INDENG 221 Introduction to Financial Engineering: Homework 8.1

Julius Graf

November 2, 2024

We aim to price a European call option with spot stock price $S_0 = \$100.0$, strike $K = \$100.0$, time to maturity $T = 1.0$ year, risk-free interest rate $r = 6\%$, continuous dividend yield $q = 6\%$ and volatility $\sigma = 35\%$ using Monte Carlo (MC) simulation. To do so, we compare three estimators to the value the Black-Scholes formula provides, to estimate

$$f = \mathrm{e}^{-rT}\,\mathbb{E}^{\mathbb{Q}}[\max(S_T - K, 0)]$$

where $\mathbb{Q}$ is the risk-neutral probability measure. As a reminder, Black-Scholes provides the estimate

$$\hat{f}^{\mathrm{BS}} = S\mathrm{e}^{-qT}N(d_1) - K\mathrm{e}^{-rT}N(d_2)$$

where N is the CDF of $\mathcal{N}(0,1)$ and

$$d_1 = \frac{\ln\left(\frac{S_0}{K}\right) + \left(r - q + \frac{\sigma^2}{2}\right)T}{\sigma\sqrt{T}} \quad \text{and} \quad d_2 = d_1 - \sigma\sqrt{T}.$$

1 Background on the three methods

We use $M = 100$ time steps, and $n = 4000$ simulated stock paths. Let $\delta t = T/M$. We define

$$\hat{S}_T^{(i)} = S_0 \prod_{j=1}^{M} \exp\left(\left(r - q - \frac{\sigma^2}{2}\right)\delta t + \sigma\sqrt{\delta t}\,\varepsilon_j^{(i)}\right)$$

$$\hat{S}_T^{(i)\sharp} = S_0 \prod_{j=1}^{M} \exp\left(\left(r - q - \frac{\sigma^2}{2}\right)\delta t - \sigma\sqrt{\delta t}\,\nu_j^{(i)}\right)$$

both estimators of S_T, where $(\varepsilon_j^{(i)})_{1\le j\le M, 1\le i\le n}$ and $(\nu_j^{(i)})_{1\le j\le M, 1\le i\le n}$ are i.i.d. distributed as $\mathcal{N}(0,1)$. Plain Monte Carlo simulation yields the estimator

$$\hat{f}_n^{\mathrm{MC}} = \frac{1}{n}\sum_{i=1}^{n} \max\left(\hat{S}_T^{(i)} - K, 0\right).$$

The antithetic variate method works similarly and provides the estimator

$$\hat{f}_n^{\mathrm{AV}} = \frac{1}{n}\sum_{i=1}^{n} \frac{1}{2}\left[\max\left(\hat{S}_T^{(i)} - K, 0\right) + \max\left(\hat{S}_T^{(i)\sharp} - K, 0\right)\right]$$

Eventually, the control variate method provides the estimator

$$\hat{f}_n^{\mathrm{CV}} = \frac{1}{n}\sum_{i=1}^{n} \max\left(\hat{S}_T^{(i)} - K, 0\right) - \hat{\beta}_n\left(\hat{S}_T^{(i)} - \mathbb{E}^{\mathbb{Q}}[S_T]\right)$$

as, given that $\mathbb{E}^{\mathbb{Q}}[S_T] = S_0 \mathrm{e}^{(r-q)T}$, we use S_T as a control variate. Here,

$$\hat{\beta}_n = \frac{1}{(n-1)\sigma_{S_T}^2} \sum_{i=1}^{n} \left(\max\left(\hat{S}_T^{(i)} - K, 0 \right) - \hat{f}_n^{\mathrm{MC}} \right) \left(\hat{S}_T^{(i)} - \frac{1}{n} \sum_{j=1}^{n} \hat{S}_T^{(j)} \right)$$

is an estimator of $\beta := \mathrm{cov}(\max(S_T - K, 0), S_T)/\sigma_{S_T}^2$, where $\sigma_{S_T}^2 = S_0^2 \mathrm{e}^{2(r-q)T}(\mathrm{e}^{\sigma^2 T} - 1)$. For each of the estimators, we provide an error estimate, that is given by

