Monte Carlo Option Pricing – Notes

by Indi

1 Overview

Monte Carlo simulation is a method that uses random sampling to estimate numerical results for things that are hard to predict. In quantitative finance, it is commonly used to simulate the future evolution of stock prices and to estimate the fair value of financial derivatives such as **options**. An option is a financial contract that allows the user to buy the right (not obligation) to buy the stock at a future price.

Goal

Estimate the price of a European call option(type of option where the user has the right to buy asset on the expiration day) using:

1. Historical stock data (e.g., Apple, ticker AAPL)
2. A stochastic model of price movements (Geometric Brownian Motion). Stochastic means it models a system that varies over time.
3. A large number of simulated price paths
4. Discounting the expected payoff back to the present

2 1. Theoretical Background

2.1 1.1 Stock Price Dynamics

We model stock prices using a **Geometric Brownian Motion (GBM)**:

$$dS_t = \mu S_t dt + \sigma S_t dW_t$$

where:

- S_t = stock price at time t
- μ = drift (expected return)
- σ = volatility (standard deviation of returns)
- dW_t = random shock (Brownian motion term)
- T = Time

2.2 1.0 Brownian Motion

Brownian Motion represents random movement, kind of like how a pollen grain wiggles in water. In finance, it's used to represent the unpredictable fluctuations of asset prices.

Mathematical Notation

In differential form, we write:

$$dW_t \sim \mathcal{N}(0, dt)$$

This means that over a very small time step dt, the change dW_t follows a normal distribution with mean 0 and variance dt.

Definition

A **standard Brownian Motion** W_t is a continuous-time stochastic process that satisfies:

1. $W_0 = 0$ almost surely.
2. It has **independent increments**: for any $0 \leq s < t$, the increment $W_t - W_s$ is independent of all previous values $\{W_u : u \leq s\}$.
3. The increments are **normally distributed**:

   $$W_t - W_s \sim \mathcal{N}(0, t - s)$$

   meaning the mean of the increment is 0, and the variance grows linearly with time.
4. The paths of W_t are continuous but almost surely nowhere differentiable.

2.3 1.1 Stock Price Dynamics

We model stock prices using a **Geometric Brownian Motion (GBM)**:

$$dS_t = \mu S_t dt + \sigma S_t dW_t$$

where:

- S_t = stock price at time t
- μ = drift (expected return)
- σ = volatility (standard deviation of returns)
- dW_t = random shock (Brownian motion term)

- T = Time

In discrete form (for simulation):

$$S_T = S_0 \exp\left((\mu - \frac{1}{2}\sigma^2)T + \sigma\sqrt{T}Z\right)$$

where $Z \sim N(0,1)$ is a standard normal random variable.

2.4 1.2 Option Payoff

For a **European call option**:

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

where K is the strike price and S_T is the simulated price at expiration.

2.5 1.3 Risk-Neutral Valuation

In practice, we use the **risk-neutral drift** r instead of μ. This assumes investors act without to risk, and the expected return equals the risk-free rate.

Hence, the simulated process becomes:

$$S_T = S_0 \exp\left((r - \frac{1}{2}\sigma^2)T + \sigma\sqrt{T}Z\right)$$

The option price is then the discounted expected payoff:

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

3 2. Black–Scholes Formula

Black and Scholes (1973) derived a closed-form formula for European options:

$$C = S_0 N(d_1) - Ke^{-rT}N(d_2)$$

with:

$$d_1 = \frac{\ln(S_0/K) + (r + \frac{1}{2}\sigma^2)T}{\sigma\sqrt{T}}, \qquad d_2 = d_1 - \sigma\sqrt{T}$$

and $N(x)$ is the cumulative distribution function of the standard normal distribution.

This gives a fast, analytical benchmark to compare against Monte Carlo results.

4 3. Implementation Steps in Python

4.1 3.1 Fetch Data

Use `yfinance` to retrieve historical prices:

```
import yfinance as yf
data = yf.download("AAPL", start="2020-01-01", end="2023-01-01")["Close"]
```

4.2 3.2 Estimate Drift and Volatility

Compute daily log returns:

$$r_t = \ln \frac{S_t}{S_{t-1}}$$

and estimate:

$$\mu = \mathrm{mean}(r_t) \times 252, \quad \sigma = \mathrm{std}(r_t) \times \sqrt{252}$$

—

4.3 3.3 Monte Carlo Simulation

```
import numpy as np

S0 = data[-1]
K = 150
T = 1
r = 0.05
simulations = 100000

Z = np.random.normal(0, 1, simulations)
ST = S0 * np.exp((r - 0.5*sigma**2)*T + sigma*np.sqrt(T)*Z)

payoffs = np.maximum(ST - K, 0)
option_price = np.exp(-r*T) * np.mean(payoffs)
```

—

4.4 3.4 Visualization

```
import matplotlib.pyplot as plt
```

```
plt.hist(ST, bins=50, alpha=0.7)
plt.axvline(K, color='red', linestyle='--', label='Strike Price')
plt.title('Simulated Stock Prices at Expiration')
plt.xlabel('Price')
plt.ylabel('Frequency')
plt.legend()
plt.show()
```

—

5 4. Comparison with Black–Scholes

```
from scipy.stats import norm

def black_scholes_call(S, K, T, r, sigma):
    d1 = (np.log(S/K) + (r + 0.5*sigma**2)*T) / (sigma*np.sqrt(T))
    d2 = d1 - sigma*np.sqrt(T)
    return S*norm.cdf(d1) - K*np.exp(-r*T)*norm.cdf(d2)

bs_price = black_scholes_call(S0, K, T, r, sigma)

print("Monte Carlo:", option_price)
print("Black-Scholes:", bs_price)
```

6 5. Key Insights

- **Monte Carlo intuition:** Simulate many random futures, take their average.
- **Law of Large Numbers:** As simulations increase, estimates converge, as the standard deviation decreases
- **Randomness:** Each Z represents a random market movement.
- **Discounting:** Future payoffs must be brought back to today's value using e^{-rT}.
- **Validation:** Always compare your simulated result to the Black–Scholes value, as it's predicted vs calculated
- **Scalability:** Monte Carlo handles complex payoffs where B–S fails, because it acconuts for random, numerous price points over time.