Asian Option Project

Author: Yixin Xu

Introduction:

In our report, we examine two advanced Monte Carlo simulation techniques:the basic Monte Carlo simulation and an enhanced version using the Control Variates technique. These two methods are really helpful in handling the complexities of Asian options by Simulating multiple paths for the underlying asset price. We will show the steps using simulation of multiple paths, calculating the arithmetic average, and reducing the variance by controlling variate with geometric averages in the primary method. Our aim in this report is to find the better model by comparing the accuracy, efficiency, and adaptability of these two methods. In addition to evaluating the convergence of estimates of option prices, computation times, and standard errors, we also research how essential parameters like volatility, risk-free rate, and maturity affect the option price and how the correlation coefficient between arithmetic and geometric averages impact the effectiveness of variance reduction.

Background Information:

It is important for risk management, hedging strategies, and investment decisions of financial derivatives to accurately predict price options. Asian options face challenges since their payoff is the average price of assets over time. So, these options can't be directly priced using traditional models like Black-Scholes, which is pivotal in options pricing theory due to the inability to handle this path-dependent feature. In this report, we mainly test two methods to determine which is better for predicting Asian option prices in terms of accuracy, stability and efficiency, and also what factors will affect Asian option prices in real life.

Methods:

Method 1- Basic Monte Carlo Simulation Approach

Under the risk-neutral measure Q, the asset price S(t) follows a geometric Brownian motion described by the stochastic differential equation (SDE). Given a drift, volatility, and initial price for underlying assets, we define a pre-specified discrete point in time, simulate M paths of the prices, and calculate an arithmetic average of the asset price for each path. The payoff for the Asian option is the maximum difference between the average price and the strike price or zero. Finally, we discount the expected payoff of M paths to present value using a risk-neutral framework.

Method 2-- Enhanced Monte Carlo with Control Variates

Essentially, variance is reduced based on these known mathematical relationships within the simulation. In the case of Asian options, since the geometric average price follows a lognormal distribution in the Black-Scholes model, we use options written on the geometric average price. The formulae for options written on the geometric average price are analytical and easy to use as control variates. With this relationship embedded, we are then capable of making the results of our simulation more accurate and effective in pricing options

We actually simulate multiple paths of the underlying asset price by the geometric Brownian motion. Each path signifies a possible trajectory of the asset price over time. For each path, we calculate the arithmetic and geometric average of the asset prices. We then compute the beta coefficient, β, using the covariance between the arithmetic and geometric payoffs and the variance of the geometric payoffs, and determine the adjusted payoff for the Asian option using the arithmetic payoff, beta, geometric payoff and geometric average of the asset prices for each path. Finally, we average the adjusted payoffs across all paths and discount them back to present value using an appropriate discount rate to obtain the option price.

Expectation:

Assessing the Convergence and Accuracy of the Simulations by Varying M and n:

For the part of varying M and n, M(the number of paths) and n(the number of time steps) will affect how the option pricing simulation converges. When we predict the Asian option, we try to estimate the price more precisely by capturing the asset's price path. At that time, we can observe the variance across various combinations of M and n to determine whether the simulation is accurate.

Evaluating the Impact of Parameters such as Volatility σ, Risk-Free Rate r, and Maturity T on the Option Price:

In order to evaluate the stability and robustness of option price under different parameters, we try to measure the option's sensitivity to changes in volatility, dependency on risk-free rate, and growth trend with respect to time. This is essential for stakeholders in understanding the behavior of Asian options under various market conditions.

Comparison between basic Monte Carlo simulation and an enhanced version:

