Q2_main

2024-10-30

3. Monte Carlo Simulation for pricing Asian Call Option

3.1 Introduction

An Asian option is a type of exotic option where the payoff depends on the average price of the underlying asset over a certain period, rather than its price at a specific point in time. This average can be computed as either an arithmetic average or a geometric average. Asian options are particularly useful in markets where the underlying asset's price is highly volatile, as they reduce the impact of price manipulation or extreme price movements near expiration. For an Asian call or put option, the payoff formulas are listed as follows.

The payoff for a continuous arithmetic average Asian call or put option is:

$$\Phi(S) = \max\left(\frac{1}{T}\int_0^T S(t)dt - K, 0\right) \quad \text{or} \quad \Phi(S) = \max\left(K - \frac{1}{T}\int_0^T S(t)dt, 0\right).$$

The payoff for a continuous geometric average Asian call or put option is:

$$\Phi(S) = \max\left(e^{\frac{1}{T}\int_0^T \log S(t)dt} - K, 0\right) \quad \text{or} \quad \Phi(S) = \max\left(K - e^{\frac{1}{T}\int_0^T \log S(t)dt}, 0\right).$$

For discrete monitoring, the arithmetic average Asian call or put option has the following payoff:

$$\Phi(S) = \max\left(\frac{1}{m+1}\sum_{i=0}^{m} S\left(\frac{iT}{m}\right) - K, 0\right) \quad \text{or} \quad \Phi(S) = \max\left(K - \frac{1}{m+1}\sum_{i=0}^{m} S\left(\frac{iT}{m}\right), 0\right).$$

Similarly, for discrete monitoring, the geometric average Asian call or put option has the payoff:

$$\Phi(S) = \max\left(e^{\frac{1}{m+1}\sum_{i=0}^{m}\log S\left(\frac{iT}{m}\right)} - K, 0\right) \quad \text{or} \quad \Phi(S) = \max\left(K - e^{\frac{1}{m+1}\sum_{i=0}^{m}\log S\left(\frac{iT}{m}\right)}, 0\right).$$

Asian options are widely used in financial markets for energy and commodity trading. Their structure reduces the risk of price manipulation near expiration and smooths out price volatility through averaging.

In the following section, we will focus on three parts. The first is pricing an arithmetic average Asian call option using the Monte Carlo simulation method. Then, we will investigate how to enhance the efficiency of the Monte Carlo method by implementing variance reduction techniques: the Control Variate Method. Additionally, we will further explore variance reduction methods, such as combining control variates with stratified sampling and the brownian bridge.

3.2 Asian Option Pricing– Monte Carlo Method

Monte Carlo (MC) simulation is a widely used numerical method for pricing financial derivatives. The MC method involves generating a large number of simulated paths for the underlying asset price and computing the average payoff over these paths to estimate the option price. Its flexibility makes it suitable for pricing

Asian options, as it can handle the arithmetic averaging of the underlying prices, which does not have a closed-form solution.

The key steps in pricing an Asian call option using the Monte Carlo method are as follows:

Simulate Asset Price Paths: The underlying asset price S_t is modeled as a geometric Brownian motion (GBM) under the risk-neutral measure: $dS_t = (r - q)S_t\,dt + \sigma S_t\,dW_t$, where: - r: Risk-free interest rate, - q: Dividend yield, - σ: Volatility of the asset, - W_t: Standard Brownian motion.

We apply a logarithmic transformation on S_t, and use Ito's Lemma to solve the stochastic differential equation. Integrating the log-transformed SDE, we get the stock price at time t under risk-neutral probability: $\log(S_t) = \log(S_0) + (r - q - 0.5\sigma^2)t + \sigma W_t$. To Simulate stock prices, we utilize a vectorized approach in the logarithmic domain to ensure both computational efficiency and numerical stability. We begin by generating a matrix of standard normal random variables, $Z \sim \mathcal{N}(0, 1)$, representing N paths over m discrete time steps. We calculate the drift term and diffusion term separately. The drift term is added to account for deterministic growth. A cumulative sum is applied to the diffusion term across time steps to approximate W_t. This approach leverages matrix operations to compute all paths simultaneously, eliminating the need for iterative updates and enhancing performance for large-scale simulations. Furthermore, working in the logarithmic domain mitigates numerical instabilities that may arise from compounding errors in the stock price domain, particularly for long time horizons or high volatility.

Calculate Payoff: For each simulated path, compute the arithmetic average of the asset prices: $\bar{S}_{\text{arith}} = \frac{1}{m}\sum_{i=1}^{m} S_{t_i}$. Then, calculate the payoff of the option for each path as: $\text{Payoff}_i = \max(\bar{S}_{\text{arith}} - K, 0)$, where K is the strike price.

Discount Payoffs and Estimate Option Price: Discount the payoff to present value using the risk-free rate: $\text{Discounted Payoff}_i = e^{-rT} \cdot \text{Payoff}_i$. The Monte Carlo estimate of the Asian option price is the average of the discounted payoffs: $\text{Option Price} = \frac{1}{N}\sum_{i=1}^{N} \text{Discounted Payoff}_i$, where N is the number of simulated paths.

Measure Uncertainty: The standard error of the Monte Carlo estimate is given by: $\text{Standard Error} = \frac{\text{Standard Deviation of Payoffs}}{\sqrt{N}}$.

