Monte Carlo Simulation using the Geometric Brownian Motion for a Stock

Aniruddha Kabra

September 11, 2025

Briefing

We simulate possible future stock-price scenarios using Monte Carlo methods with the Geometric Brownian Motion (GBM) model. The report summarizes the GBM model, provides the stochastic differential equation and its solution, discusses limitations, and presents simulated path samples.

1 Overview

Monte Carlo simulation is a widely used technique in quantitative finance to explore uncertainty in future outcomes. By repeatedly running the same random experiment under varying conditions, it generates a distribution of possible scenarios rather than a single deterministic forecast.

In this study, we employ the Geometric Brownian Motion (GBM) model to describe the dynamics of stock prices. GBM assumes that asset prices evolve according to two key parameters: drift μ, which captures the average growth rate, and volatility σ, which reflects the level of uncertainty or risk in the returns.

- Stock prices fluctuate randomly from day to day.
- On average, they exhibit a growth trend represented by the drift μ.
- The randomness around this trend is driven by volatility σ.

In simplified form, the GBM update rule can be expressed as

$$\text{Price}_{tomorrow} = \text{Price}_{today} \times \big(\text{expected growth} + \text{random fluctuation from volatility}\big).$$

Formally, GBM is defined through the stochastic differential equation (SDE).

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

where

- S_t: price of the asset at time t,
- μ: drift parameter (expected rate of return),
- σ: volatility parameter,
- W_t: standard Wiener process (Brownian motion).

This formulation provides a mathematically rigorous foundation for simulating stock price dynamics using Monte Carlo methods.

2 Mathematical Model (GBM)

2.1 Stochastic Differential Equation

The GBM model is given by the stochastic differential equation (SDE).

$$dS_t = \mu S_t\, dt + \sigma S_t\, dW_t, \tag{1}$$

where

- S_t: stock price at time t,
- μ: drift parameter,
- σ: volatility parameter,
- W_t: standard Wiener process (Brownian motion).

2.2 Solution to the Stochastic Differential Equation

Applying Itô's lemma to $\ln(S_t)$:

$$d(\ln S_t) = (\mu - \frac{\sigma^2}{2})dt + \sigma W_t \tag{2}$$

Integrating from 0 to T:

$$\ln(\frac{S_t}{S_0}) \sim \mathcal{N}((\mu - \frac{\sigma^2}{2})T, \sigma^2 T) \tag{3}$$

So, the solution is:

$$S_t = S_0 \exp\left(\left(\mu - \tfrac{1}{2}\sigma^2\right) t + \sigma W_t\right). \tag{4}$$

This means that stock prices are log-normally distributed under GBM.

2.3 Estimation of Parameters μ and σ

From daily prices, we compute logarithmic returns:

$$r_t = \ln(\frac{S_t}{S_{t-1}}) \tag{5}$$

Sample mean and variance:

$$\bar{r} = \frac{1}{N}\sum_{t=1}^{N} r_t \qquad s^2 = \frac{1}{N-1}\sum_{t=1}^{N}(r - \bar{r})^2 \tag{6}$$

Annualized volatility estimate:

$$\hat{\sigma} = \sqrt{\frac{s^2}{dt}}\ ,\ dt = \frac{1}{252} \tag{7}$$

Drift estimate:

$$\hat{\mu} = \frac{\bar{r}}{dt} + \frac{\hat{\sigma}^2}{2} \tag{8}$$

2.4 Confidence Intervals

For drift (μ), approximate using the normal distribution of the mean:

$$\mu \in [\hat{\mu} - z_{0.975}\frac{\hat{\sigma}}{\sqrt{Ndt}}, \hat{\mu} + z_{0.975}\frac{\hat{\sigma}}{\sqrt{Ndt}}] \tag{9}$$

For volatility (σ), use the chi-square distribution of the sample variance:

$$\frac{(N-1)s^2}{\sigma^2} \sim \chi^2_{N-1} \tag{10}$$

3 Python Code and Implementation

Importing essential libraries

```
import yfinance as yf
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from datetime import datetime, timedelta
from scipy.stats import norm, chi2
```

yfinance

- Yahoo! Finance API wrapper.
- Downloads financial data: stock prices, historical data, dividends.
- Used for backtesting trading strategies and financial simulations.

numpy (np)