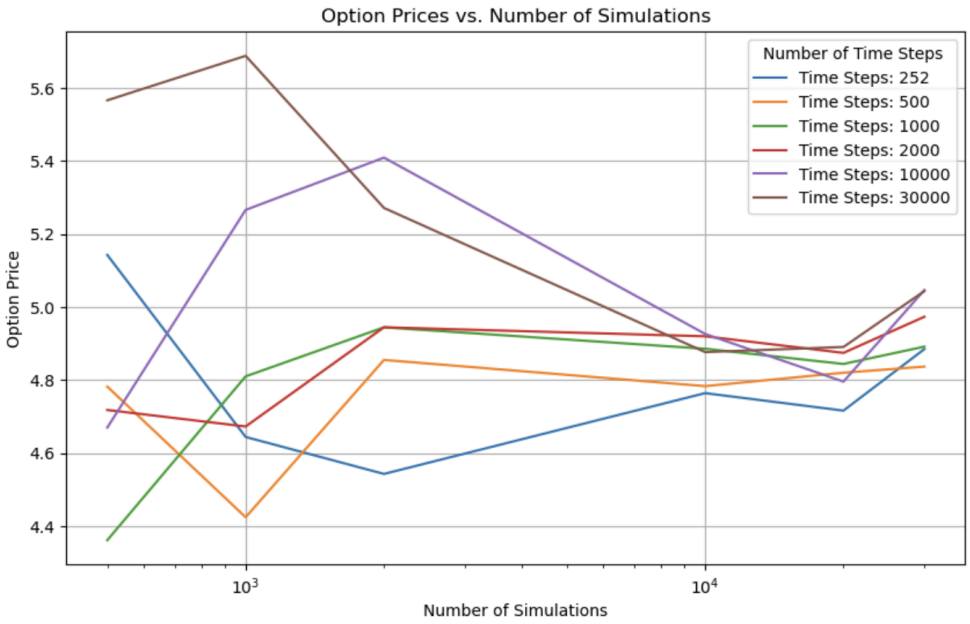
To evaluate the performances of two methods in terms of accuracy and computational efficiency, we will visualize the difference between them to verify if the variance reduction is useful. Also, we compute the correlation coefficient matrix and explore the relation between the estimates of the standard method and estimates of the control variates method to quantify the efficiency of the variance reduction technique.

Results & Analysis:

Assessing the Convergence and Accuracy of the Simulations by Varying M and n:

```
results_p = pd.DataFrame(np.array(price_simulation).reshape((6,
results_p
```

	500	1000	2000	10000	20000	30000
252	5.142281	4.644185	4.542999	4.764185	4.716312	4.884772
500	4.781955	4.424472	4.855075	4.783325	4.820062	4.836738
1000	4.361778	4.810042	4.944360	4.885688	4.844012	4.891433
2000	4.718035	4.672857	4.944437	4.919785	4.874339	4.973256
10000	4.670074	5.265812	5.408878	4.926131	4.795679	5.046402
30000	5.565791	5.687507	5.271123	4.876523	4.890530	5.043582

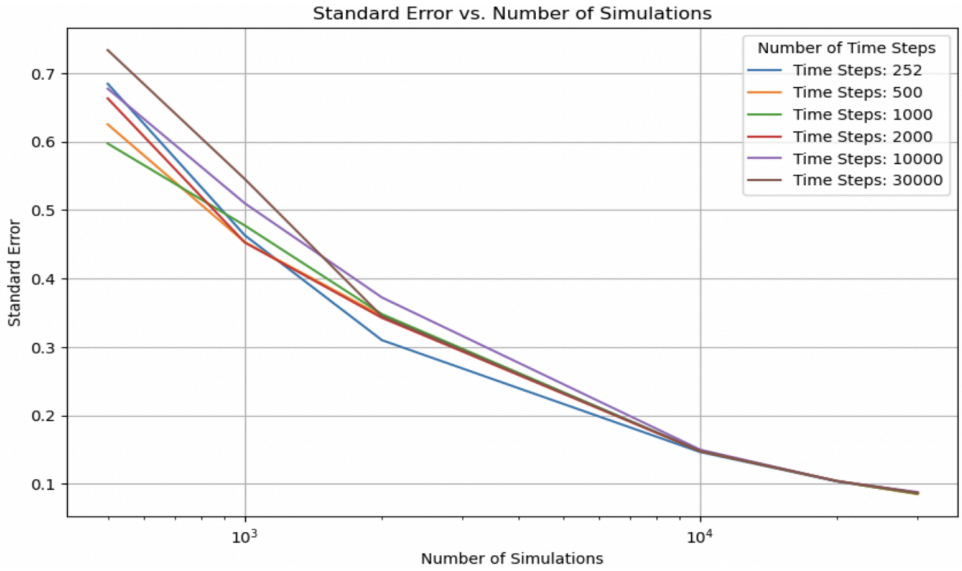


Prices Table and Plot

Prices Table and plot list the approximate prices for various combinations of N (number of time steps per path) and M (number of simulation paths). The rows are the N values, the columns are the M values. As M increases, the prices seem to level off, and lines with different colors, representing different time steps, show convergence. In addition, the fluctuations in the estimated prices are also smaller which indicates more accurate and more reliable estimates. On the other hand, increasing the number of steps,N, only leads to some small changes in the estimated prices. In our example, N changes from 252 to 30000, but the price only increases less than 0.2. Although, more time steps might catch some details of the features of a path, it might not significantly affect the average values being calculated and thus the estimated option price.

```
results_s=pd.DataFrame(np.array(Stand_Error).reshape((6,6)), co
results_s
```

