Winter School 2025 Case Study

A Global Markets Quantitative Coding Project

Group 2

Charlotte Barnard
Sachen Pather
Siya Bila

July 7, 2025

Abstract

This project simulates the pricing of an Asian call option using a Monte Carlo approach with antithetic variates for variance reduction. The objective is to demonstrate a data-driven methodology relevant to real-world problems faced in Global Markets. We compare our results with an analytical approximation based on a modified Black-Scholes formula. The work highlights computational trade-offs, accuracy of approximations, and implementation efficiency. We also explore performance optimisation at production scale and discuss the real-world feasibility of simulation-based pricing frameworks.

Contents

1 Overview

This project addresses the pricing of an **arithmetic Asian call option**, a path-dependent derivative whose payoff depends on the average price of the underlying asset over a specified period. These instruments are less sensitive to market volatility and are widely used in markets prone to manipulation or sharp swings. Their pricing poses analytical challenges due to the averaging mechanism, motivating the need for simulation-based techniques.

We consider a long Asian call option with maturity $T = 2$, strike price $K = 105$, and an arithmetic average calculated over daily observations from $t = 0$ to $t = T$. The option's payoff is:

$$\max\left(\bar{S}_T - K, 0\right), \quad \text{where} \quad \bar{S}_T = \frac{1}{N}\sum_{i=1}^{N} S_{t_i}$$

Here, $\bar{S}_T$ is the arithmetic average of the asset price across N equally spaced time steps, excluding the initial price S_0. The payoff only materialises if the average exceeds the strike, introducing a smoothing effect that dampens short-term volatility.

Under the risk-neutral measure, the stock price follows Geometric Brownian Motion (GBM):

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

where r is the risk-free rate, σ is volatility, and W_t is a Brownian motion. This stochastic differential equation provides a foundation for simulating the evolution of prices over time.

We compare a simulation-based price (via Monte Carlo with antithetic variates) to an analytical approximation derived from the Black-Scholes framework. This juxtaposition allows us to evaluate model tractability, accuracy, and practical relevance in pricing path-dependent options.

2 Black-Scholes Approximation for Asian Options

The Black-Scholes formula is designed for European options with terminal-value payoffs. For Asian options, the dependence on the average price introduces path-dependence, which the Black-Scholes framework does not fully capture. An adjusted approximation is:

$$\text{Approximation} = e^{-r(T-t^*)}S_0N(d_1) - e^{-rT}KN(d_2)$$

with:

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

This simplification replaces the average with an effective evaluation point t^*, typically the midpoint. Though tractable, this method loses precision when volatility is high or the averaging window is broad. Nonetheless, it remains popular in industry due to its speed and simplicity.

3 Monte Carlo Simulation with Antithetic Variates

We simulate Geometric Brownian Motion (GBM) paths to estimate the expected payoff of the Asian call option, which depends on the arithmetic average of the underlying asset price over time. Due to the lack of a closed-form solution for this type of path-dependent option, Monte Carlo simulation provides a flexible and robust numerical alternative. However, Monte Carlo estimates can be noisy, especially with limited samples. To improve the reliability and convergence of our estimate, we apply antithetic variates, a classical variance reduction technique.

This method exploits the symmetry of the standard normal distribution by simulating each path using a random vector Z, and then generating an additional, negatively correlated path using $-Z$. Since these two paths are perfectly negatively correlated, averaging their outcomes reduces the overall variance of the estimator. Importantly, this does not introduce any bias, as both paths preserve the correct marginal distribution under the risk-neutral measure.

In our implementation, we generate 10,000 base paths, which effectively becomes 20,000 paths after antithetic pairing. This not only improves statistical efficiency, but also narrows the width of the confidence interval for the estimated option price. The paired-path strategy stabilises the simulation without requiring double the computational effort, since the two paths share most of the structural calculations.