- Numerical Python for arrays, matrices, and mathematical functions.
- Used for numerical computations, simulations, and linear algebra.

pandas (pd)

- Data manipulation and analysis library.
- Provides DataFrames and Series for tabular data.
- Ideal for cleaning, transforming, and analyzing time series.

matplotlib.pyplot (plt)

- Data visualization library.
- Creates plots, charts, and graphs (line plots, histograms, scatter plots).
- Often used with NumPy or Pandas for numerical or time series visualization.

datetime

- Standard Python library for dates and times.
- Includes `datetime` and `timedelta` for arithmetic, comparisons, and formatting.

scipy.stats (norm, chi2)

- Part of SciPy for scientific computing.
- Provides statistical distributions, tests, and functions.
- `norm`: normal (Gaussian) distribution.
- `chi2`: chi-square distribution for hypothesis testing and confidence intervals.

Download Stock Data

```
ticker = "HDFCBANK.NS"
enddate = datetime.today()
startdate = enddate - timedelta(days=3*365)
df = yf.download(ticker, start=startdate.strftime("%Y-%m-%d"),
                 end=enddate.strftime("%Y-%m-%d"))

if "Adj Close" in df.columns:
    prices = df["Adj Close"].dropna()
else:
    prices = df["Close"].dropna()

S0 = float(prices.iloc[-1])
```

- Downloads 3 years of daily historical data for the stock from Yahoo Finance.
- Uses `Adj Close` if available (adjusted for splits/dividends), otherwise `Close`.
- `S0` is the last observed stock price, which is the starting point for the simulation.

Estimate drift (μ) and volatility (σ) with confidence intervals

```
def estimate_params(prices, trading_days=252):
    logp = np.log(prices)
    r = logp.diff().dropna()
    N = len(r)
    dt = 1.0 / trading_days

    r_mean = r.mean()
    s2 = r.var(ddof=1)
    sigma_hat = np.sqrt(s2 / dt)
    mu_hat = (r_mean / dt) + 0.5 * sigma_hat ** 2

    # Confidence intervals
    se_mu = sigma_hat / np.sqrt(N * dt)
    z = norm.ppf(0.975)
    mu_ci = (mu_hat - z * se_mu, mu_hat + z * se_mu)

    chi_low = chi2.ppf(0.025, df=N - 1)
    chi_high = chi2.ppf(0.975, df=N - 1)
    sigma_ci = (
        np.sqrt((N - 1) * s2 / chi_high / dt),
        np.sqrt((N - 1) * s2 / chi_low / dt)
    )

    return float(mu_hat), float(sigma_hat), mu_ci, sigma_ci

mu, sigma, mu_ci, sigma_ci = estimate_params(prices)
```

- `logp = np.log(prices)` takes the natural log of the price of the stock.
- `r = log.diff()` calculates daily log returns $r_t = \ln(S_t/S_{t-1})$.

- `dt = 1/252` is the fraction of a year for daily data assuming 252 trading days.
- `r_mean` and `s2` are the sample mean and variance of daily log returns.
- `sigma_hat = np.sqrt(s2/dt)` is the annualized volatility (standard deviation of returns).
- `mu_hat = (r_mean / dt) + 0.5 * sigma_hat ** 2` is the annualized drift. The `(0.5 * sigma_hat ** 2)` comes from the GBM model.
- `mu_ci` gives 95% confidence interval for drift (assumes normal distribution).
- `sigma_ci` gives 95% confidence for volatility (chi-square distribution used for variation estimation).

Monte Carlo Simulation (Geometric Brownian Motion)

```
def simulate_gbm(S0, mu, sigma, T=1.0, trading_days=252, n_sims=1000):
    np.random.seed(69)
    dt = T / trading_days
    Z = np.random.normal(size=(n_sims, trading_days))
    increments = (mu - 0.5 * sigma**2) * dt + sigma * np.sqrt(dt) * Z
    log_paths = np.cumsum(increments, axis=1)
    log_paths = np.hstack([np.zeros((n_sims, 1)), log_paths])
    return S0 * np.exp(log_paths)

paths = simulate_gbm(S0, mu, sigma, n_sims=1000)
ending_prices = paths[:, -1]
```

- Simulates future stock price paths using Geometric Brownian Motion(GBM).
- `T = 1.0` implies that the time horizon is 1 year.
- `z` is the random standard normal number for stochastic simulation.
- `increments = (mu - 0.5 sigma**2)dt + sigma * sqrt(dt) * z` is the GBM log return formula.
- `log_paths = np.cumsum(increments)` is the cumulative sum for log price path.
- `S0 * exp(log_paths)` convers log-paths back to price paths.

