# Monte Carlo Variance Reduction Techniques
# Delta, Gamma, and Antithetic Variates

Usama Buttar

December 18, 2024

**Abstract**

Monte Carlo simulations are a powerful tool for pricing options with complex payoffs or high-dimensionality, but they often require a large number of simulations to achieve a sufficiently accurate estimate. This need for numerous simulations can be computationally expensive, which is where variance reduction techniques come into play. These methods work by lowering the variability of simulation results, much like how a hedged option portfolio reduces variance compared to its unhedged counterpart, thereby improving the efficiency of the estimation process. This paper explores three variance reduction methods—antithetic variates, Delta-based control variates, and Gamma-based control variates—in the context of European option pricing. Through a case study of SPY (S&P 500 ETF) options, we compare the effectiveness of these techniques and analyze the trade-offs between computational efficiency and variance reduction. The mathematical notation and examples are drawn from Implementing Derivatives Models by Les Clewlow and Chris Strickland, which provides a solid foundation for understanding the theoretical background of these methods. By examining the application of these variance reduction techniques in the context of option pricing, this study aims to highlight how they can significantly enhance the accuracy of Monte Carlo simulations without a proportional increase in computational cost.

## 1 Introduction

Monte Carlo methods are indispensable tools in financial engineering for simulating the stochastic behavior of assets. Despite their versatility, a significant drawback is the inherent variance in estimates, necessitating a large number of simulations for accurate results. Variance reduction techniques aim to address this limitation, enabling higher accuracy with fewer simulations.

This document focuses on the following variance reduction methods:

- **Antithetic Variates**: This method uses negatively correlated asset paths to reduce the variance of estimates.
- **Delta-based Control Variates**: Leveraging the linear relationship between the Delta of an option and its payoff, this technique adjusts estimates for improved precision.
- **Gamma-based Control Variates**: Extending the control variate concept to second-order sensitivities, Gamma-based techniques provide additional refinements.

The study demonstrates the implementation of these techniques, evaluates their performance using a case study of SPY (S&P 500 ETF) options, and discusses the trade-offs between computational time and variance reduction.

# 2 Methodology and Implementation

## 2.1 Parameters and Setup

The parameters used in our Monte Carlo simulations are consistent with a standard European call option framework under the Black-Scholes model. These parameters were chosen to illustrate the effectiveness of variance reduction techniques in a realistic setting. The implementation assumes geometric Brownian motion for the underlying asset price, as defined by the stochastic differential equation:

$$\frac{dS_t}{S_t} = r\,dt + \sigma\,dW_t$$

Where:

- $S_t$ is the asset price at time $t$
- $r$ is the risk-free rate
- σis the volatility
- $W_t$ is a Wiener process

[1]:
```
# Import necessary libraries
import time
import datetime
import numpy as np
import pandas as pd
import scipy.stats as stats

# Define initial parameters for the derivative
S = 606.76            # Current stock price
K = 607               # Strike price of the option
vol = 0.113           # Volatility of the stock (annualized)
r = 0.0443            # Risk-free interest rate (annualized)
N = 10                # Number of time steps in the simulation
M = 1000              # Number of Monte Carlo simulations
market_value = 11.44  # Observed market price of the option

# Time to maturity (in years)
T = ((datetime.date(2025, 1, 31) - datetime.date(2024, 12, 17)).days + 1) / 365
```

## 2.2 Baseline Monte Carlo Simulation

The baseline approach uses a straightforward Monte Carlo simulation to estimate the option price without any variance reduction techniques. The payoff of a European call option is given by:

$$\text{Payoff} = \max(S_T - K, 0)$$

The option price is then calculated as the discounted expected payoff:

$$C = e^{-rT}\mathbb{E}[\text{Payoff}]$$

This serves as the reference point for evaluating the variance reduction methods.