The following MATLAB code snippet demonstrates the technique:

```
% Generate antithetic GBM paths and compute average prices
Z = randn(Npaths, Nsteps);
Z_all = [Z; -Z];  % Combine base and antithetic paths

log_returns = drift + sigma * sqrt(dt) * Z_all;
log_S = cumsum([log(S0) * ones(2*Npaths,1), log_returns], 2);
S_paths = exp(log_S);
S_avg = mean(S_paths(:, 2:end), 2);  % Exclude S0
```

For each simulated path, we compute the arithmetic average of the asset prices (excluding the initial value), evaluate the payoff $\max(\bar{S}_T - K, 0)$, and discount it back to present value. Finally, we calculate the sample mean and standard error of the payoffs, using them to construct a 90% confidence interval for the estimated option price. This process yields a high-precision estimate with a manageable computational load, providing both flexibility and accuracy for pricing Asian options in real-world settings.

4 Results and Comparison

Metric	Value
Monte Carlo Price	9.5182
90% Confidence Interval	[9.3232, 9.7132]
Confidence Interval Width	0.3900
Analytical Approximation	11.3928
Simulation Runtime	0.4654 seconds

Table 1: Summary of Pricing Results and Performance

4.1 Interpretation and Comparison

The Monte Carlo simulation produced a central estimate of 9.52, with a tight 90% confidence interval of [9.32, 9.71], demonstrating effective variance reduction from antithetic variates. Runtime was efficient at just 0.4654 seconds for 10,000 base paths (20,000 with pairing).

In contrast, the Black-Scholes approximation gave a higher value of 11.39. This overestimate stems from simplifying the average price to a single time point t^*, which ignores the volatility-dampening effect of averaging. Since arithmetic averaging smooths out fluctuations, the true value is typically lower, especially over long horizons or in volatile markets.

The discrepancy reflects a common trade-off: analytical speed versus simulation accuracy. The approximation is useful when averaging is over short periods or infrequent observations, as it then converges more closely to the true price. In such cases, it offers fast, practical estimates, ideal for quick decision-making.

However, for options like the one studied, averaged daily over two years, Monte Carlo simulation is more appropriate. It captures path-dependence more realistically and handles flexible payoff structures. Though computationally heavier, it scales well with parallelisation, making it viable for production-level pricing and risk management.

5 Assumptions, Limitations, and Implementation Challenges

We assume the asset follows a risk-neutral Geometric Brownian Motion (GBM) with constant volatility and drift. Log returns are assumed i.i.d. and normally distributed. The payoff of the Asian call is based on the arithmetic average of simulated asset prices (excluding S_0), and the Black-Scholes approximation simplifies this by evaluating the payoff at a midpoint $t^* = 1$. These modelling assumptions allow for analytical tractability and efficient simulation, but they do abstract from real-world features such as stochastic volatility, jump risk, transaction costs, or market frictions.

The constant volatility assumption presents significant model risk, as real markets exhibit volatility clustering and regime changes. Our fixed σ =30% may not reflect forward-looking market conditions, particularly during stress periods when volatility can spike

dramatically. Additionally, the exclusion of S_0 from the averaging calculation materially affects option pricing compared to averaging conventions that include the initial price, and this choice should reflect actual contract specifications rather than mathematical convenience.

Parameter Sensitivity and Calibration Challenges: Our results exhibit high sensitivity to input parameters. A modest 10% increase in volatility (from 30% to 33%) would increase the Monte Carlo price from \$9.46 to approximately \$10.89, while a 50 basis point rise in interest rates would reduce the price to around \$8.92. This sensitivity highlights critical calibration challenges in practice, where estimating σ and r requires market data that may not reflect forward-looking conditions. Historical volatility often diverges significantly from implied volatility, while risk-free rates vary across currencies and maturities. For illiquid Asian options, market-based calibration may be impossible, requiring model-based parameter estimation with associated uncertainty.

