Week 2: Application in Quant

From Brownian Motion to Algorithmic Pricing

Objective

This assignment is an **optional assignment** for Week 2. This assignment deals with dynamic stochastic processes.
You will build a **Derivative Pricing Engine** from scratch. You will start with the Nobel Prize-winning Black-Scholes model and progress to pricing "Exotic" options where no analytical formulas exist, forcing the use of advanced Monte Carlo algorithms (Asian Options, Longstaff-Schwartz, and Variance Reduction).

Part 1: The Physics of the Market

To price an asset, we must first model how it moves. Stocks are modeled using **Geometric Brownian Motion (GBM)**. This model assumes stock prices drift upwards over time (the trend) but are shaken by random continuous shocks (volatility).

The Theory: Geometric Brownian Motion

The stochastic differential equation (SDE) for a stock price S_t is solved using Itô's Lemma to give:

$$S_t = S_0 \cdot \exp\left(\left(\mu - \frac{\sigma^2}{2}\right)t + \sigma W_t\right)$$

Where:

- S_0: Starting price.
- μ: Expected return (Drift).
- σ: Volatility (Standard Deviation).
- W_t: A Wiener Process (Random Walk), sampled from $\mathcal{N}(0, t)$.

Task 1: The Vectorized Engine

Simulate the path of a stock (e.g., Apple) over 1 year ($T = 1.0$, 252 trading days).

- **Parameters:** $S_0 = 100$, $\mu = 0.05$ (5% risk-free rate), $\sigma = 0.20$.
- **Vectorization:** You must simulate **50,000 paths** simultaneously. Your code should produce a matrix of shape `(50000, 253)` (Steps + Initial).
- **Constraint:** Do **not** use a for-loop to iterate through days. Use `np.cumsum` on the random shocks to generate the exponent.
- **Plot:** Visualize the first 100 paths using `matplotlib` with low alpha (transparency). Overlay the **Mean Path** and the **Expected Value** $E[S_t] = S_0 e^{\mu t}$.

Part 2: The European Option (The Baseline)

An **Option** is a contract giving the right to buy/sell a stock at a Strike Price K. A **European Call Option** pays you profit if the stock ends up above K at expiry T.

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

Theory (Risk-Neutral Pricing): The value of an option today is the *Discounted Expected Payoff*.

$$V_0 = e^{-rT} \cdot \mathbb{E}[\text{Payoff}]$$

If we assume the market is efficient, we can estimate this expectation by averaging the payoffs of our thousands of Monte Carlo simulations.

Task 2: Pricing and Verification

1. **Simulation:** Using the final prices S_T from Task 1, calculate the payoff for a Call Option with Strike $K = 100$.
2. **Pricing:** Average the payoffs and discount them by e^{-rT} (assume risk-free rate $r = 0.05$).
3. **The Sanity Check:** European options have a closed-form solution called the **Black-Scholes Formula**.
   - Implement a function `black_scholes_call(S, K, T, r, sigma)` using `scipy.stats.norm`.
   - Compare your Monte Carlo price to the exact Black-Scholes price.
   - Plot a **Convergence Graph**: Price Error vs Number of Simulations (N).

Part 3: Path Dependency (Asian Options)

Real-world finance is rarely as simple as Black-Scholes. Consider the **Asian Option**. Unlike a European option which cares only about the price at the *last second* (S_T), an Asian option payoff depends on the **Average Price** over the whole year.

$$\text{Payoff} = \max\left(\frac{1}{M}\sum_{i=1}^{M} S_{t_i} - K, 0\right)$$

Why Monte Carlo? The sum of Log-Normal distributions (the prices) does not have a closed-form distribution. Black-Scholes cannot price this. Monte Carlo is the industry standard here.

Task 3: Pricing the Exotic

1. Using your full path matrix from Task 1, calculate the arithmetic mean of each path axis.
2. Calculate the Asian Call Payoff and discount it.
3. **Analysis:** Is the Asian Option cheaper or more expensive than the European Option? Explain *intuitively* why. (Hint: Does averaging smooth out the volatility?)

Part 4: Optimal Stopping (American Options)

This is the hard part of the assignment. A **European** option can only be exercised at time T. An **American** option can be exercised at **any** time $t \leq T$.

This is an **Optimal Stopping Problem**. At every time step, you have a choice:

- **Exercise:** Take the money now $(S_t - K)$.
- **Continue:** Hold the option, hoping it becomes more valuable later.

To solve this, we cannot just simulate forward. We must work **backwards** from the future. This requires the **Longstaff-Schwartz Algorithm** (Least Squares Monte Carlo).

Task 4: The Longstaff-Schwartz Implementation

Implement the pricing for an **American Put Option** $(K = 100)$.

1. Generate paths. At $t = T$, Value $= \max(K - S_T, 0)$.
2. Iterate backwards from $t = T - 1$ to 1.
3. Identify paths that are "In the Money" (where $K > S_t$).
4. For these paths, run a regression (e.g., `np.polyfit` deg=2):
$$Y = \text{Discounted Value from } t + 1, \quad X = S_t$$
5. Use the regression to predict the **Expected Continuation Value**.
6. Update the Value matrix: If Immediate Payoff > Predicted Continuation, exercise early.
7. Discount back to $t = 0$.

Check: The American Put price must be $\geq$ European Put price.

Part 5: Engineering Rigor (Variance Reduction)

Monte Carlo is computationally expensive. To reduce error by $10\times$, you need $100\times$ simulations. Or, you can use better math.

Task 5: Antithetic Variates

Instead of generating N random paths, generate $N/2$ pairs:

- Path A: Uses random shocks ϵ.
- Path B: Uses random shocks $-\epsilon$.

These paths are negatively correlated. If A goes up, B likely goes down. Averaging them cancels out the noise.

Task: Re-run Task 2 using Antithetic Variates. Plot the Standard Error of the price vs N for both "Vanilla MC" and "Antithetic MC".