[2]:
```
start_time = time.time()

#precompute constants
dt = T/N
nudt = (r - 0.5*vol**2)*dt
volsdt = vol*np.sqrt(dt)
lnS = np.log(S)

# Monte Carlo Method
Z = np.random.normal(size=(N, M))
delta_lnSt = nudt + volsdt*Z
lnSt = lnS + np.cumsum(delta_lnSt, axis=0)
lnSt = np.concatenate( (np.full(shape=(1, M), fill_value=lnS), lnSt ) )

# Compute Expectation and SE
ST = np.exp(lnSt)
CT = np.maximum(0, ST - K)
C0_se = np.exp(-r*T)*np.sum(CT[-1])/M

sigma = np.sqrt( np.sum( (np.exp(-r*T)*CT[-1] - C0_se)**2) / (M-1) )
SE_se = sigma/np.sqrt(M)

mc_time_se = time.time() - start_time

print("Sample Estimate: Call value is ${0} with SE +/- {1}".format(np.
 ↪round(C0_se,2),np.round(SE_se,2)))
print("Computation time is: ", round(mc_time_se,4))
```

```
Sample Estimate: Call value is $11.47 with SE +/- 0.49
Computation time is:  0.0044
```

# 3 Variance Reduction Techniques

## 3.1 Antithetic Variates

Antithetic variates reduce variance by leveraging the negative correlation between simulated paths and their antithetic counterparts. For a given path $S_t$, an antithetic path $S'_t$ is generated using:

$$S_{t+\Delta t} = S_t \exp\left(\nu \Delta t + \sigma(z_{t+\Delta t} - z_t)\right)$$

Where

$$(z_{t+\Delta t} - z_t) \sim N(0, \Delta t) \sim \sqrt{\Delta t} N(0,1) \sim \sqrt{\Delta t}\epsilon_i$$

**Contract Simulation** The simulated payoffs for the original and antithetic paths are:

$$C_{T,i} = \max(0, S\exp(\nu \Delta T + \sigma\sqrt{T}\epsilon_i) - K)$$
$$\bar{C}_{T,i} = \max(0, S\exp(\nu \Delta T + \sigma\sqrt{T}(-\epsilon_i)) - K)$$

The final estimate is the average of payoffs from both paths:

$$\text{Estimate} = \frac{1}{2}\left(\mathbb{E}[\text{Payoff}] + \mathbb{E}[\text{Payoff from antithetic path}]\right)$$

[3]:
```
start_time = time.time()

#precompute constants
dt = T/N
nudt = (r - 0.5*vol**2)*dt
volsdt = vol*np.sqrt(dt)
lnS = np.log(S)

# Monte Carlo Method
Z = np.random.normal(size=(N, M))
delta_lnSt1 = nudt + volsdt*Z
delta_lnSt2 = nudt - volsdt*Z
lnSt1 = lnS + np.cumsum(delta_lnSt1, axis=0)
lnSt2 = lnS + np.cumsum(delta_lnSt2, axis=0)

# Compute Expectation and SE
ST1 = np.exp(lnSt1)
ST2 = np.exp(lnSt2)
CT = 0.5 * ( np.maximum(0, ST1[-1] - K) + np.maximum(0, ST2[-1] - K) )
C0_av = np.exp(-r*T)*np.sum(CT)/M

sigma = np.sqrt( np.sum( (np.exp(-r*T)*CT - C0_av)**2) / (M-1) )
SE_av = sigma/np.sqrt(M)
```

```
mc_time_av = time.time() - start_time

print("Call value is ${0} with SE +/- {1}".format(np.round(C0_av,2),np.
 ↪round(SE_av,2)))
print("Computation time is: ", round(mc_time_av,4))
```

```
Call value is $11.09 with SE +/- 0.24
Computation time is:  0.0052
```

