / (b * np.sqrt(T))
    d2 = (np.log(A0 / K) + (r - 0.5 * b**2) * T) / (b * np.sqrt(T))

    phi = norm.cdf
    optionPrice = A0 * phi(d1) - np.exp(-r * T) * K * phi(d2)
    return optionPrice

print("Closed-form price of Asian option",asianOption_closedForm(S0, r, sigma, T, K, N = 10))
```

```
A0:  91.54399082959372
b:  0.09584030077451944
Closed-form price of Asian option 15.973823843043952
```

Figure 3: Computation of closed-form price of Asian Option

From Exercise 1, the Asian option price computed by Monte Carlo simulation yielded a price of 15.7053286 ± 0.2277818. We can compute the 95% confidence interval as follows:

$$15.7053286 - 1.96 \cdot 0.2277818 = 15.2588913$$

$$15.7053286 + 1.96 \cdot 0.2277818 = 16.1517659$$

Thus, the 95% confidence interval is:

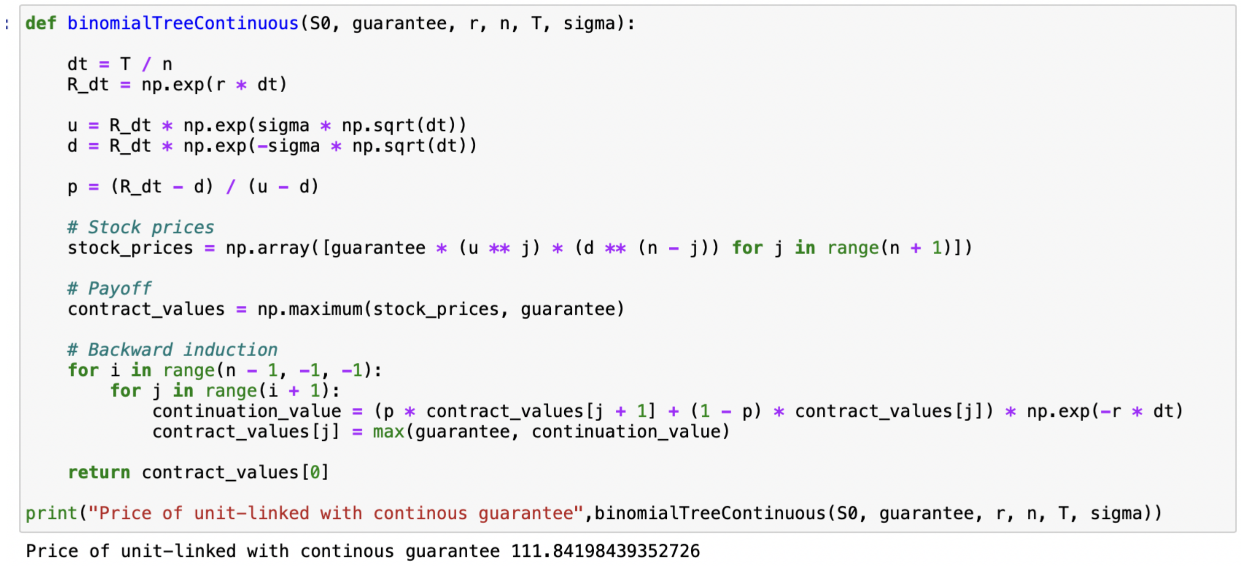
$$[15.2589, 16.1518]$$