	500	1000	2000	10000	20000	30000
252	0.684492	0.463050	0.309850	0.146277	0.103245	0.085640
500	0.625229	0.453060	0.345277	0.147645	0.103639	0.084748
1000	0.597187	0.477372	0.347859	0.147660	0.103436	0.085421
2000	0.663155	0.452524	0.342479	0.147720	0.103727	0.086875
10000	0.677352	0.509830	0.372245	0.149759	0.103578	0.087794
30000	0.733741	0.545271	0.344751	0.148182	0.104067	0.086944



Standard Errors Table and Plot

This table presents the standard errors associated with the estimated prices for the same combinations of *N* and M. Consistently across all *N* levels; the standard error decreases as M increases. According to the law of large numbers, this trend indicates a reduction in sampling variability. But for a number of time steps, standard errors are slightly affected by changes in N. It is clear from the plot that all paths converge to nearly zero under the effect of a higher number of simulations, but subtle changes are between different time steps.

Overall, price stabilization and standard error reduction are observed as M increases, demonstrating the convergence of the simulation. It is vital for practical implementations to choose a higher M to obtain reliable estimates—the cost of computation and the accuracy of a model trade-off. In general, higher N and M will result in better and more stable forecasts but at

the cost of a longer computation time and more resources used. As a result, N and M values can be selected according to the desired accuracy and computational resources depending on the financial modeling and risk management.

Impact of Parameters Such as Volatility σ, Risk-Free Rate r, and Maturity T on the Option Price:

```
Volatility: 0.1, Estimated Call Price: 0.8816
Volatility: 0.2, Estimated Call Price: 7.5863
Volatility: 0.3, Estimated Call Price: 16.9625
Volatility: 0.4, Estimated Call Price: 27.1696
Risk-Free Rate: 0.01, Estimated Call Price: 3.6670
Risk-Free Rate: 0.05, Estimated Call Price: 5.3201
Risk-Free Rate: 0.1, Estimated Call Price: 8.0913
Maturity: 0.5, Estimated Call Price: 1.0764
Maturity: 1.0, Estimated Call Price: 4.4365
Maturity: 2.0, Estimated Call Price: 12.4371

Volatility: 0.1, Estimated Enhanced Call Price: 0.8808
Volatility: 0.2, Estimated Enhanced Call Price: 7.5500
Volatility: 0.3, Estimated Enhanced Call Price: 16.9275
Volatility: 0.4, Estimated Enhanced Call Price: 27.1395
Risk-Free Rate: 0.01, Estimated Enhanced Call Price: 3.6502
Risk-Free Rate: 0.05, Estimated Enhanced Call Price: 5.3037
Risk-Free Rate: 0.1, Estimated Enhanced Call Price: 8.0936
Maturity: 0.5, Estimated Enhanced Call Price: 1.0678
Maturity: 1.0, Estimated Enhanced Call Price: 4.4082
Maturity: 2.0, Estimated Enhanced Call Price: 12.4110
```

Impact of Parameters on option price for two method

Financial parameters play a significant role in Asian options pricing for both two methods. Volatility ranging from 0.1 and 0.4 increases option prices from 0.88 to nearly 27. When volatility rises, the asset's average price is more likely to exceed the strike price because of increased uncertainty. Consequently, higher volatility levels result in greater returns and risk associated with options.

Financial modeling relies heavily on risk-free rates (r) as well, especially in low-interest rate environments. Our example shows that even small changes in the risk-free rate can

significantly affect option valuations, nearly doubling the price. As r increases, the discount factor decreases, which raises the present value of expected payoffs.

According to the table, option prices approximately triple for longer maturity periods, from 0.5 to 2 years. Intuitively, the asset has more time to exceed the strike price. Due to this non-linear relationship between price increases and time, asset prices will likely fluctuate over time.

