**Monte Carlo Methods: additional Example**

This repository contains a Python implementation of the Monte Carlo method to price a European basket call option under the Black-Scholes framework. The project showcases various techniques to simulate correlated asset prices, compute option prices, and apply variance reduction strategies.

# Case Scenario

Let us suppose that the price of an asset at time t, evolves, under the risk-neutral measure according to the stochastic differential equation (SDE):

.

With:

* r: risk-free rate,
* : volatility of the i-th asset,
* : increment of a Brownian motion under the risk-neutral measure.

Now the European basket call option payoff at maturity T with strike K is given by

Since no closed-form solution exists, we estimate the option price using Monte Carlo simulation.

# Project Highlights

## Euler Discretization

In this project, we aim to generate a vector representing the d asset prices at maturity T, given the volatility vector and other parameters. To achieve this, we will utilize the Euler discretization method for the stochastic differential equation (SDE) provided. This involves partitioning the interval into time steps with .

The approach follows the same principles applied in our previous project, where we employed numerical methods to simulate stochastic processes and compute derivative prices. This ensures consistency in methodology while extending its application to the pricing of basket options.

**Note 1:** The SDE is driven by a single (common) Brownian motion for all . That is, the assets prices are perfectly correlated as = =

## Monte Carlo Estimation

The code implemented in Figure 2 computes the price of a European Basket call using Monte-Carlo simulation using the vector of simulated asset terminal prices with the function from part a. It is being done in a similar fashion to our previous project.

## Control Variate

We reduce the variance by introducing a variate whose payoff is

Where is proportionally derived from .

Our code will calculate the optimal c that we will be using in our control variate method.

# Key Parameters

- **Number of assets (d)**: 4

- **Initial asset prices ():** [100, 100, 100, 100]

- **Volatilities (σ)**: [0.18, 0.22, 0.28, 0.36]

- **Risk-free rate (r)**: 0.02

- **Strike price (K)**: 425

- **Maturity (T)**: 2

- **Time steps (M)**: 104

- **Paths (N)**: 102, 103, 104, 105

# Code

## PART 1

**import** numpy **as** np

**from** scipy.stats **import** norm *# for part c*

**import** pandas **as** pd *# for better visualization of the results*

*# we will initialize a list to store results for the efficiency part in a dataframe*

results = []

np.random.seed(420)

*# Function to simulate asset prices using Euler discretization*

**def** simulate\_asset\_prices(d, T, M, S0, r, sigma):

delta\_t = T / M

S = np.zeros((M + 1, d)) *# Matrix to store asset prices at each step*

S[0] = S0 *# Initial prices*

*# We will generate Brownian increments for each asset*

*# Note that np.random.randn(M,d) gives :matrix of independent standard normal random variables 𝑁(0,1) with M rows (time steps) and d columns (assets).*

dW = np.random.randn(M, d) \* np.sqrt(delta\_t)

*# We will apply the discritization*

**for** t **in** range(M):

S[t + 1] = S[t] + r \* S[t] \* delta\_t + sigma \* S[t] \* dW[t]

**return** S[-1] *# We only need the final prices at time T*

## PART 2

*# Problem Parameters*

volatility = [0.18, 0.22, 0.28, 0.36] *# Volatilities for 4 assets*

T = 2 *# Time to maturity (1 year)*

num\_assets = 4 *# Number of assets*

initial\_prices = [100, 100, 100, 100] *# Initial prices of the assets*

K = 425

risk\_free\_rate = 0.02 *# Risk-free rate (2%)*

Ns = [10\*\*2, 10\*\*3, 10\*\*4, 10\*\*5]

p = 0.1

num\_steps = 104

*# Function for the Monte Carlo estimator of the European basket call option*

**def** monte\_carlo\_basket\_call(S0, K, r, sigma, T, M, d, N):

temp = 0 *# Sum of payoffs*

temp2 = 0 *# Sum of squared payoffs*

**for** \_ **in** range(N):

final\_prices = simulate\_asset\_prices(d, T, M, S0, r, sigma) *# Final prices of all assets*

basket\_payoff = max(np.sum(final\_prices) - K, 0) *# Basket call payoff*

discounted\_payoff = np.exp(-r \* T) \* basket\_payoff *# Discounted payoff*

temp += discounted\_payoff

temp2 += discounted\_payoff\*\*2

*# Monte Carlo estimates*

C0\_CR = temp / N *# Mean payoff*

S2\_CR = (1 / (N - 1)) \* temp2 - (N / (N - 1)) \* C0\_CR\*\*2

Var\_CR = S2\_CR / N *# Adjusted variance*

CI\_CR = [C0\_CR - 1.96 \* np.sqrt(Var\_CR), C0\_CR + 1.96 \* np.sqrt(Var\_CR)] *# 95% Confidence Interval*

**return** C0\_CR, CI\_CR, Var\_CR

## PART 3

*# Function to calculate K adj*

**def** adjusted\_K(K, S0):

K\_adjusted = K \* np.array(S0) / np.sum(S0)