Below is the R implementation of calculating Asian call option prices using monte carlo method:

```r
price_asian_call_MC <- function(S0, K, T, r, q, sigma, m, N) {
  start_time <- proc.time()
  delta_t <- T / m
  sqrt_delta_t <- sqrt(delta_t)

  # Generate N x m standard normal random variables
  Z <- matrix(rnorm(N * m), nrow = N, ncol = m)

  # Simulate log prices and then transform to prices
  drift <- (r - q - 0.5 * sigma^2) * delta_t
  diffusion <- sigma * sqrt_delta_t * Z
  W <- t(apply(diffusion, 1, cumsum))
  log_S <- log(S0) + outer(rep(1, N), drift * (1:m)) + W
  S <- exp(log_S)

  # Calculate arithmetic average and payoffs
  S_bar <- rowMeans(S)
  payoffs <- pmax(S_bar - K, 0)
  discounted_payoffs <- exp(-r * T) * payoffs

  # Estimate option price
  option_price <- mean(discounted_payoffs)
```

```r
  std_error <- sd(discounted_payoffs) / sqrt(N)

  # Calculate 95% confidence interval
  CI_lower <- option_price - 1.96 * std_error
  CI_upper <- option_price + 1.96 * std_error

  end_time <- proc.time()
  comp_time <- (end_time - start_time)[["elapsed"]]

  return(list(
    N = N,
    Option_Price = round(option_price, 2),
    Standard_Error = round(std_error, 5),
    Confidence_Interval = c(round(CI_lower, 2), round(CI_upper, 2)),
    Computation_Time_sec = round(comp_time, 4)
  ))
}
```

```r
# Set parameters
S0 <- 100      # Initial asset price
K <- 100       # Strike price
T <- 1         # Time to maturity (in years)
r <- 0.10      # Risk-free rate
q <- 0         # Dividend yield
sigma <- 0.20  # Volatility
m <- 50        # Number of monitoring points

sample_sizes <- c(1000, 4000, 16000, 64000, 256000)

# Initialize a data frame to store results
standard_mc_results <- data.frame(
  Sample_Size = numeric(),
  Option_Price = numeric(),
  Standard_Error = numeric(),
  CI_Lower = numeric(),
  CI_Upper = numeric(),
  Computation_Time_sec = numeric(),
  stringsAsFactors = FALSE
)

# Set the seed once before the loop
set.seed(123)

# Loop through different sample sizes
for (N in sample_sizes) {
  result <- price_asian_call_MC(S0, K, T, r, q, sigma, m, N)
  standard_mc_results <- rbind(standard_mc_results, data.frame(
    Sample_Size = N,
    Option_Price = result$Option_Price,
    Standard_Error = result$Standard_Error,
    CI_Lower = result$Confidence_Interval[1],
    CI_Upper = result$Confidence_Interval[2],
    Computation_Time_sec = result$Computation_Time_sec
```

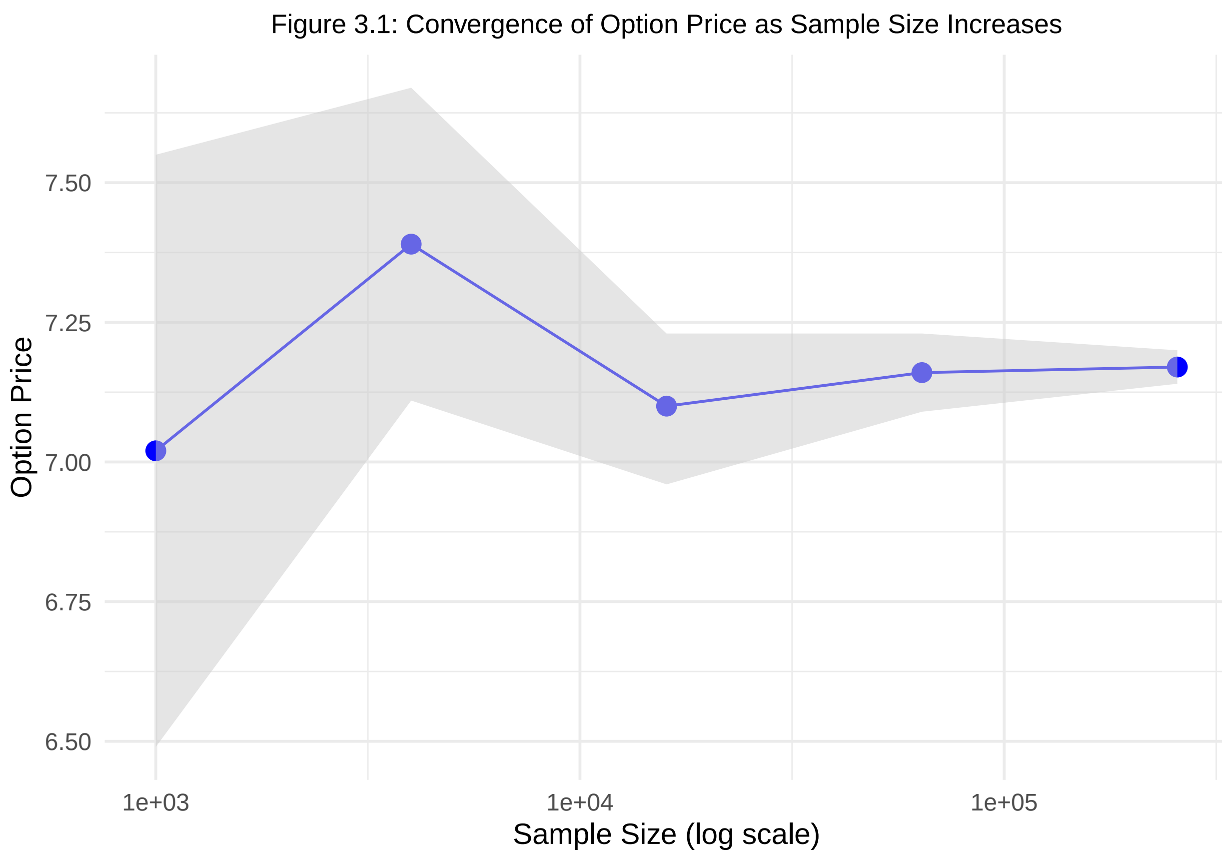
```
  ))
}

suppressWarnings(suppressPackageStartupMessages({
  library(ggplot2)
  library(knitr)
  library(dplyr)
}))
```