### 3.2 Delta-Based Control Variates

The Delta-based control variate method adjusts the simulation outcomes by exploiting the linear relationship between the option Delta and its payoff. The adjustment formula is:

$$cv_1 = \sum_{i=0}^{N-1} \frac{\delta C_{t_i}}{\delta S}(S_{t_{i+1}} - \mathbb{E}[S_{t_i}]) \exp(r(T - t_{i+1}))$$

$$C_{t_0} \exp(rT) = C_T + \beta_1 cv_1 + \eta$$

Where:

- $\mathbb{E}[S_{t_{i+1}}] = S_{t_i} \exp(r\Delta t_i)$
- $\beta_1 = -1$, the appropriate value for exact Delta for European options

[4]:
```
def delta_calc(r, S, K, T, sigma, type="c"):
    "Calculate delta of an option"
    d1 = (np.log(S/K) + (r + sigma**2/2)*T)/(sigma*np.sqrt(T))
    try:
        if type == "c":
            delta_calc = stats.norm.cdf(d1, 0, 1)
        elif type == "p":
            delta_calc = -stats.norm.cdf(-d1, 0, 1)
        return delta_calc
    except:
        print("Please confirm option type, either 'c' for Call or 'p' for Put!")

start_time = time.time()

#precompute constants
dt = T/N
nudt = (r - 0.5*vol**2)*dt
volsdt = vol*np.sqrt(dt)

erdt = np.exp(r*dt)
cv = 0
beta1 = -1
```

```
# Monte Carlo Method
Z = np.random.normal(size=(N, M))
delta_St = nudt + volsdt*Z
ST = S*np.cumprod( np.exp(delta_St), axis=0)
ST = np.concatenate( (np.full(shape=(1, M), fill_value=S), ST ) )
deltaSt = delta_calc(r, ST[:-1].T, K, np.linspace(T,dt,N), vol, "c").T
cv = np.cumsum(deltaSt*(ST[1:] - ST[:-1]*erdt), axis=0)


CT = np.maximum(0, ST[-1] - K) + beta1*cv[-1]
C0_dv = np.exp(-r*T)*np.sum(CT)/M

sigma = np.sqrt( np.sum( (np.exp(-r*T)*CT - C0_dv)**2) / (M-1) )
sigma = np.std(np.exp(-r*T)*CT)
SE_dv = sigma/np.sqrt(M)

mc_time_dv = time.time() - start_time

print("Call value is ${0} with SE +/- {1}".format(np.round(C0_dv,2),np.
 ↪round(SE_dv,3)))
print("Computation time is: ", round(mc_time_dv,4))
```

```
Call value is $11.33 with SE +/- 0.081
Computation time is:  0.0083
```

### 3.3 Gamma-Based Control Variates

Gamma-based control variates extend the concept of control variates to second-order sensitivities. The formula for the Gamma-based adjustment is:

$$cv_2 = \sum_{i=0}^{N-1} \frac{\delta^2 C_{t_i}}{\delta S^2} \left( (\Delta S_{t_{i+1}})^2 - \mathbb{E}[(\Delta S_{t_i})^2] \right) \exp(r(T - t_{i+1}))$$

Where:

$$\mathbb{E}[(\Delta S_{t_i})^2] = S_{t_i}^2 \left( \exp\left( (2r + \sigma^2)\Delta t_i \right) - 2\exp(r\Delta t_i) + 1 \right)$$

This method accounts for the curvature of the option price with respect to changes in the underlying asset, providing additional variance reduction.