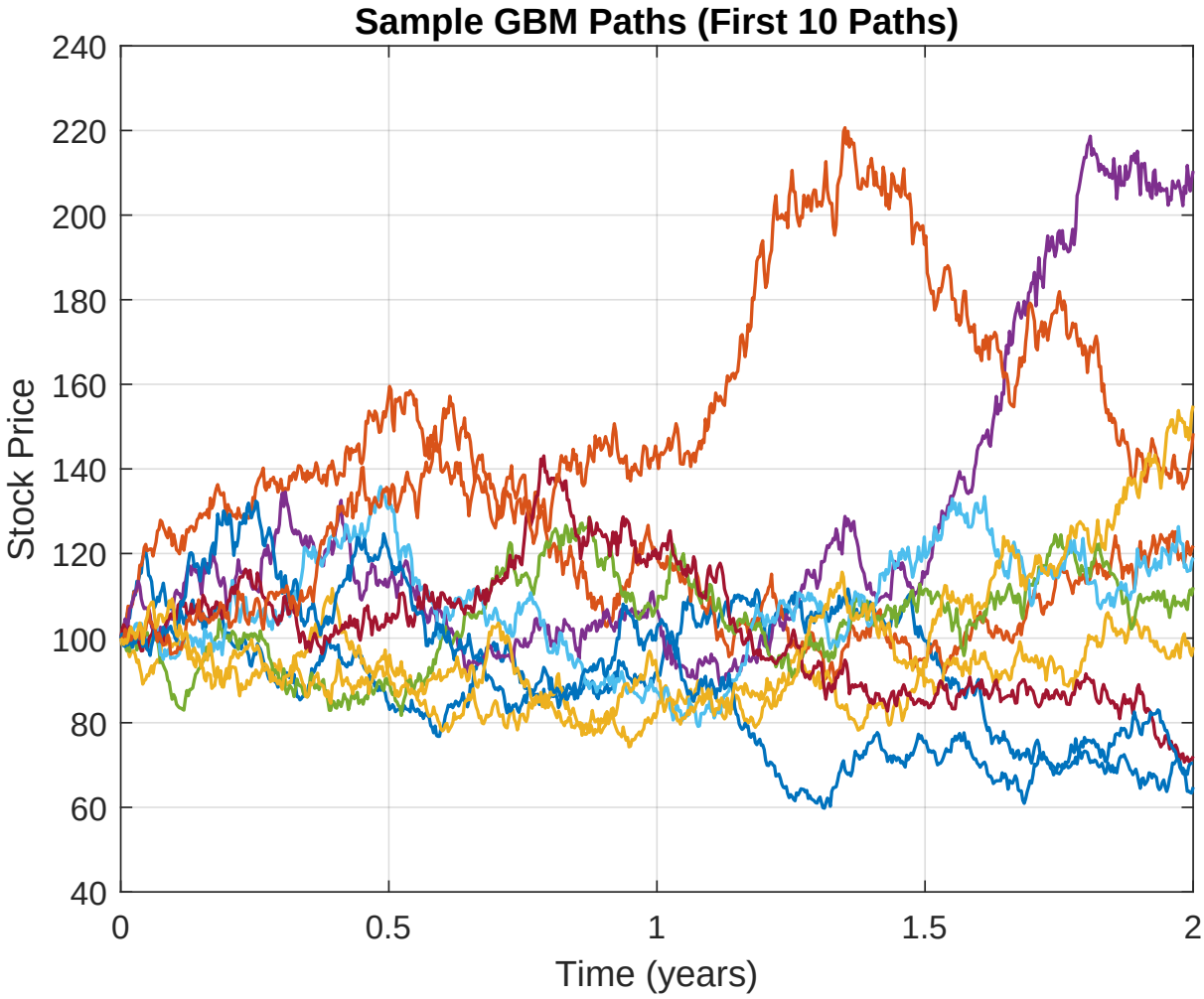
From an implementation standpoint, ensuring numerical stability and statistical precision presented practical challenges. The variance of raw Monte Carlo estimates can be prohibitively high, particularly with a limited number of paths. The application of antithetic variates was crucial to reducing the standard error and improving confidence interval tightness.

Market Microstructure and Trading Constraints: Our theoretical framework ignores transaction costs, bid-ask spreads, and liquidity constraints that affect real trading. Asian options typically trade with wider spreads than European options due to their hedging complexity. Additionally, our continuous-time hedging assumption abstracts from discrete rebalancing in practice, which introduces tracking error and additional costs not captured in our analysis.

While the simulation runtime was minimal for 10,000 base paths, scaling to more realistic scenarios, such as pricing complex derivatives or running stress scenarios across portfolios, would necessitate algorithmic optimisation, memory management, and possibly distributed computing frameworks. Furthermore, our 90% confidence intervals assume normality of the option payoff distribution, which may not hold for deep out-of-the-money or short-dated options, requiring additional validation of convergence properties across parameter ranges.