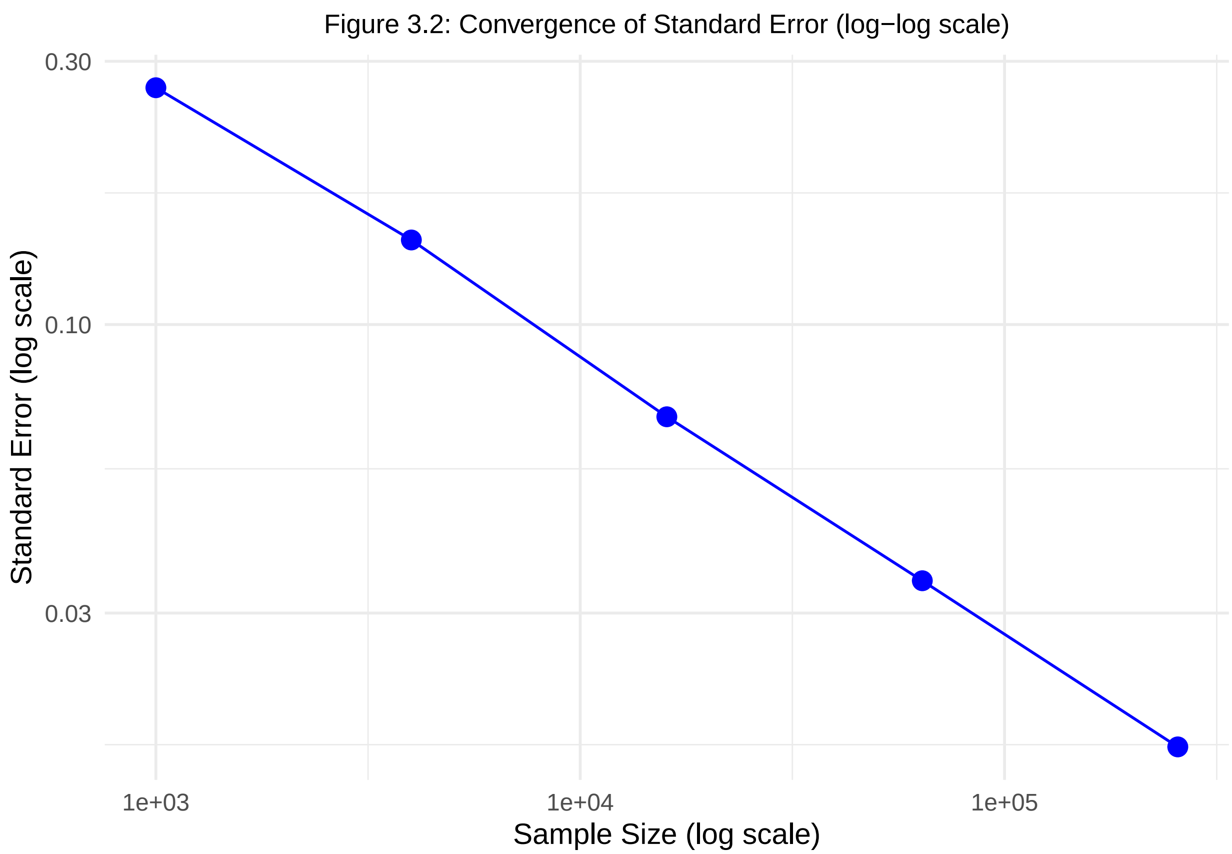
Table 3.1: Monte Carlo Simulation Results

Sample_Size	Option_Price	Standard_Error	CI_Lower	CI_Upper	Computation_Time_sec
1000	7.02	0.26869	6.49	7.55	0.004
4000	7.39	0.14233	7.11	7.67	0.013
16000	7.10	0.06805	6.96	7.23	0.090
64000	7.16	0.03433	7.09	7.23	0.324
256000	7.17	0.01716	7.14	7.20	1.223

```
# Plot the convergence of Option Price
ggplot(standard_mc_results, aes(x = Sample_Size, y = Option_Price)) +
  geom_line(color = "blue") +
  geom_point(size = 3, color = "blue") +
  geom_ribbon(aes(ymin = CI_Lower, ymax = CI_Upper), fill = "grey80", alpha = 0.5) +
  scale_x_log10() +
  labs(
    title = "Figure 3.1: Convergence of Option Price as Sample Size Increases",
    x = "Sample Size (log scale)",
    y = "Option Price"
  ) +
  theme_minimal()+
  theme(
    plot.title = element_text(
      size = 10,
      hjust = 0.5,
      vjust = 1.2
    )
  )
```




```
# Plot the convergence of Standard Error on log-log scale
ggplot(standard_mc_results, aes(x = Sample_Size, y = Standard_Error)) +
  geom_line(color = "blue") +
  geom_point(size = 3, color = "blue") +
  scale_x_log10() +
  scale_y_log10() +
  labs(
    title = "Figure 3.2: Convergence of Standard Error (log-log scale)",
    x = "Sample Size (log scale)",
    y = "Standard Error (log scale)"
  ) +
  theme_minimal()+
  theme(
    plot.title = element_text(
      size = 10,
      hjust = 0.5,
      vjust = 1.2
    )
  )
```



Convergence of Option Price

The first plot highlights the behavior of the estimated option price as the sample size increases. For smaller sample sizes (e.g., $N = 1000$), the option price exhibits larger variability and a wider confidence interval, indicating less precision. As the sample size increases to $N = 256,000$, the option price converges to a stable value of approximately 7.17, with the 95% confidence interval narrowing significantly to a width of 0.06. This reflects the expected improvement in accuracy with larger sample sizes, consistent with Monte Carlo theory. The convergence is gradual, with the estimates stabilizing beyond $N = 64,000$.