$$\forall \varphi \in \{\mathrm{MC}, \mathrm{AV}, \mathrm{CV}\}, \quad \delta \hat{f}_n^{\varphi} = \sqrt{\frac{1}{n(n-1)} \sum_{i=1}^{n} (\hat{f}_n^{\varphi}[i] - \hat{f}_n^{\varphi})^2}$$

where $\hat{f}_n^{\varphi}[i]$ is the i-th term in the sum defining $\hat{f}_n^{\varphi}$.

2 Results

The option price estimations and error estimates for each method are summarized in Table 1. We include the mathematical notation for each estimator and error.

Method	**Estimation**	**Quantity**
Plain Monte Carlo	13.08	$\hat{f}_n^{\mathrm{MC}}$
Antithetic Variate	13.12	$\hat{f}_n^{\mathrm{AV}}$
Control Variate	13.17	$\hat{f}_n^{\mathrm{CV}}$
Black-Scholes	13.08	$\hat{f}^{\mathrm{BS}}$

(a) Option Price Estimations by Method

Method	**Error**	**Quantity**
Plain Monte Carlo	0.40	$\delta \hat{f}_n^{\mathrm{MC}}$
Antithetic Variate	0.28	$\delta \hat{f}_n^{\mathrm{AV}}$
Control Variate	0.41	$\delta \hat{f}_n^{\mathrm{CV}}$

(b) Error Estimates by Method

Table 1: Summary of Option Price Estimations and Error Estimates by Method

The associated code can be found in the appendix on the following page.