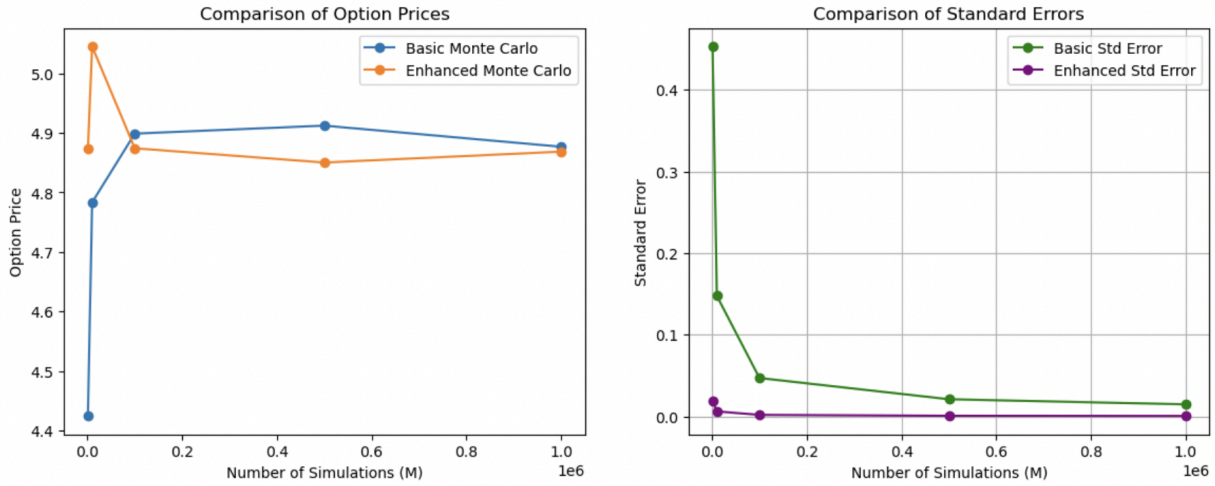
Based on our findings, volatility, maturity, and risk-free rate are essential factors in valuation. Volatility has a significant impact on price in the option price model. A risk-free rate can also affect the discount rate by changing the future payoff, which is why it is an economic indicator. Maturity involves the pricing and strategically positioning of options in a portfolio based on underlying asset price movements. Traders and risk managers must understand and leverage them to optimize investment and hedging strategies under different market conditions.

Comparison between two methods under different replications:

Replications	Basic Price	Basic Std Error	Basic Time	Enhanced Price	Enhanced Std Error	Enhanced Time	Correlation
1000	4.4245	0.4531	0.0303s	4.8726	0.0187	0.0340s	0.9993
10000	4.7833	0.1476	0.2535s	5.045	0.0062	0.6538s	0.9992
100000	4.8987	0.0471	3.1667s	4.8741	0.002	5.7862s	0.9991
500000	4.9121	0.0211	13.4660s	4.8502	0.0009	29.3288s	0.9991
1000000	4.8767	0.0149	26.9068s	4.8684	0.0006	56.5744s	0.9991

Results after increasing M for both algorithms

Comparison of Prices and Standard Error After Increasing the Number of Replications:

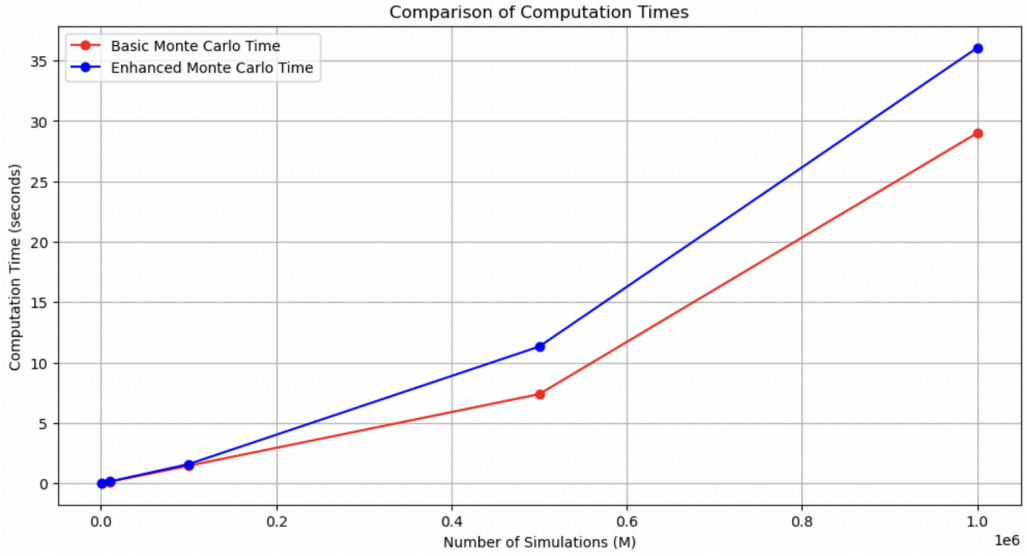


Graph.1

Based on charts and graphs, as the number of replications increases, the estimated prices converge toward a more consistent value, as a result of the law of large numbers. The table should note the reduced variability of the estimated prices as the number of paths increases for both algorithms. As measured by standard error analysis, the basic method produces higher

standard errors that decrease with more simulations, while the enhanced method produces lower standard errors indicating higher accuracy at the outset. The standard error decreases, which means the precision of our estimates is increasing. It is very clear from the method that Monte Carlo simulations increase precision with the addition of paths and thereby exhibit convergence. Reducing standard error with growing numbers of paths gives us more confidence in the convergence and reliability of the estimated option prices. The enhanced method is more effective because it consistently reduces variance with control variates.