Convergence of Standard Error

The second plot, presented on a log-log scale, demonstrates the relationship between the standard error and the sample size. The convergence follows the theoretical rate of:

$$\text{Standard Error} \propto \frac{1}{\sqrt{N}}$$

This means that as the sample size increases by a factor of 4, the standard error decreases approximately by half. The straight line in the log-log plot shows that doubling the sample size results in a consistent proportional reduction in the standard error, making it predictable and reliable. For example: at $N = 1000$, the standard error is 0.26869. At $N = 4000$ (4 times $N = 1000$), the standard error decreases to approximately 0.14233. Similarly, at $N = 16,000$ (another 4x increase), the standard error further decreases to 0.06805.

3.3 Variance Reduction Techniques– Control Variate Method

In this section, we will introduce a variance reduction technique: control variate method. Suppose we want to estimate the expected value $\theta = \mathbb{E}[Y]$, where $Y = g(X)$ is a function of some random variable X. If we can find another random variable Z, for which the expected value $\mathbb{E}[Z]$ is known, we can construct alternative estimators for θ. For example,

The standard Monte Carlo estimator:

$$\hat{\theta} = Y$$

The control variate estimator:

$$\hat{\theta}_c = Y + c \cdot (Z - \mathbb{E}[Z]),$$

where c is a constant.

It can be shown that the control variate estimator $\hat{\theta}_c$ is unbiased, as:

$$\mathbb{E}[\hat{\theta}_c] = \mathbb{E}[Y] + c \cdot (\mathbb{E}[Z] - \mathbb{E}[Z]) = \mathbb{E}[Y] = \theta.$$

To minimize the variance of $\hat{\theta}_c$, we start with its variance formula:

$$\mathrm{Var}(\hat{\theta}_c) = \mathrm{Var}(Y) + c^2 \cdot \mathrm{Var}(Z) + 2c \cdot \mathrm{Cov}(Y, Z).$$

Treating this as a function of c, we differentiate with respect to c to obtain:

$$f'(c) = 2c \cdot \mathrm{Var}(Z) + 2 \cdot \mathrm{Cov}(Y, Z).$$

Setting $f'(c) = 0$, we solve for c to find the critical point:

$$c_{\mathrm{opt}} = -\frac{\mathrm{Cov}(Y, Z)}{\mathrm{Var}(Z)}.$$

To confirm that this value minimizes the variance, we compute the second derivative:

$$f''(c) = 2 \cdot \mathrm{Var}(Z).$$

Since $\mathrm{Var}(Z) > 0$, $f(c)$ is convex, and c_{opt} is the minimizer.

This demonstrates that the control variate method reduces variance by leveraging the correlation between Y and Z, particularly when they are highly correlated.

In this practice, we will use geometric Asian Call as a control variant. This method is efficient, since first, the correlation between the arithmetic average and the geometric average is very high; second, the counterpart geometric average Asian option price has a closed-form solution. We use the following formula to calculate geometric Asian call option price, serving as the expected value of the control variant in the control variate estimator:

$$\begin{aligned}
\sigma_z^2 &= \sigma^2 \cdot \frac{(m+1)(2m+1)}{6m^2} \\
\mu &= \left(r - q - \frac{1}{2}\sigma^2\right) \cdot \frac{m+1}{2m} + \frac{1}{2}\sigma_z^2 \\
d_1 &= \frac{\ln\left(\frac{S_0}{K}\right) + \left(\mu + \frac{1}{2}\sigma_z^2\right) T}{\sigma_z \sqrt{T}} \\
d_2 &= \frac{\ln\left(\frac{S_0}{K}\right) + \left(\mu - \frac{1}{2}\sigma_z^2\right) T}{\sigma_z \sqrt{T}} \\
\text{Geometric Asian Call Price} &= e^{-rT} \left[S_0 \cdot e^{\mu T} \cdot N(d_1) - K \cdot N(d_2) \right]
\end{aligned}$$

Here: σ_z^2: Effective volatility of the geometric average. μ: Drift term adjusted for the geometric average. d_1, d_2: Terms used in the Black-Scholes framework for the option price. $N(d)$: Cumulative distribution function of the standard normal distribution.