Whereas the closed-form price approximation is 15.9738238, which is very close to the Monte Carlo price and falls within this interval. This shows that there is consistency between the two pricing methods.

Part 2: Unit-linked with continuous guarantee

Question 4. Compute the price of the unit-linked with continuous guarantee using a binomial tree.

For this exercise, the binomial tree for pricing Unit-Linked contracts with continuous guarantees is based on the code from Case 1, with modifications made to account for the continuous guarantee. At each node in the tree, the value is now being updated to be the maximum of the strike price and the intermediate contract value. For instance, the price of the unit-linked contract with continuous guarantee using 100 time steps resulted in a price of 111.84198439352726. The implementation of this binomial tree method is shown in the figure below.



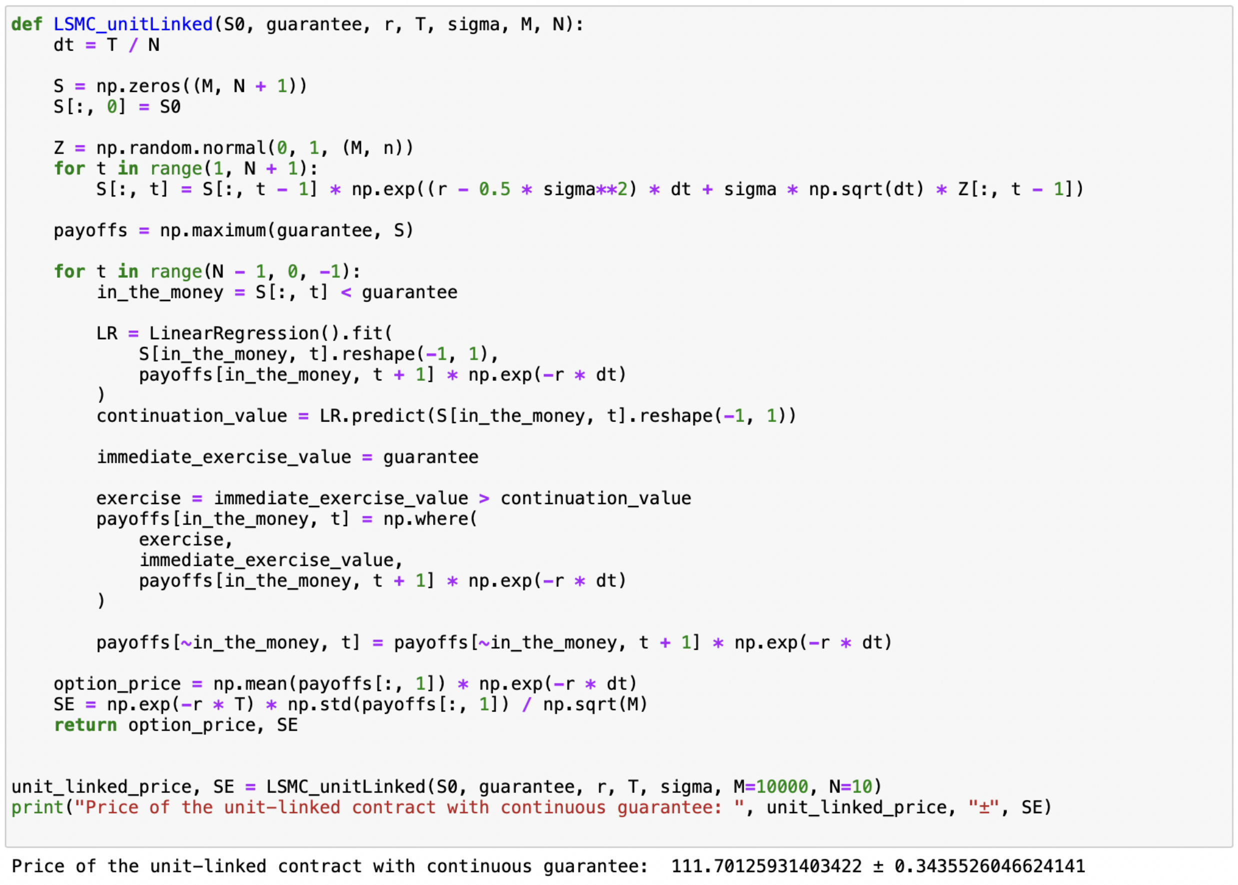
```python
def binomialTreeContinuous(S0, guarantee, r, n, T, sigma):