This gives `n_sims` future price trajectories.

Statistics derived from the GBM Model

```
print(f"Drift (mu): {mu:.4f}")
print(f"95% CI for mu: ({float(mu_ci[0]):.4f}, {float(mu_ci[1]):.4f})")
print(f"Volatility (sigma): {sigma:.4f}")
print(f"95% CI for sigma:({float(sigma_ci[0]):.4f},{float(sigma_ci[1]):.4f}
                                                              )\n")

print(f"Initial Price S0: {S0:.2f}")
print(f"Expected Price (mean): {ending_prices.mean():.2f}")
print(f"Median Price: {np.median(ending_prices):.2f}")
p2_5, p97_5 = np.percentile(ending_prices, [2.5, 97.5])
print(f"95% Simulation Interval: [{p2_5:.2f}, {p97_5:.2f}]")
print(f"Probability S_T > S0: {np.mean(ending_prices > S0):.2%}")
```

4 Visualization of Monte Carlo Simulation

To visualize Monte Carlo simulation, let us draw sample paths (S_t) for HDFC Bank stock.

Python Code for sample GBM paths

```
plt.figure(figsize=(10,6), dpi =200)
plt.figure
trading_days = 252
T = 1.0
time_grid = np.linspace(0, T, trading_days + 1)  # from 0 to 1 year

n_plot = 35
for i in range(n_plot):
    plt.plot(time_grid, paths[i], lw=1, alpha=0.7)
plt.xlabel("Time (Years)")
plt.ylabel("Price (INR)")
plt.grid(True, linestyle="--", alpha=0.6)
plt.show()
```

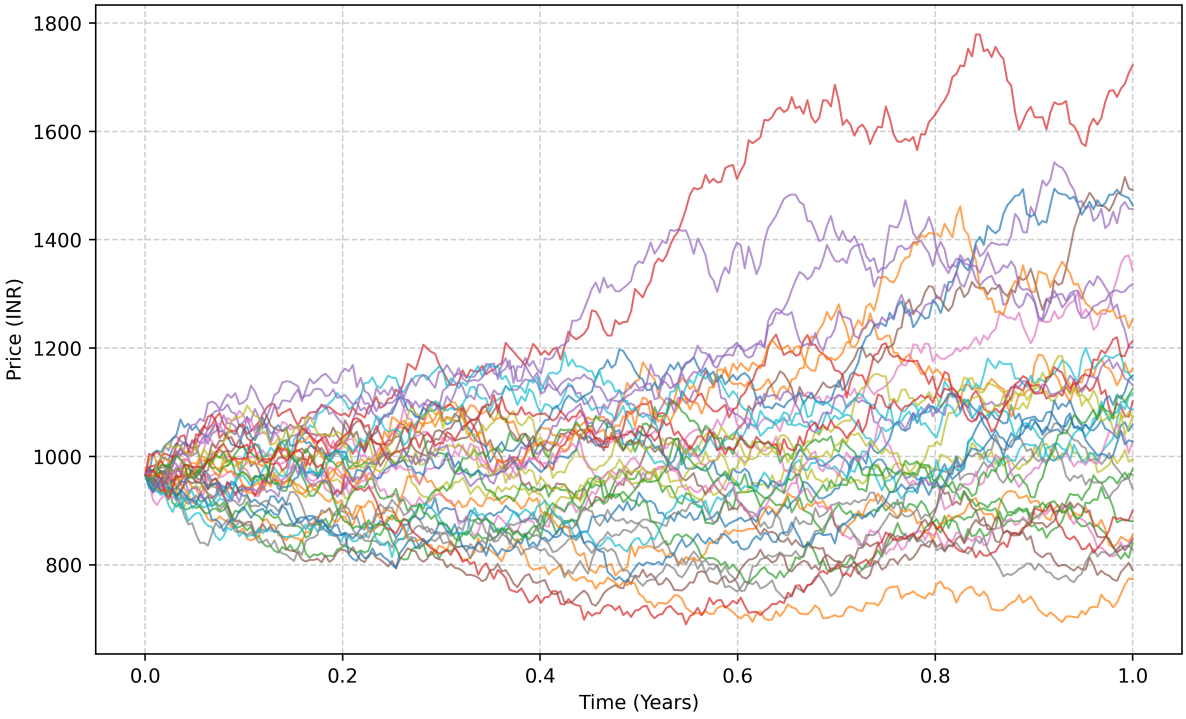

Figure 1: Monte Carlo Simulation of HDFC Bank Stock Price (GBM)