Below is the R implementation of calculating the analytical price of Geometric Asian Call Option:

```r
# function to calculate the analytical price of Geometric Asian Call Option
price_geometric_asian_call <- function(S0, K, T, r, q, sigma, m){
  delta_t <- T/m
  sigma_sq <- sigma^2
  sigma_z_sq <- (sigma_sq) * (m + 1) * (2 * m + 1) / (6 * m^2)
  drift <-(r - q - 0.5*sigma_sq) * (m + 1) / (2 * m) + 0.5 * sigma_z_sq
  sigma_z <- sqrt(sigma_z_sq)
  d1 <- (log(S0/K) + (drift + 0.5 * sigma_z_sq) * T) / (sigma_z * sqrt(T))
  d2 <- (log(S0/K) + (drift - 0.5 * sigma_z_sq) * T) / (sigma_z * sqrt(T))
  geo_asian_price <- exp(-r*T) * (S0* exp(drift * T) * pnorm(d1) - K * pnorm(d2))
  return(geo_asian_price)
}
```

To implement this method, we first simulate the stock price Y using standard Monte Carlo techniques. Next, we compute the geometric average of the stock prices and take its logarithm to derive the geometric option price, denoted as Z. The closed-form solution for the geometric option price allows us to calculate its expected value, $\mathbb{E}[Z]$. Using this information, we construct the control variate estimator:

$\hat{\theta}_c = Y + c \cdot (Z - \mathbb{E}[Z])$, where $c_{\text{opt}} = -\frac{\text{Cov}(Y,Z)}{\text{Var}(Z)}$ is the optimal control coefficient. Below is the R implementation of the Monte Carlo control variate method:

```r
# function to price asian call option using geometric asian call price as a control variate
price_asian_call_MC_control_variate <- function(S0, K, T, r, q, sigma, m, N) {
  start_time <- proc.time()

  delta_t <- T / m
  sqrt_delta_t <- sqrt(delta_t)

  Z <- matrix(rnorm(N * m), nrow = N, ncol = m)

  drift <- (r - q - 0.5 * sigma^2) * delta_t
  diffusion <- sigma * sqrt_delta_t * Z
  W <- t(apply(diffusion, 1, cumsum))
  log_S <- log(S0) + outer(rep(1, N), (r - q - 0.5 * sigma^2) * (delta_t * (1:m))) + W
  S <- exp(log_S)

  S_bar_arith <- rowMeans(S)
  S_bar_geo <- exp(rowMeans(log(S)))

  payoffs_arith <- pmax(S_bar_arith - K, 0)

  # Calculate payoffs for geometric Asian call
  payoffs_geo <- pmax(S_bar_geo - K, 0)

  # Discount payoffs to present value
  discounted_payoffs_arith <- exp(-r * T) * payoffs_arith
  discounted_payoffs_geo <- exp(-r * T) * payoffs_geo

  # Calculate analytical price of Geometric Asian Call
  geo_asian_price <- price_geometric_asian_call(S0, K, T, r, q, sigma, m)

  # Calculate covariance and variance for Control Variate
  cov_xy <- cov(discounted_payoffs_arith, discounted_payoffs_geo)
  var_y <- var(discounted_payoffs_geo)
```

```r
  # Optimal coefficient
  theta <- cov_xy / var_y

  # Calculate Control Variate estimator
  control_variate_estimator <- discounted_payoffs_arith - theta * (discounted_payoffs_geo - geo_asian_p

  # Calculate option price using Control Variate
  option_price_cv <- mean(control_variate_estimator)

  std_error_cv <- sd(control_variate_estimator) / sqrt(N)

  CI_lower_cv <- option_price_cv - 1.96 * std_error_cv
  CI_upper_cv <- option_price_cv + 1.96 * std_error_cv

  end_time <- proc.time()
  comp_time <- (end_time - start_time)[["elapsed"]]

  return(list(
    N = N,
    Option_Price = round(option_price_cv, 2),
    Standard_Error = round(std_error_cv, 5),
    Confidence_Interval = c(round(CI_lower_cv, 2), round(CI_upper_cv, 2)),
    Computation_Time_sec = round(comp_time, 4)
  ))
}
```

```r
# Set parameters
S0 <- 100       # Initial asset price
K <- 100        # Strike price
T <- 1          # Time to maturity (in years)
r <- 0.10       # Risk-free rate
q <- 0          # Dividend yield
sigma <- 0.20   # Volatility
m <- 50         # Number of monitoring points

sample_sizes <- c(1000, 4000, 16000, 64000, 256000)
# Initialize the results data frame
results_cv <- data.frame(
  Sample_Size = numeric(),
  Option_Price = numeric(),
  Standard_Error = numeric(),
  CI_Lower = numeric(),
  CI_Upper = numeric(),
  Computation_Time_sec = numeric(),
  stringsAsFactors = FALSE
)
# Remove or comment out the next line to prevent printing the empty data frame
# results_cv

for (N in sample_sizes) {
  # Monte Carlo with Control Variate
  res_cv <- price_asian_call_MC_control_variate(S0, K, T, r, q, sigma, m, N)
  results_cv <- rbind(results_cv, data.frame(
```

```r
    Sample_Size = N,
    Option_Price = res_cv$Option_Price,
    Standard_Error = res_cv$Standard_Error,
    CI_Lower = res_cv$Confidence_Interval[1],
    CI_Upper = res_cv$Confidence_Interval[2],
    Computation_Time_sec = res_cv$Computation_Time_sec
  ))
}
```