[5]:
```
def gamma_calc(r, S, K, T, sigma):
    "Calculate delta of an option"
    d1 = (np.log(S/K) + (r + sigma**2/2)*T)/(sigma*np.sqrt(T))
    try:
        gamma_calc = stats.norm.pdf(d1, 0, 1)/(S*sigma*np.sqrt(T))
        return gamma_calc
    except:
        print("Please confirm option type, either 'c' for Call or 'p' for Put!")
```

```
start_time = time.time()

#precompute constants
dt = T/N
nudt = (r - 0.5*vol**2)*dt
volsdt = vol*np.sqrt(dt)
erdt = np.exp(r*dt)
ergamma = np.exp((2*r+vol**2)*dt) - 2*erdt + 1
beta2 = -0.5

# Monte Carlo Method
Z = np.random.normal(size=(N, M))
delta_St = nudt + volsdt*Z
ST = S*np.cumprod( np.exp(delta_St), axis=0)
ST = np.concatenate( (np.full(shape=(1, M), fill_value=S), ST ) )
gammaSt = gamma_calc(r, ST[:-1].T, K, np.linspace(T,dt,N), vol).T
cv2 = np.cumsum(gammaSt*((ST[1:] - ST[:-1])**2 - ergamma*ST[:-1]**2), axis=0)


CT = np.maximum(0, ST[-1] - K) + beta2*cv2[-1]
C0_gv = np.exp(-r*T)*np.sum(CT)/M

sigma = np.sqrt( np.sum( (np.exp(-r*T)*CT - C0_gv)**2) / (M-1) )
sigma = np.std(np.exp(-r*T)*CT)
SE_gv = sigma/np.sqrt(M)

mc_time_gv = time.time() - start_time

print("Call value is ${0} with SE +/- {1}".format(np.round(C0_gv,2),np.
 ↪round(SE_gv,3)))
print("Computation time is: ", round(mc_time_gv,4))
```

```
Call value is $11.45 with SE +/- 0.469
Computation time is:  0.0065
```

### 3.4 Combined Antithetic and Delta Variates

To further reduce variance, the antithetic variates technique can be combined with Delta-based control variates. The adjusted payoff becomes:

$$C_T = 0.5 \left( \max(0, S_{1,t} - K) + \max(0, S_{2,t} - K) + \beta_1 cv_1 \right)$$

Where $cv_1$ now accounts for antithetic variates:

$$cv_1 = 0.5 \cdot \beta_1 \cdot (cv_{11} + cv_{12})$$

With:

$$cv_{11} = \Delta S_{1,t} \left( S_{1,t_{i+1}} - S_{1,t_i} \exp(r \Delta t_i) \right)$$

$$cv_{12} = \Delta S_{2,t} \left( S_{2,t_{i+1}} - S_{2,t_i} \exp(r \Delta t_i) \right)$$

[6]:
```
start_time = time.time()

#precompute constants
dt = T/N
nudt = (r - 0.5*vol**2)*dt
volsdt = vol*np.sqrt(dt)
erdt = np.exp(r*dt)
beta1 = -1

# Monte Carlo Method
Z = np.random.normal(size=(N, M))
delta_St1 = nudt + volsdt*Z
delta_St2 = nudt - volsdt*Z
ST1 = S*np.cumprod( np.exp(delta_St1), axis=0)
ST2 = S*np.cumprod( np.exp(delta_St2), axis=0)
ST1 = np.concatenate( (np.full(shape=(1, M), fill_value=S), ST1 ) )
ST2 = np.concatenate( (np.full(shape=(1, M), fill_value=S), ST2 ) )

# Calculate delta for both sets of underlying stock prices
deltaSt1 = delta_calc(r, ST1[:-1].T, K, np.linspace(T,dt,N), vol, "c").T
deltaSt2 = delta_calc(r, ST2[:-1].T, K, np.linspace(T,dt,N), vol, "c").T

# Calculate two sets of delta control variates for negatively correlated assets
cv11 = np.cumsum(deltaSt1*(ST1[1:] - ST1[:-1]*erdt), axis=0)
cv12 = np.cumsum(deltaSt2*(ST2[1:] - ST2[:-1]*erdt), axis=0)

CT = 0.5 * (  np.maximum(0, ST1[-1] - K) + beta1*cv11[-1]
            + np.maximum(0, ST2[-1] - K) + beta1*cv12[-1] )

C0_adv = np.exp(-r*T)*np.sum(CT)/M

sigma = np.sqrt( np.sum( (np.exp(-r*T)*CT - C0_adv)**2) / (M-1) )
sigma = np.std(np.exp(-r*T)*CT)
SE_adv = sigma/np.sqrt(M)

mc_time_adv = time.time() - start_time

print("Call value is ${0} with SE +/- {1}".format(np.round(C0_adv,2),np.
 ↪round(SE_adv,3)))
print("Computation time is: ", round(mc_time_adv,4))
```

```
Call value is $11.36 with SE +/- 0.081
Computation time is:  0.0129
```