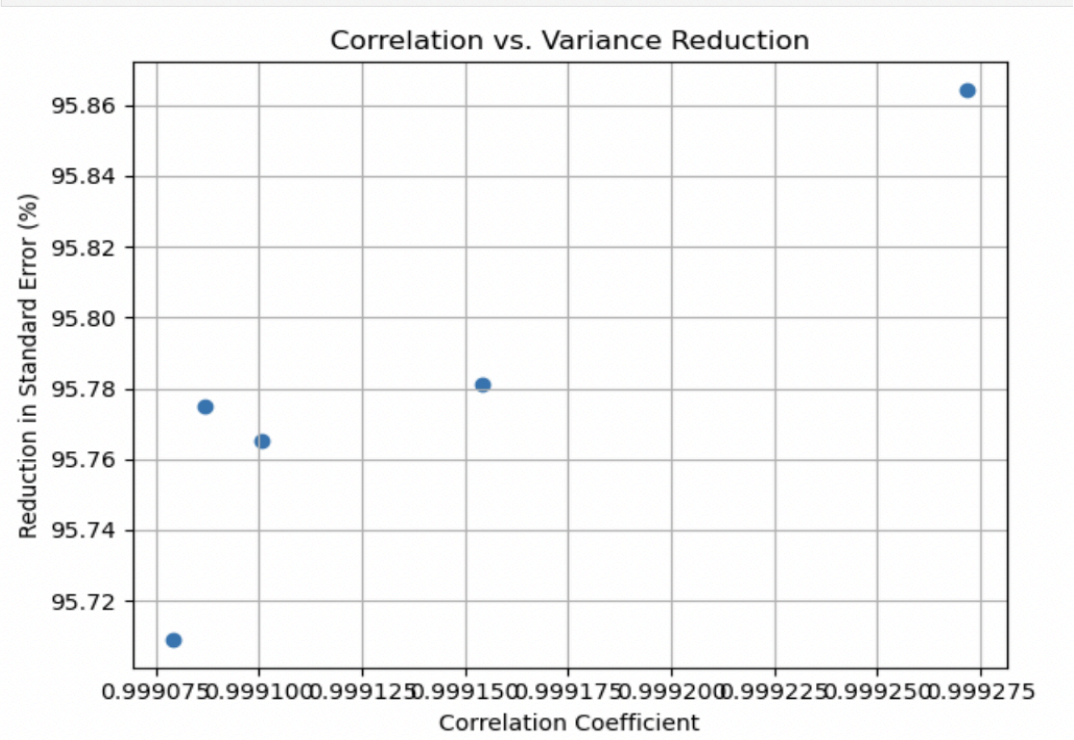
Comparison of Computation Times:



Graph.2

From this result, we can see that basic Monte Carlo is faster for scenarios with a high demand for speed, regardless of simulation size. In scenarios that require high accuracy, the enhanced method may be worth the additional computation time even though it takes longer to calculate since it is efficient in variance reduction.Thus, in terms of choosing which method, we should focus on specific requirement of financial modeling tasks to adjust the trade-off between speed and accuracy.

Impact of Correlation Coefficients:



Graph.3

For estimation of correlation coefficient, there is a clear relationship between the coefficient and percentage reduction in standard error for a financial simulation using the enhanced Monte Carlo method with control variates. High correlation coefficients substantially reduce the variance in the pricing estimates. Given such a huge efficiency, fewer simulations might be needed to achieve a certain accuracy. Consequently, control variates is a powerful tool for improving financial model precision because of this strong correlation.

Conclusion:

For pricing Asian options, this study compares basic and enhanced Monte Carlo methods. Although both methods show price convergence with increasing simulations, the enhanced method, which utilizes control variates, consistently provides more accurate and stable estimates. Due to its significant reduction in variance, it is suitable for scenarios requiring high precision, as indicated by its significant reduction in standard errors. The basic method, however, is preferred when calculations need to be performed quickly with moderate accuracy. It is important to select methods based on their accuracy and speed in order to assist in strategic decision-making and regulatory compliance in financial markets.

Reference:

1. Glasserman, Paul. “Monte Carlo Methods in Financial Engineering.” Https://Www.Bauer.Uh.Edu/Spirrong/Monte_Carlo_Methods_In_Financial_Enginee.Pdf.

2. Avellaneda , M, et al. STEVE_SHREVESTOCHASTIC_CALCUL..., cms.dm.uba.ar/academico/materias/2docuat2016/analisis_cuantitativo_en_finanzas/Steve_ShreveStochastic_Calculus_for_Finance_II.pdf.