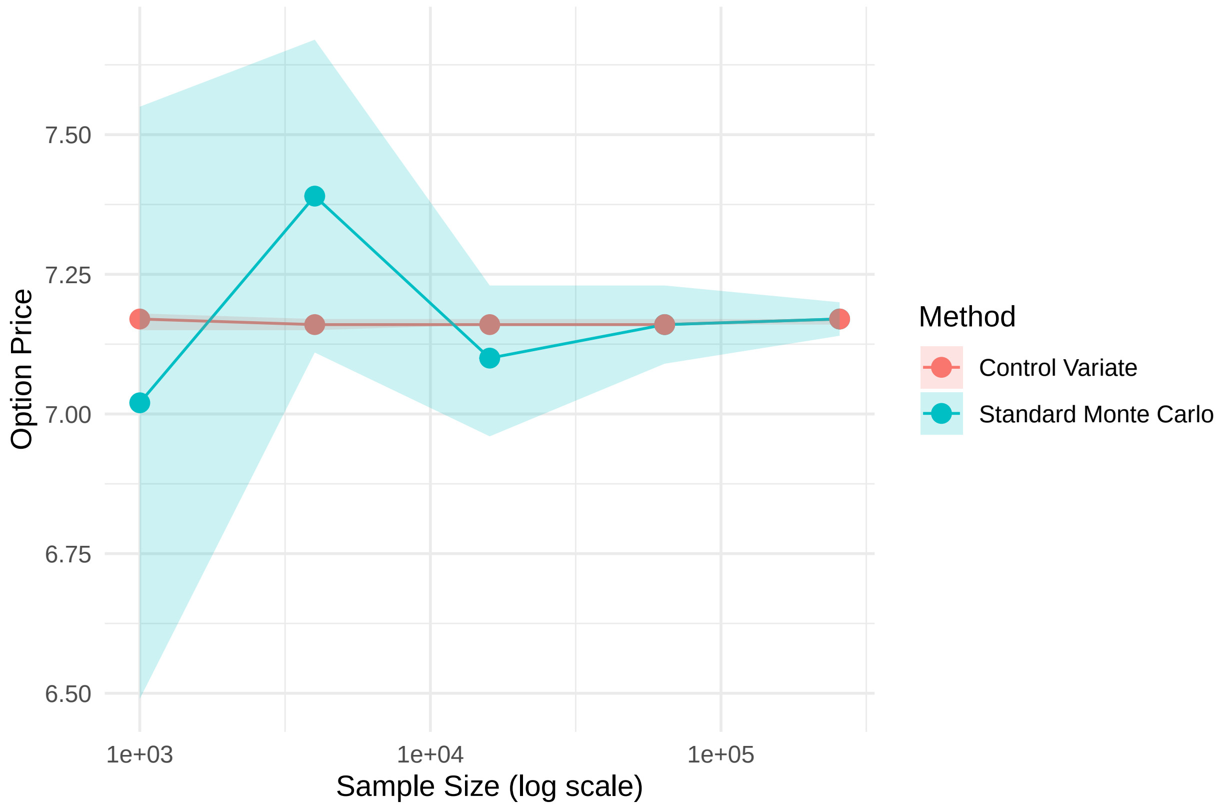
Table 3.2: Monte Carlo with Control Variate Simulation Results

Sample_Size	Option_Price	Standard_Error	CI_Lower	CI_Upper	Computation_Time_sec
1000	7.17	0.00888	7.15	7.18	0.012
4000	7.16	0.00401	7.15	7.17	0.012
16000	7.16	0.00205	7.16	7.17	0.051
64000	7.16	0.00102	7.16	7.17	0.222
256000	7.17	0.00051	7.16	7.17	1.140

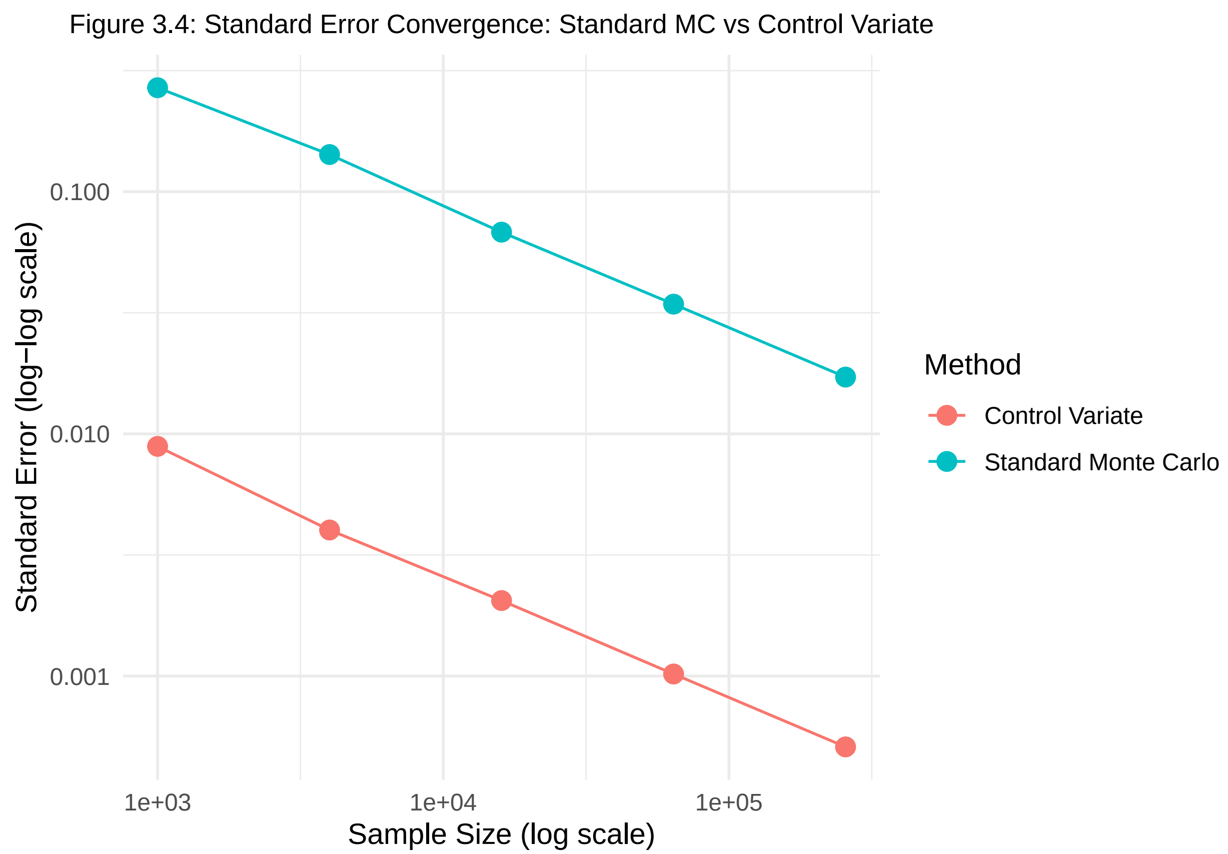
```r
# Combine data if not already done
standard_mc_results$Method <- "Standard Monte Carlo"
results_cv$Method <- "Control Variate"
comparison_results <- rbind(standard_mc_results, results_cv)
```

```r
# Plot Option Price Convergence with 95% CI
ggplot(comparison_results, aes(x = Sample_Size, y = Option_Price, color = Method, group = Method)) +
  geom_line() +
  geom_point(size = 3) +
  geom_ribbon(
    aes(ymin = CI_Lower, ymax = CI_Upper, fill = Method),
    alpha = 0.2,
    color = NA
  ) +
  scale_x_log10() +
  labs(
    title = "Figure 3.3: Option Price Convergence with 95% CI: Standard MC vs Control Variate",
    x = "Sample Size (log scale)",
    y = "Option Price",
    color = "Method",
    fill = "Method"
  ) +
  theme_minimal() +
  theme(
    plot.title = element_text(
      size = 10,
      hjust = 0.5,
      vjust = 1.2
    )
  )
```