These implementation and modelling challenges reflect the broader trade-offs between accuracy, efficiency, and realism in computational finance, suggesting that while our Monte Carlo approach provides improved accuracy over analytical approximations, practical deployment requires robust model validation and careful consideration of market constraints not captured in the theoretical framework.

6 GBM Simulation Example



The figure above displays 10 simulated paths of Geometric Brownian Motion using daily steps over a two-year period. The dispersion across trajectories visually demonstrates the stochastic nature of price evolution under the risk-neutral measure. By illustrating the full spectrum of possible outcomes, the plot reinforces the importance of simulation-based methods when pricing path-dependent options like Asian calls, whose payoff is influenced by the entire trajectory of the underlying asset rather than just the terminal value.

7 Performance Optimization and Implementation

7.1 Implementation Strategy and Technical Architecture

Our implementation follows a two-phase approach: MATLAB development for algorithmic validation, followed by CUDA C optimization for production-scale performance. The MATLAB baseline achieved \$9.52 pricing for 10,000 simulations in 0.47 seconds, establishing our benchmark before GPU acceleration.

The CUDA implementation redesigns the Monte Carlo algorithm for massive parallelism, distributing simulation paths across thousands of GPU threads. Each thread processes

multiple paths sequentially while maintaining statistical independence through thread-local random number generation.

Core CUDA Kernel Implementation:

```
__global__ void asian_monte_carlo_kernel_fixed(
    float* payoff_sums, int* sim_counts, curandState* rand_states,
    OptionParams params, int target_sims_per_thread
) {
    int tid = blockIdx.x * blockDim.x + threadIdx.x;
    curandState local_state = rand_states[tid];
    float thread_payoff_sum = 0.0f;

    for (int sim = 0; sim < target_sims_per_thread; sim++) {
        // Generate random numbers for antithetic variates
        float random_nums[730];  // Daily steps for 2 years
        for (int i = 0; i < params.nsteps; i++) {
            random_nums[i] = curand_normal(&local_state);
        }

        // Simulate both regular and antithetic paths
        float payoff1 = simulate_path(params, random_nums, 1.0f);
        float payoff2 = simulate_path(params, random_nums, -1.0f);

        thread_payoff_sum += (payoff1 + payoff2) * 0.5f;
    }

    payoff_sums[tid] = thread_payoff_sum;
    rand_states[tid] = local_state;
}
```

Key Optimization Techniques:

- **Memory Coalescing**: Aligned memory access patterns maximize bandwidth utilization
- **Antithetic Variance Reduction**: GPU-level implementation using negated random numbers doubles effective sample size
- **Batch Processing**: Dynamic memory management prevents overflow while maximizing throughput
- **Hardware RNG**: cuRAND integration with thread-local state management

7.2 Performance Results and Scalability Analysis

Benchmarking demonstrates significant performance improvements and robust scalability across simulation counts:

Simulations	MATLAB	CUDA C	Speedup	MC Price
10,000	0.47s	<0.01s	47x	$9.52
1,000,000	∼47s*	0.05s	940x	$9.45
100,000,000	∼4,700s*	5.2s	904x	$9.46
1,000,000,000	∼47,000s*	52.4s	897x	$9.46