After the above figure, it is evident that the model:

- Uses historical estimation to estimate the drift and volatility. It computes the confidence intervals to quantify uncertainty in the estimates.
- The above graph gives 35 out of 1000 possible price paths for the next year.

Python Code for Histogram

```
plt.figure(figsize=(10,6), dpi=500)
plt.hist(ending_prices, bins=50, alpha=0.7, edgecolor="black")

plt.axvline(S0, color="red", linestyle="--", label=f"Initial Price S0 =
                    {S0:.2f}")
plt.axvline(ending_prices.mean(), color="green", linestyle="--", label=
                        f"Mean = {ending_prices.mean():.2f}")
plt.axvline(np.median(ending_prices), color="blue", linestyle="--", label=
                        f"Median = {np.median(ending_prices):.2f}")

p2_5, p97_5 = np.percentile(ending_prices, [2.5, 97.5])
plt.axvline(p2_5, color="purple", linestyle="--", label=f"2.5% =
                                                    {p2_5:.2f}")
plt.axvline(p97_5, color="purple", linestyle=":", label=f"97.5% =
                                                    {p97_5:.2f}")

plt.xlabel("Price (INR)")
plt.ylabel("Frequency")
plt.legend()
plt.grid(True, linestyle="--", alpha=0.6)
plt.show()
```

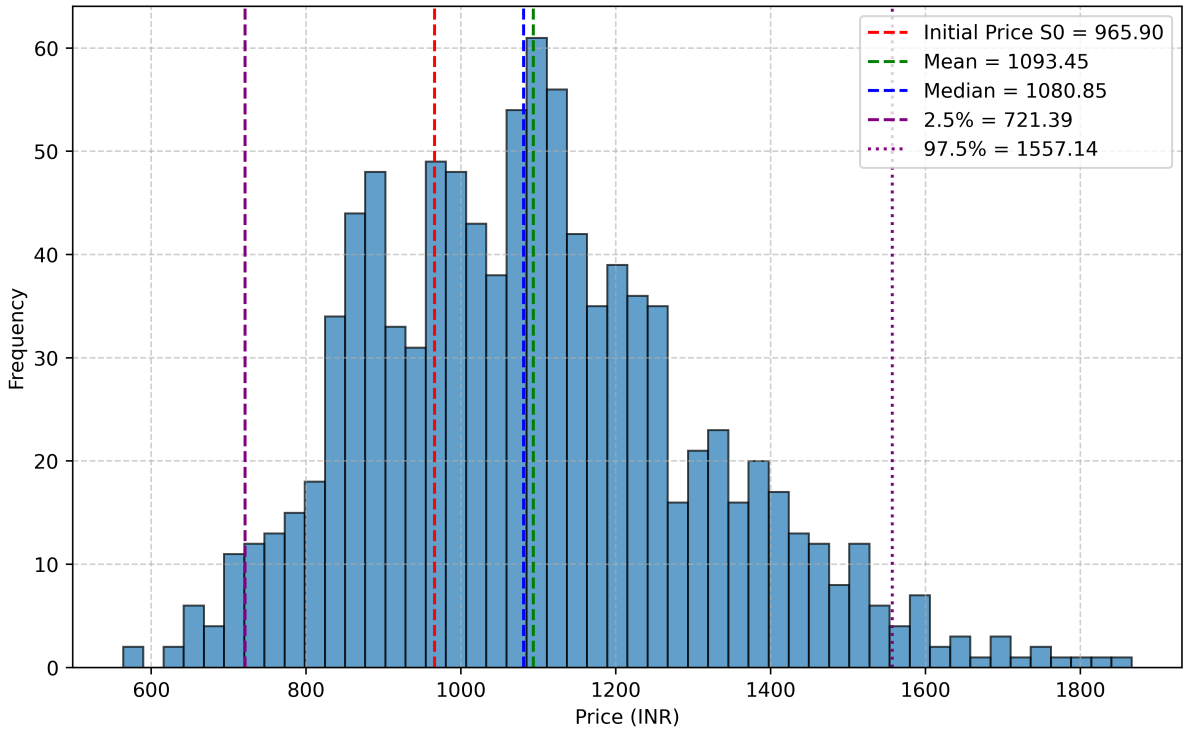

Figure 2: Distribution of Simulated Ending Prices (1 Year)