**return** K\_adjusted

*# Function to be used as E[g(Ui)]*

**def** get\_Analytical\_Value(d, T, S0, r, sigma, K):

*# We will adjust the strike prices*

K\_adj = adjusted\_K(K, S0)

*# We will transform into NumPy arrays and not a list*

S0 = np.array(S0)

sigma = np.array(sigma)

*# We will compute d1 and d2 using vectorized operations*

d1\_values = (np.log(S0 / K\_adj) + (r + 0.5 \* sigma\*\*2) \* T) / (sigma \* np.sqrt(T))

d2\_values = d1\_values - sigma \* np.sqrt(T)

*# We will compute the Black-Scholes prices using vectorized operations*

BS\_prices = S0 \* norm.cdf(d1\_values) - K\_adj \* np.exp(-r \* T) \* norm.cdf(d2\_values)

*# Return the sum of all option prices*

**return** np.sum(BS\_prices)

*# Function for the Monte Carlo estimator of the European basket call option with control variates*

**def** monte\_carlo\_basket\_call\_control(S0, K, r, sigma, T, M, d, N):

*# We will compute the adjusted strikes for the control variates*

m = int(p \* N)

M\_CV = N - m

K\_adjusted = adjusted\_K(K, S0)

temp\_muB = 0

temp\_muG = 0

temp\_s2G = 0

disc\_AG = 0

disc\_B\_gT = 0

**for** \_ **in** range(m):

final\_prices = simulate\_asset\_prices(d, T, M, S0, r, sigma)

basket\_payoff = max(np.sum(final\_prices) - K, 0)

gT\_payoff = np.sum(np.maximum(final\_prices - K\_adjusted, 0))

discounted\_payoff = np.exp(-r \* T) \* basket\_payoff

discounted\_gT\_payoff = np.exp(-r \* T) \* gT\_payoff

disc\_B\_gT += discounted\_payoff \* discounted\_gT\_payoff

temp\_muB += discounted\_payoff

temp\_muG += discounted\_gT\_payoff

temp\_s2G += discounted\_gT\_payoff \*\* 2

muB = temp\_muB / m

muG = temp\_muG / m

s2G = (temp\_s2G / (m - 1)) - (m / (m - 1)) \* muG \*\* 2

chat = (disc\_B\_gT - m \* muB \* muG) / ((m - 1) \* s2G) *#optimal chat*

*# Main CV Estimator*

temp\_muCV = 0

temp\_s2CV = 0

c0\_CV\_est = get\_Analytical\_Value(d, T, S0, r, sigma, K) *#true value of the control variate*

**for** \_ **in** range(M\_CV):

final\_prices = simulate\_asset\_prices(d, T, M, S0, r, sigma)

basket\_payoff = max(np.sum(final\_prices) - K, 0)

gT\_payoff = np.sum(np.maximum(final\_prices - K\_adjusted, 0))

discounted\_payoff = np.exp(-r \* T) \* basket\_payoff

discounted\_gT\_payoff = np.exp(-r \* T) \* gT\_payoff

temp\_CV = discounted\_payoff - chat \* (discounted\_gT\_payoff - c0\_CV\_est )

temp\_muCV += temp\_CV

temp\_s2CV += temp\_CV \*\* 2

muCV = temp\_muCV / M\_CV

s2CV = (temp\_s2CV / (M\_CV - 1)) - (M\_CV / (M\_CV - 1)) \* muCV \*\* 2

sCV = np.sqrt(s2CV)

MSE = s2CV / M\_CV

ci\_error = 1.96 \* sCV / np.sqrt(M\_CV)

ci\_lower = muCV - ci\_error

ci\_upper = muCV + ci\_error

CI = [ muCV - 1.96 \* sCV / np.sqrt(M\_CV) , muCV + 1.96 \* sCV / np.sqrt(M\_CV)] *# 95% Confidence Interval*

**return** muCV, CI , MSE

*# We will compute and store results for each N*

**for** N **in** Ns:

*# Crude Monte Carlo*

price\_CR, CI\_CR, mse\_crude = monte\_carlo\_basket\_call(initial\_prices, K, risk\_free\_rate, volatility, T, num\_steps, num\_assets, N)

*# Control Variate Monte Carlo*

price\_CV, CI\_CV, mse\_cv = monte\_carlo\_basket\_call\_control(initial\_prices, K, risk\_free\_rate, volatility, T, num\_steps, num\_assets, N)

*# We will append the result to the list*

results.append({'N': N, 'MSE Crude': mse\_crude, 'MSE CV': mse\_cv})

print(f"N = {N}: Estimated Crude Price = {price\_CR:.4f}, 95% CI = {CI\_CR}, MSE = {mse\_crude:.6f}")

print(f"N = {N}: Estimated Control Variate Price = {price\_CV:.4f}, 95% CI = {CI\_CV}, MSE = {mse\_cv:.6f}")