### 3.5 Combined Antithetic, Delta, and Gamma Variates

By including Gamma-based control variates in the combined approach, we account for both first- and second-order sensitivities. The final adjusted payoff is given by:

$$C_T = 0.5\left(\max(0, S_{1,t} - K) + \max(0, S_{2,t} - K) + \beta_1 cv_1 + \beta_2 cv_2\right)$$

Where $cv_1$ and $cv_2$ are adjusted for antithetic techniques:

$cv_1 = 0.5 \cdot \beta_1 \cdot (cv_{11} + cv_{12})$

- $cv_{11} = \Delta_{S_{1,t}}[S_{1,t_{i+1}} - S_{1,t_i} \exp(r\Delta t_i)]$
- $cv_{12} = \Delta_{S_{2,t}}[S_{2,t_{i+1}} - S_{2,t_i} \exp(r\Delta t_i)]$

$cv_2 = 0.5 * \beta_2 * (cv_{21} + cv_{22})$

- $cv_{21} = \gamma_{S_{1,t}}[(S_{1,t_{i+1}} - S_{1,t_i})^2 - S_{1,t_i}^2(\exp([2r + \sigma^2]\Delta t_i) - 2\exp(r\Delta t_i) + 1)]$
- $cv_{22} = \gamma_{S_{2,t}}[(S_{2,t_{i+1}} - S_{2,t_i})^2 - S_{2,t_i}^2(\exp([2r + \sigma^2]\Delta t_i) - 2\exp(r\Delta t_i) + 1)]$

[7]:
```
start_time = time.time()

#precompute constants
dt = T/N
nudt = (r - 0.5*vol**2)*dt
volsdt = vol*np.sqrt(dt)
erdt = np.exp(r*dt)
ergamma = np.exp((2*r+vol**2)*dt) - 2*erdt + 1

beta1 = -1
beta2 = -0.5

# Monte Carlo Method
Z = np.random.normal(size=(N, M))
delta_St1 = nudt + volsdt*Z
delta_St2 = nudt - volsdt*Z
ST1 = S*np.cumprod( np.exp(delta_St1), axis=0)
ST2 = S*np.cumprod( np.exp(delta_St2), axis=0)
ST1 = np.concatenate( (np.full(shape=(1, M), fill_value=S), ST1 ) )
ST2 = np.concatenate( (np.full(shape=(1, M), fill_value=S), ST2 ) )

# Calculate delta for both sets of underlying stock prices
deltaSt1 = delta_calc(r, ST1[:-1].T, K, np.linspace(T,dt,N), vol, "c").T
deltaSt2 = delta_calc(r, ST2[:-1].T, K, np.linspace(T,dt,N), vol, "c").T

# Calculate gamma for both sets of underlying stock prices
gammaSt1 = gamma_calc(r, ST1[:-1].T, K, np.linspace(T,dt,N), vol).T
gammaSt2 = gamma_calc(r, ST2[:-1].T, K, np.linspace(T,dt,N), vol).T
```

```
# Calculate two sets of delta control variates for negatively correlated assets
cv11 = np.cumsum(deltaSt1*(ST1[1:] - ST1[:-1]*erdt), axis=0)
cv12 = np.cumsum(deltaSt2*(ST2[1:] - ST2[:-1]*erdt), axis=0)

# Calculate two sets of gamma control variates for negatively correlated assets
cv21 = np.cumsum(gammaSt1*((ST1[1:] - ST1[:-1])**2 - ergamma*ST1[:-1]**2),␣
 ↪axis=0)
cv22 = np.cumsum(gammaSt2*((ST2[1:] - ST2[:-1])**2 - ergamma*ST2[:-1]**2),␣
 ↪axis=0)

CT = 0.5 * (  np.maximum(0, ST1[-1] - K) + beta1*cv11[-1] + beta2*cv21[-1]
            + np.maximum(0, ST2[-1] - K) + beta1*cv12[-1] + beta2*cv22[-1])

C0_adgv = np.exp(-r*T)*np.sum(CT)/M

sigma = np.sqrt( np.sum( (np.exp(-r*T)*CT - C0_adgv)**2) / (M-1) )
sigma = np.std(np.exp(-r*T)*CT)
SE_adgv = sigma/np.sqrt(M)

mc_time_adgv = time.time() - start_time

print("Call value is ${0} with SE +/- {1}".format(np.round(C0_adgv,2),np.
 ↪round(SE_adgv,3)))
print("Computation time is: ", round(mc_time_adgv,4))
```

```
Call value is $11.34 with SE +/- 0.017
Computation time is:  0.0198
```