Appendix:Code

```
import pandas as pd
import random
import numpy as np
from scipy.stats import norm
import matplotlib.pyplot as plt
from scipy.stats import gmean
```

```
import yfinance as yf
```

```
ticker = "^XSP"
xsp = yf.Ticker(ticker)
opt = xsp.option_chain('2024-12-20')
opt
```

```
S0_xsp = 456.78
opt_call = opt.calls.set_index('contractSymbol')
```

```
# Calculate real price and time to maturity

real_call_p = (16.94 + 17.56)/2

import datetime
lastTradeDate = datetime.datetime(2023,11,30)
expDate = datetime.datetime(2024,12,20)
time_to_maturity = ((expDate - lastTradeDate).days)/365
time_to_maturity # in years
```

```python
def AsianSim(S0, K, T, r, sigma, N, M, security_type):
    np.random.seed(10)  # Fixing the seed for reproducibility

    # Time step
    delta_t = T / N

    # Simulate the random components of the stock price paths
    random_matrix = np.random.normal(0, 1, (N, M))

    # Precompute constants for the simulation
    drift = (r - 0.5 * sigma**2) * delta_t
    diffusion = sigma * np.sqrt(delta_t)

    # Initialize the stock price matrix
    stock_prices = np.zeros((N + 1, M))
    stock_prices[0, :] = S0

    # Generate the stock price paths
    for t in range(1, N + 1):
        stock_prices[t, :] = stock_prices[t - 1, :] * np.exp(drift + diffusion * random_matrix[t - 1, :])

    # Calculate the arithmetic average of the stock prices for each simulation
    arithmetic_average = np.mean(stock_prices[1:, :], axis=0)  # Exclude the initial price

    # Calculate payoffs based on the type of option
    if security_type == 'call':
        payoffs = np.maximum(arithmetic_average - K, 0)
    else:
        payoffs = np.maximum(K - arithmetic_average, 0)

    # Discount the average payoff back to present value
    option_price = np.exp(-r * T) * np.mean(payoffs)
    std_error = np.std(np.exp(-r * T) * payoffs) / np.sqrt(M)

    return option_price, std_error

# Example parameters

# Parameters
S0 = S0_xsp       # initial stock price
K = 510         # strike price
T =  time_to_maturity        # time to maturity in years
r = 0.04031        # risk-free rate
sigma = 0.16668     # volatility
N = 500        # number of time steps
M = 10000       # number of simulations


# Call option pricing
call_price, Standard_Error_c= AsianSim(S0, K, T, r, sigma, N, M, 'call')
print(f"Estimated Asian Call Option Price: {call_price:.4f} Esmated Standard Error:{Standard_Error_c:4f}")

# Put option pricing
put_price, Standard_Error_p = AsianSim(S0, K, T, r, sigma, N, M, 'put')
print(f"Estimated Asian Put Option Price: {put_price:.4f} Esmated Standard Error:{Standard_Error_p:4f}")
```

```python
# Replications levels
replications = [1000, 10000, 100000, 1000000]

results = []
for M in replications:
    price, error = AsianSim(S0, K, T, r, sigma, N, M,'call')
    results.append((M, price, error))
# Display results
print("Number of Paths | Estimated Price | Standard Error")
for result in results:
    print(f"{result[0]:>15} | {result[1]:>15.4f} | {result[2]:>14.4f}")
```

```python
N_2 = [252, 500, 1000, 2000, 10000,30000]
M_2 = [500, 1000, 2000, 10000, 20000,30000]

price_simulation = []
Stand_Error=[]
for i in N_2:
  for j in M_2:
    # print(i, j)
    p,s = AsianSim(S0, K, T, r, sigma, i, j, security_type = 'call')
    price_simulation.append(p)
    Stand_Error.append(s)

results_p = pd.DataFrame(np.array(price_simulation).reshape((6,6)), columns = M_2, index = N_2)
results_p
```

```python
# Plotting
plt.figure(figsize=(10, 6))
for n in results_p.index:
    plt.plot(results_p.columns, results_p.loc[n, :], label=f'Time Steps: {n}')

plt.title('Option Prices vs. Number of Simulations')
plt.xlabel('Number of Simulations')
plt.ylabel('Option Price')
plt.legend(title='Number of Time Steps')
plt.grid(True)
plt.xscale('log')  # Optional: Logarithmic scale for better visualization if needed
plt.show()
```

```python
results_s=pd.DataFrame(np.array(Stand_Error).reshape((6,6)), columns = M_2, index = N_2)
results_s
```

```python
# Plotting
plt.figure(figsize=(10, 6))
for n in results_s.index:
    plt.plot(results_s.columns, results_s.loc[n, :], label=f'Time Steps: {n}')

plt.title('Standard Error vs. Number of Simulations')
plt.xlabel('Number of Simulations')
plt.ylabel('Standard Error')
plt.legend(title='Number of Time Steps')
plt.grid(True)
plt.xscale('log')  # Optional: Logarithmic scale for better visualization if needed
plt.show()
```

```python
# Vary volatility
for new_sigma in [0.1, 0.2, 0.3, 0.4]:
    price, std = AsianSim(S0, K, T, r, new_sigma, N, M, 'call')
    print(f"Volatility: {new_sigma}, Estimated Call Price: {price:.4f}")

# Vary risk-free rate
for new_r in [0.01, 0.05, 0.1]:
    price,std = AsianSim(S0, K, T, new_r, sigma, N, M, 'call')
    print(f"Risk-Free Rate: {new_r}, Estimated Call Price: {price:.4f}")

# Vary maturity
for new_T in [0.5, 1.0, 2.0]:
    price,std = AsianSim(S0, K, new_T, r, sigma, N, M, 'call')
    print(f"Maturity: {new_T}, Estimated Call Price: {price:.4f}")
```

```python
def enhanced_asian_option_price(S0, K, T, r, sigma, N, M, option_type='call'):
    dt = T / N
    paths = np.zeros((M, N+1))
    paths[:, 0] = S0

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

    # Calculate the arithmetic average
    arithmetic_avg = np.mean(paths[:, 1:], axis=1)

    # Calculate the geometric average
    geometric_avg = np.exp(np.mean(np.log(paths[:, 1:]), axis=1))

    # Calculate payoffs for both averages
    arithmetic_payoffs = np.maximum(arithmetic_avg - K, 0) if option_type == 'call' else np.maximum(K - arithmetic_avg, 0)
    geometric_payoffs = np.maximum(geometric_avg - K, 0) if option_type == 'call' else np.maximum(K - geometric_avg, 0)

    # Analytical price for geometric Asian options (as control)
    # Placeholder for actual analytical formula here
    c = np.mean(geometric_payoffs)

    # Using control variates
    beta = np.cov(arithmetic_payoffs, geometric_payoffs)[0, 1] / np.var(geometric_payoffs)
    enhanced_payoffs = arithmetic_payoffs - beta * (geometric_payoffs - c)

    # Discount and average the payoffs
    option_price = np.exp(-r * T) * np.mean(enhanced_payoffs)

    std_error_enhanced = np.std(np.exp(-r * T) * enhanced_payoffs) / np.sqrt(M)


    return option_price,np.corrcoef(arithmetic_payoffs, geometric_payoffs)[0, 1],std_error_enhanced

# Pricing the option with control variates
enhanced_price,coeff,std_error_enhanced = enhanced_asian_option_price(S0, K, T, r, sigma, N, M)
print(f"Estimated Asian Call Option Price (Enhanced with Control Variates): {enhanced_price:.4f}")
print(f"Estimated Asian Call Coefficient (Enhanced with Control Variates): {coeff:.4f}")
print(f"Estimated Asian Call Standard Error (Enhanced with Control Variates): {std_error_enhanced:.4f}")

# Parameters
S0 = S0_xsp       # initial stock price
K = 510         # strike price
T =  time_to_maturity        # time to maturity in years
r = 0.04031        # risk-free rate
sigma = 0.16668     # volatility
N = 500        # number of time steps
M = 10000       # number of simulations
```

```python
# Replications levels
replications = [1000, 10000, 100000, 1000000]

results = []
for M in replications:
    price, corff, error = enhanced_asian_option_price(S0, K, T, r, sigma, N, M,'call')
    results.append((M, price, error))
# Display results
print("Number of Paths | Estimated Enhanced Price | Enhanced Standard Error")
for result in results:
    print(f"{result[0]:>15} | {result[1]:>15.4f} | {result[2]:>14.4f}")
```

```python
# Vary volatility
for new_sigma in [0.1, 0.2, 0.3, 0.4]:
    price, std,coeff = enhanced_asian_option_price(S0, K, T, r, new_sigma, N, M, 'call')
    print(f"Volatility: {new_sigma}, Estimated Enhanced Call Price: {price:.4f}")

# Vary risk-free rate
for new_r in [0.01, 0.05, 0.1]:
    price,std,coeff = enhanced_asian_option_price(S0, K, T, new_r, sigma, N, M, 'call')
    print(f"Risk-Free Rate: {new_r}, Estimated Enhanced Call Price: {price:.4f}")

# Vary maturity
for new_T in [0.5, 1.0, 2.0]:
    price,std,coeff = enhanced_asian_option_price(S0, K, new_T, r, sigma, N, M, 'call')
    print(f"Maturity: {new_T}, Estimated Enhanced Call Price: {price:.4f}")