Figure 3.3: Option Price Convergence with 95% CI: Standard MC vs Control Variate

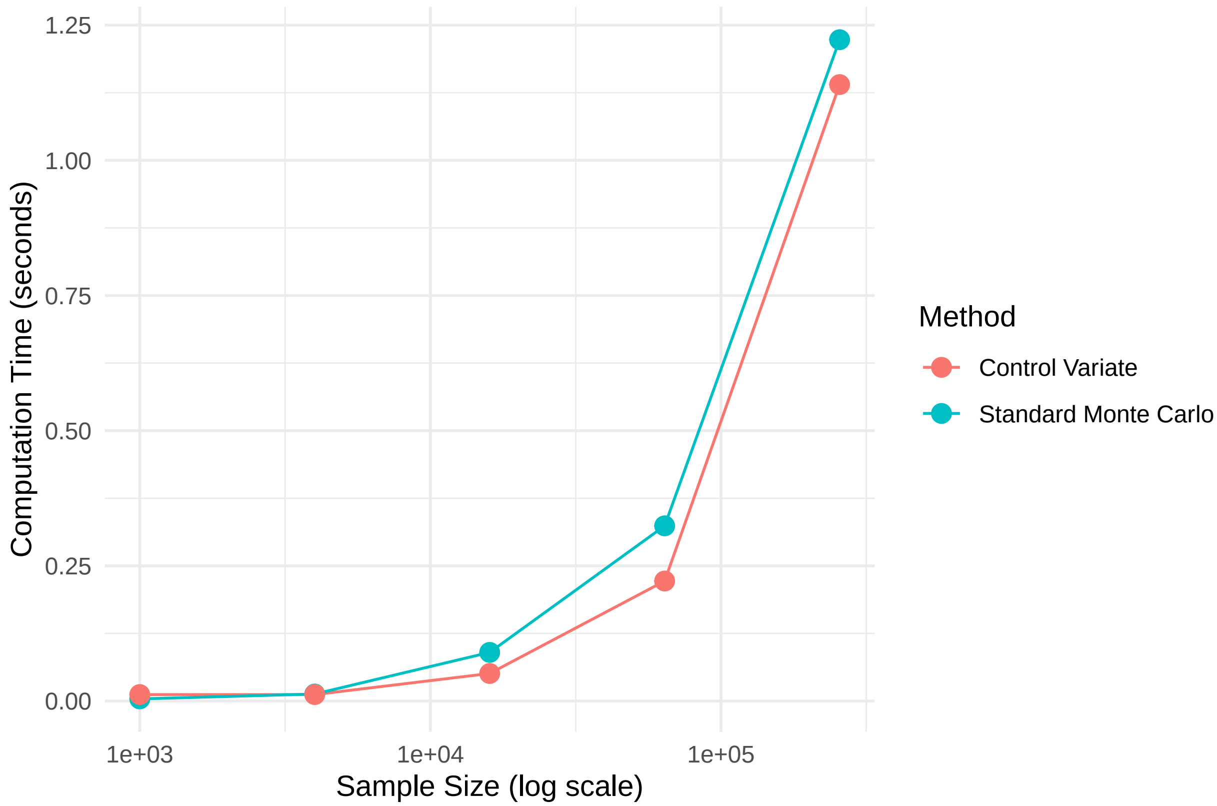



```r
# Plot Standard Error Convergence
ggplot(comparison_results, aes(x = Sample_Size, y = Standard_Error, color = Method, group = Method)) +
  geom_line() +
  geom_point(size = 3) +
  scale_x_log10() +
  scale_y_log10() +
  labs(
    title = "Figure 3.4: Standard Error Convergence: Standard MC vs Control Variate",
    x = "Sample Size (log scale)",
    y = "Standard Error (log-log scale)",
    color = "Method"
  ) +
  theme_minimal() +
  theme(
    plot.title = element_text(
      size = 10,
      hjust = 0.5,
      vjust = 1.2
    )
  )
```




```
# Plot Computation Time Comparison
ggplot(comparison_results, aes(x = Sample_Size, y = Computation_Time_sec, color = Method, group = Metho
  geom_line() +
  geom_point(size = 3) +
  scale_x_log10() +
  labs(
    title = "Figure 3.5: Computation Time: Standard MC vs Control Variate",
    x = "Sample Size (log scale)",
    y = "Computation Time (seconds)",
    color = "Method"
  ) +
  theme_minimal() +
  theme(
    plot.title = element_text(
      size = 10,
      hjust = 0.5,
      vjust = 1.2
    )
  )
```