## 4 Results

The following table summarizes the results of the Monte Carlo simulations with and without variance reduction techniques:

[8]:
```
se_variates = [SE_se, SE_av, SE_dv, SE_gv, SE_adv, SE_adgv]
se_rd = [round(se,4) for se in se_variates]
se_red = [round(SE_se/se,2) for se in se_variates]

comp_time = [mc_time_se, mc_time_av, mc_time_dv, mc_time_gv, mc_time_adv,␣
 ↪mc_time_adgv]
rel_time = [round(mc_time/mc_time_se,2) for mc_time in comp_time]

data = {'Standard Error (SE)': se_rd,
        'SE Reduction Multiple': se_red,
        'Relative Computation Time': rel_time}

# Creates pandas DataFrame.
df = pd.DataFrame(data, index =['Simple estimate', 'with antithetic variate',
```

```
'with delta-based control variate', 'with gamma-based control variate', 'with␣
↪antithetic and delta variates', 'with all combined variates'])

df
```

[8]:

| | Standard Error | Reduction Multiple | Computation Time |
|---|---|---|---|
| Simple Estimate | 0.4862 | 1.00 | 1.00 |
| Antithetic Variate | 0.2377 | 2.05 | 1.17 |
| Delta-Based Control Variate | 0.0814 | 5.97 | 1.89 |
| Gamma-Based Control Variate | 0.4693 | 1.04 | 1.48 |
| Antithetic & Delta Variates | 0.0814 | 5.98 | 2.94 |
| All Variates Combined | 0.0168 | 28.89 | 4.50 |

### 4.1 Analysis

- **Antithetic Variates**: Provides a simple yet effective method to reduce variance, achieving a reduction multiple of approximately 2x with minimal computational overhead.
- **Delta Control Variates**: Leverages first-order sensitivities to achieve a substantial reduction in variance, significantly outperforming antithetic variates alone.
- **Gamma Control Variates**: While modest when used individually, Gamma-based adjustments enhance results when combined with Delta control.
- **Combined Techniques**: The combination of all three methods results in the most significant variance reduction, with a reduction multiple of 29. However, this comes at a higher computational cost.

### 4.2 Trade-offs

The combined methods offer superior accuracy but require careful consideration of computational efficiency. This trade-off is particularly important in scenarios like high-frequency trading or real-time risk assessment.

## 5 Conclusion

Variance reduction techniques enhance the precision of Monte Carlo simulations, making them invaluable tools for quantitative finance. This study demonstrates the effectiveness of antithetic variates, Delta-based control variates, and Gamma-based control variates in reducing the standard error of European option pricing. Combining these methods yields the most significant improvements, albeit at the cost of higher computation time. The choice of technique depends on the specific requirements for accuracy and computational resources in a given application.

Future work could explore the application of these methods to more complex derivatives, stochastic volatility models, and alternative payoff structures.