```

```python
import time
import numpy as np

# Simulation parameters
M_values =1000,10000,100000,500000, 1000000

results = []

for M in M_values:
    # Basic Monte Carlo
    start_time = time.time()
    basic_price, std_error_basic = AsianSim(S0, K, T, r, sigma, N, M, 'call')
    basic_duration = time.time() - start_time

    # Enhanced Monte Carlo with Control Variates
    start_time = time.time()
    enhanced_price, correlation_coeff,std_error_enhanced = enhanced_asian_option_price(S0, K, T, r, sigma, N, M)
    enhanced_duration = time.time() - start_time

    # Collect results
    results.append({
        'M': M,
        'Basic Price': basic_price,
        'Basic Std Error': std_error_basic,
        'Basic Duration': basic_duration,
        'Enhanced Price': enhanced_price,
        'Correlation Coefficient': correlation_coeff,
        'Enhanced Std Error': std_error_enhanced,
        'Enhanced Duration': enhanced_duration
    })

# Display results
print("Replications | Basic Price | Basic Std Error | Basic Time | Enhanced Price | Enhanced Std Error | Enhanced Time | Correlation")
for result in results:
    print(f"{result['M']} | {result['Basic Price']:.4f} | {result['Basic Std Error']:.4f} | {result['Basic Duration']:.4f} s | "
          f"{result['Enhanced Price']:.4f} | {result['Enhanced Std Error']:.4f}| {result['Enhanced Duration']:.4f} s | {result['Correlation Co
```

```python
# Convert results to DataFrame for easier plotting
df = pd.DataFrame(results)
print(df)
```

```python
plt.figure(figsize=(14, 5))
plt.subplot(1, 2, 1)
plt.plot(df['M'], df['Basic Price'], label='Basic Monte Carlo', marker='o')
plt.plot(df['M'], df['Enhanced Price'], label='Enhanced Monte Carlo', marker='o')
plt.xlabel('Number of Simulations (M)')
plt.ylabel('Option Price')
plt.title('Comparison of Option Prices')
plt.legend()
#plt.grid(True)
#plt.show()

plt.subplot(1, 2, 2)
#plt.figure(figsize=(14, 5))
plt.plot(df['M'], df['Basic Std Error'], label='Basic Std Error', marker='o', color='green')
plt.plot(df['M'], df['Enhanced Std Error'], label='Enhanced Std Error', marker='o', color='purple')
plt.xlabel('Number of Simulations (M)')
plt.ylabel('Standard Error')
plt.title('Comparison of Standard Errors')
plt.legend()
plt.grid(True)
plt.show()

plt.tight_layout()
plt.show()
```

```python
plt.figure(figsize=(12, 6))
plt.plot(df['M'], df['Basic Duration'], label='Basic Monte Carlo Time', marker='o', color='red')
plt.plot(df['M'], df['Enhanced Duration'], label='Enhanced Monte Carlo Time', marker='o', color='blue')
plt.xlabel('Number of Simulations (M)')
plt.ylabel('Computation Time (seconds)')
plt.title('Comparison of Computation Times')
plt.legend()
plt.grid(True)
plt.show()
```

```python
plt.figure(figsize=(12, 6))
plt.plot(df['M'], df['Basic Std Error'], label='Basic Std Error', marker='o', color='green')
plt.plot(df['M'], df['Enhanced Std Error'], label='Enhanced Std Error', marker='o', color='purple')
plt.xlabel('Number of Simulations (M)')
plt.ylabel('Standard Error')
plt.title('Comparison of Standard Errors')
plt.legend()
plt.grid(True)
plt.show()
```

```python
import matplotlib.pyplot as plt
from scipy.stats import pearsonr

# Gather data for plotting
correlations = [result['Correlation Coefficient'] for result in results]
std_errors_basic = [result['Basic Std Error'] for result in results]
std_errors_enhanced = [result['Enhanced Std Error'] for result in results]
std_reduction_percentages = [(1 - (e / b)) * 100 for b, e in zip(std_errors_basic, std_errors_enhanced)]


# Plot Correlation vs. Variance Reduction
plt.scatter(correlations, std_reduction_percentages)
plt.xlabel('Correlation Coefficient')
plt.ylabel('Reduction in Standard Error (%)')
plt.title('Correlation vs. Variance Reduction')
plt.grid(True)

plt.tight_layout()
plt.show()

# Statistical correlation
corr_coef, p_value = pearsonr(correlations, std_reduction_percentages)
print(f"Pearson correlation coefficient between correlation and variance reduction: {corr_coef:.4f}")
print(f"P-value: {p_value:.4f}")
```