A Code

```
import numpy as np
from tabulate import tabulate
from scipy.stats import norm

def option_payoff(S, K, option_type):
    if option_type == 'call':
        return np.maximum(S - K, 0)
    elif option_type == 'put':
        return np.maximum(K - S, 0)
    else:
        raise ValueError('Option type must be either "call" or "put".')

def black_scholes(S0, K, option_type, T, r, q, sigma):
    d1 = (np.log(S0 / K) + (r - q + 0.5 * sigma ** 2) * T) / (sigma * np.sqrt(T))
    d2 = d1 - sigma * np.sqrt(T)
    if option_type == 'call':
        return S0 * np.exp(-q * T) * norm.cdf(d1) - K * np.exp(-r * T) * norm.cdf(d2)
    elif option_type == 'put':
        return K * np.exp(-r * T) * norm.cdf(-d2) - S0 * np.exp(-q * T) * norm.cdf(-d1)
    else:
        raise ValueError('Option type must be either "call" or "put".')

def plain_monte_carlo(S0, K, option_type, T, r, q, sigma, n_steps, n_simulation):
    dt = T / n_steps
    sqrt_dt = np.sqrt(dt)
    payoff = np.zeros(n_simulation, dtype=float)
    step = range(0, int(n_steps), 1)
    for i in range(n_simulation):
        S = S0
        for _ in step:
            S *= np.exp((r - q - 0.5 * sigma ** 2) * dt + sigma * np.random.normal() * sqrt_dt)
        payoff[i] = option_payoff(S, K, option_type)
    error_estimate = np.std(payoff) / np.sqrt(n_simulation)
    return np.exp(-r * T) * np.mean(payoff), error_estimate

def antithetic_variate(S0, K, option_type, T, r, q, sigma, n_steps, n_simulation):
    dt = T / n_steps
    sqrt_dt = np.sqrt(dt)
    payoff_down = np.zeros(n_simulation, dtype=float)
    payoff_up = np.zeros(n_simulation, dtype=float)
    step = range(0, int(n_steps), 1)
    for i in range(n_simulation):
        S_up = S0
        S_down = S0
        for _ in step:
            S_up *= np.exp((r - q - 0.5 * sigma ** 2) * dt + sigma * np.random.normal() * sqrt_dt)
            S_down *= np.exp((r - q - 0.5 * sigma ** 2) * dt - sigma * np.random.normal() * sqrt_dt)
        payoff_down[i] = option_payoff(S_up, K, option_type)
        payoff_up[i] = option_payoff(S_down, K, option_type)
    payoff = (payoff_down + payoff_up) / 2
    error_estimate = np.std(payoff) / np.sqrt(n_simulation)
    return np.exp(-r * T) * np.mean(payoff), error_estimate
```

```
def control_variate(S0, K, option_type, T, r, q, sigma, n_steps, n_simulation):
    dt = T / n_steps
    sqrt_dt = np.sqrt(dt)
    payoff_init = np.zeros(n_simulation, dtype=float)
    f = np.zeros(n_simulation, dtype=float)

    step = range(0, int(n_steps), 1)
    for i in range(n_simulation):
        S = S0
        for _ in step:
            S *= np.exp((r - q - 0.5 * sigma ** 2) * dt + sigma * np.random.normal() * sqrt_dt)
        f[i] = S
        payoff_init[i] = option_payoff(S, K, option_type)

    mu = S0 * np.exp((r - q) * T)
    sigma_f = S0 ** 2 * np.exp((2 * r - q) * T) * (np.exp(sigma ** 2 * T) - 1)
    beta_estimate = np.cov(payoff_init, f)[0][1] / (sigma_f * (n_simulation-1))

    payoff = payoff_init - beta_estimate * (S - mu)
    error_estimate = np.std(payoff) / np.sqrt(n_simulation)
    return np.exp(-r * T) * np.mean(payoff), error_estimate

# PARAMETERS
S0 = 100
K = 100
option_type = 'call'
T = 1
r = 0.06
q = 0.06
sigma = 0.35

# SIMULATION PARAMETERS
n_steps = 100
np.random.seed(110124)
n_simulation = 4000

# Store the results in a structured way
estimations = [
    ["Plain Monte Carlo", f"{plain_monte_carlo(S0, K, option_type, T, r, q, sigma,
        n_steps, n_simulation)[0]:.2f}"],
    ["Antithetic Variate", f"{antithetic_variate(S0, K, option_type, T, r, q, sigma,
        n_steps, n_simulation)[0]:.2f}"],
    ["Control Variate", f"{control_variate(S0, K, option_type, T, r, q, sigma,
        n_steps, n_simulation)[0]:.2f}"],
    ["Black-Scholes", f"{black_scholes(S0, K, option_type, T, r, q, sigma):.2f}"]
]

errors = [
    ["Plain Monte Carlo", f"{plain_monte_carlo(S0, K, option_type, T, r, q, sigma,
        n_steps, n_simulation)[1]:.2f}"],
    ["Antithetic Variate", f"{antithetic_variate(S0, K, option_type, T, r, q, sigma,
        n_steps, n_simulation)[1]:.2f}"],
    ["Control Variate", f"{control_variate(S0, K, option_type, T, r, q, sigma,
```

```
        n_steps, n_simulation)[1]:.2f}"]
]

# Print the tables
print("### Option Price Estimations ###")
print(tabulate(estimations, headers=["Method", "Estimation"], tablefmt="grid"))

print("\n### Error Estimates ###")
print(tabulate(errors, headers=["Method", "Error"], tablefmt="grid"))
```

B Output

```
### Option Price Estimations ###
+--------------------+--------------+
| Method             |   Estimation |
+====================+==============+
| Plain Monte Carlo  |        13.08 |
+--------------------+--------------+
| Antithetic Variate |        13.12 |
+--------------------+--------------+
| Control Variate    |        13.17 |
+--------------------+--------------+
| Black-Scholes      |        13.08 |
+--------------------+--------------+

### Error Estimates ###
+--------------------+---------+
| Method             |   Error |
+====================+=========+
| Plain Monte Carlo  |    0.4  |
+--------------------+---------+
| Antithetic Variate |    0.28 |
+--------------------+---------+
| Control Variate    |    0.41 |
+--------------------+---------+
```