The model we built is based on Geometric Brownian Motion (GBM), the same assumption as the Black–Scholes option pricing model. While it's mathematically elegant and widely taught, it is too simplistic for real-world quantitative trading.

5 Limitations of GBM in Quantitative Trading

- **Assumption of Constant Drift (μ) and Volatility (σ):**
  - What the model assumes: In GBM, drift (the expected rate of return) and volatility (standard deviation of returns) are fixed constants. Why it's unrealistic:
    - In real markets, volatility is heteroskedastic — it changes over time. For instance, volatility spikes during crises and drops during calm periods (volatility clustering).
    - Drift depends on market regimes (bull vs. bear markets, monetary policy cycles, recessions, etc.). A fixed μ ignores these shifts.
  - Strategies built on GBM may misestimate risk — they'll underestimate volatility during crises and overestimate during calm periods. This is especially dangerous in risk management (VaR, option pricing) where accurate volatility modeling is critical.
- **Log-normal returns:**
  - What the model assumes: Stock prices are log-normally distributed, meaning log returns are normally distributed.
  - Reality:
    - Fat tails: Large price moves (10σ events) happen far more often than a Gaussian model suggests.
    - Skewness: Returns distributions are not symmetric — downside risk is often greater than upside risk.
  - GBM severely underestimates crash risk. This can lead to catastrophic under-hedging of tail events (like 2008, COVID-19 crash).
- **No mean reversion:**
  - What the model assumes: Prices drift indefinitely up or down.
  - Reality: Many assets mean revert around some equilibrium value:
    - Interest rates around a policy target.
    - Commodities around production cost levels.
    - FX rates around purchasing power parity.
  - GBM overstates the possibility of runaway moves in mean-reverting assets. For example, using GBM for bond yield modeling leads to nonsense predictions like "negative infinity yields" or yields skyrocketing unbounded.
- **Poor for Short-Term Predictions**
  - What the model assumes: Drift is meaningful over all horizons.
  - Reality: Over intraday or daily horizons:
    - Drift (μ) is tiny compared to volatility (σ).
    - The signal-to-noise ratio is very low.
    - Prediction models based on GBM essentially output random noise in the short run.
  - GBM is useless for high-frequency trading (HFT) or even swing trading. The model might still work for long-term risk-neutral pricing, but not for generating alpha at short horizons.
- **No Jumps or Regime Changes:**

- What the model assumes: Prices evolve smoothly and continuously.
- Reality: Markets experience discrete jumps:
  - Earnings announcements, central bank decisions, geopolitical shocks, flash crashes.
  - Volatility can shift dramatically between regimes (quiet vs. crisis markets).
- GBM can't capture gap risk — the risk that an asset jumps across a stop-loss overnight. This leads to underestimated option prices and flawed hedges.

6 Why GBM is still Useful?

Despite its limitations, the Geometric Brownian Motion (GBM) model remains useful because of its simplicity, mathematical tractability, and role as a foundation for more advanced models. It captures two essential market features — stochastic (random) price evolution and the non-negativity of asset prices — while being easy to simulate and analyze. GBM underpins the Black–Scholes option pricing model, which revolutionized modern finance and still serves as a benchmark today. In practice, GBM provides a baseline model for stress testing, risk-neutral pricing, and educational purposes, helping traders and researchers build intuition about randomness in asset prices. While not realistic for detailed quantitative trading, it serves as a first approximation and a stepping stone toward more sophisticated models like stochastic volatility or jump-diffusion processes.

7 Conclusion

GBM is elegant and mathematically tractable, which is why it's still taught and used in theoretical finance (especially for the Black-Scholes option pricing model). But in real-world quantitative trading, it's far too simplistic: it misses volatility dynamics, fat tails, mean reversion, jumps, and trading frictions. More advanced models or data-driven ML approaches are used to address these limitations.