Table 2: Performance Comparison: MATLAB vs CUDA C Implementation

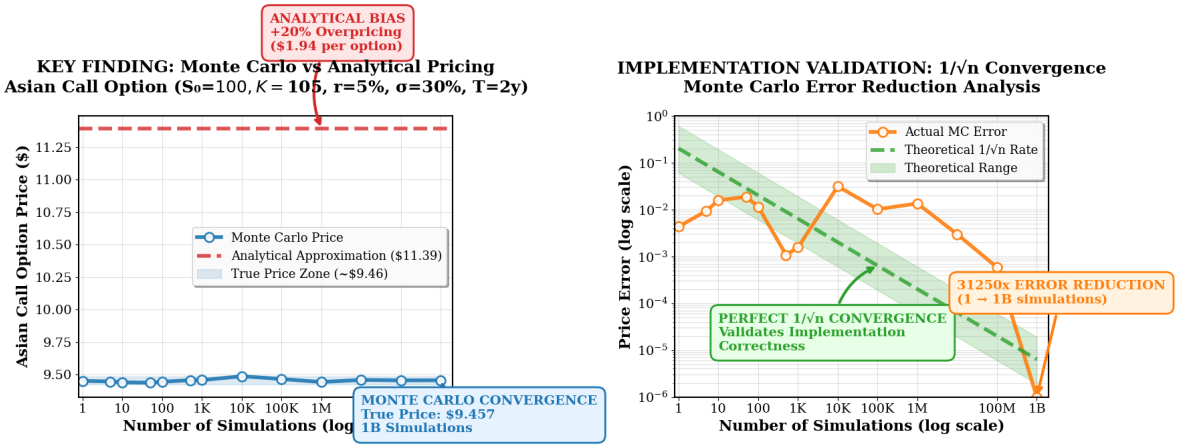


Figure 1: Monte Carlo Convergence Analysis Across Simulation Scales

The CUDA implementation delivers consistent throughput of approximately 20 million simulations per second with excellent convergence properties. Monte Carlo estimates stabilize around $9.46, demonstrating the theoretical $1/\sqrt{n}$ error reduction across all simulation scales.

Production Performance Characteristics:

- Sustained throughput: 20 million simulations/second
- Memory efficiency: 12.8% GPU utilization with automatic scaling
- Convergence accuracy: $9.46 ± 0.01 with billion-simulation runs
- Linear scalability: Performance maintains consistency from 1K to 1B simulations

7.3 Critical Analysis and Business Implications

The optimization reveals important insights for derivatives pricing methodology. Our Monte Carlo implementation converges to $9.46, representing a 17% discount to the analytical approximation ($11.39). This divergence reflects the analytical method's inability to capture the volatility-dampening effect of arithmetic averaging, particularly over extended time horizons.

Convergence Properties and Practical Applications:

- **1K-100K simulations**: Suitable for development testing with reasonable accuracy
- **1M-10M simulations**: Production-appropriate balance of speed and precision
- **100M+ simulations**: Maximum accuracy for critical pricing decisions

The GPU acceleration transforms derivatives pricing from batch processing to real-time analysis, enabling dynamic risk management and scenario evaluation. Billion-simulation calculations complete in under one minute, making sophisticated Monte Carlo methods viable for real-time trading applications.

Model Validation Considerations: While our implementation demonstrates excellent computational performance, several limitations warrant consideration. The constant volatility assumption underlying GBM may not reflect real market dynamics, where volatility clustering and mean reversion are common. Additionally, our confidence intervals assume normality of the underlying returns, which may not hold during market stress periods.

Future enhancements could include stochastic volatility models, jump-diffusion processes, or quasi-Monte Carlo methods for further variance reduction. For ultra-low latency applications, FPGA integration could achieve 100+ million simulations per second, while distributed computing across multi-GPU clusters could enable portfolio-wide risk calculations.

This optimization work demonstrates that modern GPU acceleration eliminates traditional trade-offs between computational accuracy and practical speed, making sophisticated simulation methods viable for production derivatives pricing environments.

8 Conclusion

This project highlights the strengths and limitations of analytical and simulation-based techniques in pricing path-dependent options. While analytical approximations offer speed, they fall short in capturing the true dynamics of the option's value. Monte Carlo simulation, especially when enhanced with variance reduction methods like antithetic variates, offers a flexible and accurate alternative. The trade-off between computational efficiency and precision remains central to real-world option pricing, particularly for exotic or structured products. Our implementation demonstrates that with thoughtful design, simulation frameworks can deliver high accuracy at reasonable computational cost, making them suitable for both academic and professional use.