## N = 100: Estimated Crude Price = 52.1566, 95% CI = [29.82257632736575, 74.49059373182165], MSE = 129.843801

## N = 100: Estimated Control Variate Price = 54.7707, 95% CI = [54.706522971081476, 54.83483639108933], MSE = 0.001071

## N = 1000: Estimated Crude Price = 59.2102, 95% CI = [52.21101230220023, 66.20942874710268], MSE = 12.752217

## N = 1000: Estimated Control Variate Price = 54.7932, 95% CI = [54.780608231979464, 54.80584267900016], MSE = 0.000041

## N = 10000: Estimated Crude Price = 53.7395, 95% CI = [51.6535486083318, 55.82552981876027], MSE = 1.132694

## N = 10000: Estimated Control Variate Price = 54.7823, 95% CI = [54.77771931182483, 54.78680571394059], MSE = 0.000005

## N = 100000: Estimated Crude Price = 54.5511, 95% CI = [53.89648795168591, 55.20572064910513], MSE = 0.111548

## N = 100000: Estimated Control Variate Price = 54.7822, 95% CI = [54.780749778314664, 54.783681268182015], MSE = 0.000001

*# we will create a dataframe to showcase the MSE*

df\_results = pd.DataFrame(results)

*# Ensure all columns of the dataframe are displayed*

pd.set\_option('display.max\_columns', **None**)

*# We will add the efficiency component*

df\_results['Efficiency Crude'] = 1 / df\_results['MSE Crude']

df\_results['Efficiency CV'] = 1 / df\_results['MSE CV']

df\_results['Efficiency Improvement'] = df\_results['Efficiency CV'] / df\_results['Efficiency Crude'] - 1

*# Now we display the end result*

print("\nUpdated Dataframe with Efficiency Columns:")

##

## Updated Dataframe with Efficiency Columns:

print(df\_results)

## N MSE Crude MSE CV Efficiency Crude Efficiency CV \

## 0 100 129.843801 1.071450e-03 0.007702 9.333144e+02

## 1 1000 12.752217 4.143959e-05 0.078418 2.413151e+04

## 2 10000 1.132694 5.372937e-06 0.882851 1.861179e+05

## 3 100000 0.111548 5.592483e-07 8.964755 1.788115e+06

##

## Efficiency Improvement

## 0 121184.090672

## 1 307729.281695

## 2 210813.647552

## 3 199459.494465

## Results

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **N (Paths)** | **Method** | **Estimated Price** | **MSE** | **95% Confidence Interval** | **Efficiency Improvement** |
| **100** | CMC | 59.2102 | 12.752 | [52.211, 66.209] | - |
|  | CV | 54.7707 | 0.0011 | [54.707, 54.835] | 121,184 |
| **1,000** | CMC | 56.8423 | 4.236 | [55.212, 58.472] | - |
|  | CV | 54.7932 | 0.000041 | [54.781, 54.806] | 307,729 |
| **10,000** | CMC | 53.7395 | 0.752 | [51.654, 55.826] | - |
|  | CV | 54.781 | 3.9E-06 | [54.779, 54.783] | 210,814 |
| **100,000** | CMC | 54.5511 | 0.1115 | [53.896, 55.206] | - |
|  | CV | 54.7822 | 0.000001 | [54.781, 54.784] | 199,459 |

The crude Monte Carlo method yields reasonable estimates of the option price as NNN increases, but the results demonstrate significant variance for small values of NNN. For instance:

* For , the estimated price is with a wide confidence interval and a large MSE of . This indicates high uncertainty in the estimate.
* Even for , the crude Monte Carlo method yields an estimated price of 53.7395 with a still-wide confidence [51.654, 55.826], reflecting persistent variance.
* As , the crude Monte Carlo estimate stabilizes with a price of 54.5511 , narrower confidence intervals [53.896, 55.206] , and reduced MSE (0.1115), showing improved accuracy but still lagging behind the control variate approach.

**Control Variate Results**

The control variate method dramatically enhances the precision of the estimates:

* For , the estimated price is 54.7707 with a narrow confidence interval [[54.707, 54.835] , and a very low MSE of 0.0011.
* For , the control variate estimate improves further, yielding 54.7932, with a highly precise confidence interval [54.781, 54.806] and a significantly smaller MSE of 0.000041.
* Even at , the control variate consistently produces precise estimates. The price converges 54.7822 with extremely tight confidence intervals [54.781, 54.784] and a negligible MSE of 0.000001.

**Efficiency Comparison**

The efficiency of the control variate method compared to the crude Monte Carlo estimator is evident from the metrics:

* For the efficiency improvement is over , reflecting the drastic reduction in variance.
* For , the efficiency improvement grows to 30M% demonstrating the method's reliability even with moderate sample sizes.
* At and , the efficiency improvements are over 21M% and , 19M% respectively, highlighting the control variate method's scalability and robustness.

The control variate’s superiority stems from leveraging the high correlation between the option payoff and the chosen control variate, which substantially reduces the variance.

**Conclusion**

The control variate method proves to be a highly effective variance reduction technique, when applicable and the use of it comprehensible, yielding accurate and precise option price estimates even for smaller sample sizes. While crude Monte Carlo estimates converge with larger N, the control variate achieves similar or better results with far fewer paths, underscoring its practical advantage in computational finance.