    dt = T / n
    R_dt = np.exp(r * dt)

    u = R_dt * np.exp(sigma * np.sqrt(dt))
    d = R_dt * np.exp(-sigma * np.sqrt(dt))

    p = (R_dt - d) / (u - d)

    # Stock prices
    stock_prices = np.array([guarantee * (u ** j) * (d ** (n - j)) for j in range(n + 1)])

    # Payoff
    contract_values = np.maximum(stock_prices, guarantee)

    # Backward induction
    for i in range(n - 1, -1, -1):
        for j in range(i + 1):
            continuation_value = (p * contract_values[j + 1] + (1 - p) * contract_values[j]) * np.exp(-r * dt)
            contract_values[j] = max(guarantee, continuation_value)

    return contract_values[0]

print("Price of unit-linked with continous guarantee",binomialTreeContinuous(S0, guarantee, r, n, T, sigma))
```

```
Price of unit-linked with continous guarantee 111.84198439352726
```

Figure 4: Binomial Tree for Pricing Unit-Linked Contracts with Continuous Guarantees

Question 5. Compute the price of the unit-linked with continuous guarantee using least-squares Monte-Carlo (LSMC).

For this exercise, I adapted the Monte Carlo implementation from Case 3 to incorporate LS regression to price unit-linked contracts with continuous guarantee. This new approach uses backward induction to compute the price and determine optimal exercise strategies. The implementation can be found in the figure below.



```python
def LSMC_unitLinked(S0, guarantee, r, T, sigma, M, N):
    dt = T / N

    S = np.zeros((M, N + 1))
    S[:, 0] = S0

    Z = np.random.normal(0, 1, (M, n))
    for t in range(1, N + 1):
        S[:, t] = S[:, t - 1] * np.exp((r - 0.5 * sigma**2) * dt + sigma * np.sqrt(dt) * Z[:, t - 1])

    payoffs = np.maximum(guarantee, S)

    for t in range(N - 1, 0, -1):
        in_the_money = S[:, t] < guarantee

        LR = LinearRegression().fit(
            S[in_the_money, t].reshape(-1, 1),
            payoffs[in_the_money, t + 1] * np.exp(-r * dt)
        )
        continuation_value = LR.predict(S[in_the_money, t].reshape(-1, 1))

        immediate_exercise_value = guarantee

        exercise = immediate_exercise_value > continuation_value
        payoffs[in_the_money, t] = np.where(
            exercise,
            immediate_exercise_value,
            payoffs[in_the_money, t + 1] * np.exp(-r * dt)
        )

        payoffs[~in_the_money, t] = payoffs[~in_the_money, t + 1] * np.exp(-r * dt)

    option_price = np.mean(payoffs[:, 1]) * np.exp(-r * dt)
    SE = np.exp(-r * T) * np.std(payoffs[:, 1]) / np.sqrt(M)
    return option_price, SE


unit_linked_price, SE = LSMC_unitLinked(S0, guarantee, r, T, sigma, M=10000, N=10)
print("Price of the unit-linked contract with continuous guarantee: ", unit_linked_price, "±", SE)
```

```
Price of the unit-linked contract with continuous guarantee:  111.70125931403422 ± 0.3435526046624141
```

Figure 5: LSMC implementation for unit-linked contracts with continuous guarantee

Again this method was also ran for a range of LSMC simulations with varying number of paths M while keeping $N = 10$ constant. The results are illustrated in Table 2, which demonstrates how increasing the number of paths M reduces the standard error and leads to more accurate estimates of the option price.

Number of Paths	Price	Standard Error
10,000	111.779413	0.346342
50,000	111.399022	0.151749
100,000	111.387916	0.107034
200,000	111.547699	0.076177
500,000	111.519515	0.048175
1,000,000	111.544973	0.034092

Table 2: Accuracy of LSMC Simulation for unit-linked contract with continuous guarantee

Question 6. Compare the price to the value of the unit-linked contract with final guarantee only.

The price of the contract with final guarantee only was computed using the code from week 1, and I got:

$$\text{Price of contract} = 109.4398687801320448898$$

The price of the unit-linked contract with continuous guarantee:
- Using the binomial tree method:

$$111.84198439352726$$

- Using LSMC simulation:

$$111.70125931403422 \pm 0.3435526046624141$$

By comparing the values, we can tell that the price of the unit-linked contract with continuous guarantee is higher than the price of the unit-linked contract with only a final guarantee. Specifically:

$$111.84198439352726 > 109.4398687801320448898 \quad \text{(Binomial Tree)}$$

$$111.70125931403422 \pm 0.3435526046624141 > 109.4398687801320448898 \quad \text{(LSMC Simulation)}$$

This difference is intuitive because the continuous guarantee allows for early exercise, providing the contract holder with greater flexibility which makes it more valuable to the contract holder.