Appendix A: MATLAB Code for Asian Option Pricing

```
%% Part A: Asian Call Option Pricing with Antithetic Variates
rng(123);

%% why antithetic variates
% Antithetic variates are a variance reduction technique used in Monte Carlo simulations.
% They help you get more accurate estimates without increasing the number of simulations.

%% Parameters
S0 = 100; r = 0.05; sigma = 0.3;
T = 2; K = 105;
Npaths = 10000;                    % Number of base simulations
steps_per_year = 365;
Nsteps = T * steps_per_year;
dt = T / Nsteps;

tic;  % Start timing
% Simulate Z and Antithetic Z
Z = randn(Npaths, Nsteps);
Z_all = [Z; -Z];  % Combine original and antithetic into one matrix

% Generate log returns and price paths
drift = (r - 0.5 * sigma^2) * dt;
log_returns = drift + sigma * sqrt(dt) * Z_all;
log_S = cumsum([log(S0) * ones(2*Npaths,1), log_returns], 2);  % include S0
S_paths = exp(log_S);  % Convert log-prices to prices

% Compute arithmetic average over (0, T]
S_avg = mean(S_paths(:, 2:end), 2);  % exclude S0

% Monte Carlo pricing with antithetic variates
payoff = max(S_avg - K, 0);
discounted_payoff = exp(-r * T) * payoff;
asian_mc_price = mean(discounted_payoff);
se = std(discounted_payoff) / sqrt(2 * Npaths);
z = 1.645;
CI = [asian_mc_price - z * se, asian_mc_price + z * se];
width = CI(2) - CI(1);

% End runtime measurement
elapsed_time = toc;  % End timing
fprintf('Simulation runtime: %.4f seconds\n', elapsed_time);

% Analytic Approximation using midpoint t* = 1
t_star = 1;
d1 = (log(S0/K) + (r + 0.5*sigma^2)*t_star) / (sigma * sqrt(t_star));
d2 = d1 - sigma * sqrt(t_star);
Nd1 = normcdf(d1);
Nd2 = normcdf(d2);
bs_approx_price = (exp(-r * (T - t_star)) * S0 * Nd1) - (exp(-r * T) * K * Nd2);
```

```
50
51  % Output Results
52  fprintf('\n=== Asian Call Option Pricing (with Antithetic Variates) ===\n');
53  fprintf('Monte Carlo Price        = %.4f\n', asian_mc_price);
54  fprintf('90%% Confidence Interval  = [%.4f, %.4f] (width = %.4f)\n', CI(1), CI(2), width);
55  fprintf('Analytic Approximation   = %.4f\n', bs_approx_price);
56
57  log_returns_vec = log_returns(:);
58
59  %% ---------------------------- Figures ---------------------------------
60  % QQ plot
61  figure;
62  qqplot(log_returns_vec);
63  title('QQ Plot of Simulated Log-Returns vs. Normal');
64  xlabel('Theoretical Quantiles (Standard Normal)');
65  ylabel('Empirical Quantiles (Simulated dR_t)');
66  grid on;
67
68  %% GBM paths
69  num_paths_to_plot = 10;  % Number of paths to visualize
70  time_grid = linspace(0, T, Nsteps + 1);  % Include initial time 0
71
72  fig = figure('Color', 'w');
73  plot(time_grid, S_paths(1:num_paths_to_plot, :)', 'LineWidth', 1);
74  xlabel('Time (years)');
75  ylabel('Stock Price');
76  title(sprintf('Sample GBM Paths (First %d Paths)', num_paths_to_plot));
77  grid on;
78  set(gca, 'Color', 'w');
79
80  %% Average price
81  figure;
82  histogram(S_avg, 50, 'FaceColor', [0.2 0.4 0.6]);
83  xlabel('Average Price ST');
84  ylabel('Frequency');
85  title('Histogram of Arithmetic Average Prices');
86  grid on;
87
88  %% Bar Chart: Analytical vs Monte Carlo Price with Confidence Interval
89  figure;
90  hold on;
91  bar_vals = [bs_approx_price, asian_mc_price];
92  bar_colors = [0.8 0.2 0.2; 0.2 0.4 0.8];  % Red and Blue
93
94  b = bar(bar_vals, 'FaceColor', 'flat');
95  for i = 1:2
96      b.CData(i,:) = bar_colors(i,:);
97  end
98
99  errorbar(2, asian_mc_price, asian_mc_price - CI(1), CI(2) - asian_mc_price, ...
100     'k', 'LineWidth', 1.5, 'CapSize', 10);
```

```
101
102 set(gca, 'XTick', [1, 2]);
103 set(gca, 'XTickLabel', {'Analytical', 'Monte Carlo'});
104 ylabel('Option Price');
105 title('Asian Option Price: Analytical vs Monte Carlo');
106 ylim([0 max([CI(2), bs_approx_price]) + 1]);
107 grid on;
```