Figure 3.5: Computation Time: Standard MC vs Control Variate



3.4 Analysis of Results: Comparing Standard Monte Carlo (MC) and Control Variate (CV)

The comparison of the Standard Monte Carlo method and the Control Variate method reveals clear advantages of the latter in terms of convergence rate, variance reduction, and overall efficiency.

1. Option Price Convergence

The Option Price Convergence plot highlights the convergence of option price estimates as the sample size increases. The control variate method demonstrates significantly tighter confidence intervals compared to the standard Monte Carlo method.

For the CV method, the option price estimate stabilizes more rapidly and converges to a consistent value, even at smaller sample sizes. In contrast, the standard MC method requires larger sample sizes to achieve comparable stability. Both methods ultimately converge to approximately the same option price as the sample size increases ($N = 256,000$), confirming the consistency of their final estimates.

The tighter confidence intervals produced by the CV method highlight its ability to improve precision, particularly when computational resources are limited or fewer samples are used.

2. Standard Error Convergence

The Standard Error Convergence plot examines the convergence of standard error on a log-log scale. Both methods exhibit the expected Monte Carlo convergence rate of $O(1/\sqrt{N})$, as evidenced by the linear trends in the plot. However, the CV method consistently achieves lower standard errors compared to the standard MC method at all sample sizes.

For instance: at $N = 1000$, the CV method achieves a standard error more than 10 times smaller than the standard MC method. Even as N increases to $256,000$, the CV method maintains a significant advantage in standard error reduction.

This reduction in standard error reflects the CV method's effectiveness in reducing variance, which is critical in applications requiring high precision. The CV technique allows for comparable accuracy with far fewer samples, making it computationally more efficient than standard MC.

3. Computation Time

The third plot compares the computation times for the two methods. As expected, the computation time increases linearly with sample size for both approaches. The CV method incurs slightly higher computational costs compared to standard MC, reflecting the additional calculations required for the control variate adjustment. For example, at $N = 256,000$, the computation time for CV is marginally higher than that of standard MC.

Despite this minor overhead, the significant reduction in variance achieved by the CV method more than compensates for the slightly higher computation cost. This trade-off makes the CV technique highly effective in scenarios where variance reduction is a priority.

3.5 Further Discussion on Variance Reduction Techniques

In addition to the control variate method, variance reduction can be further enhanced by combining it with other techniques such as stratified sampling and the Brownian bridge construction. These methods introduce additional refinements to the Monte Carlo simulation process, addressing specific sources of randomness and variability in the simulation.

Stratified sampling divides the sample space into non-overlapping strata and ensures that an equal number of samples are drawn from each stratum. This method reduces variance by minimizing randomness in the sampling process.

Mathematically, the variance of the stratified sampling estimator can be expressed as:

$$\mathrm{Var}(\hat{\theta}_{\text{stratified}}) = \sum_{k=1}^{L} \frac{w_k^2}{n_k} \mathrm{Var}(\theta_k),$$

where: - L is the number of strata, - w_k is the weight of the k-th stratum, - n_k is the number of samples in the k-th stratum, - $\mathrm{Var}(\theta_k)$ is the variance within the k-th stratum.

By ensuring that n_k is proportional to w_k, stratified sampling can achieve a lower variance compared to direct sampling. A common application of stratified sampling in Monte Carlo simulations is in the context of Brownian motion, where stratification is often applied to the first step of the simulation, Z_1, which introduces the most variability in the path. This step simplifies the implementation and ensures that the variability in the simulation is minimized from the outset. However, stratified sampling has its limitations. It is computationally more expensive than standard Monte Carlo sampling because it requires the sample space to be divided and the simulation to be carefully managed within each stratum.

The Brownian bridge construction is an effective technique for variance reduction that refines the simulation of Brownian paths by interpolating intermediate points based on the known values at the start and end of the process. By introducing conditional dependencies among simulated points, this method ensures consistency with the overall process, making it particularly useful when the terminal value of the asset significantly influences the option payoff. Moreover, the Brownian bridge is computationally efficient for high-dimensional problems, as it reduces the number of independently generated random variables and complements the control variate method by improving the accuracy of simulated paths.

Despite its advantages, the implementation of the Brownian bridge introduces added complexity. Interpolation of intermediate points and adjustments to the simulation framework require careful execution.

Additionally, the variance reduction achieved depends on the characteristics of the option and the underlying asset paths, and in some cases, the benefits may be marginal relative to the additional implementation effort.

3.6 Variance Reduction Techniques– Moment Matching

The Moment Matching (MM) method is a variance reduction technique used to improve the convergence and accuracy of Monte Carlo simulations. By adjusting the random samples to match the theoretical properties of the underlying distribution, MM ensures that the simulated paths conform more closely to the model assumptions.

Mathematical Formulation Let $Z_{ij} \sim \mathcal{N}(0,1)$ be the random samples used to simulate the asset price paths, where Z_{ij} represents the j-th time step in the i-th simulated path. For Z, we define:

1. Empirical mean:
$$\hat{\mu} = \frac{1}{N \cdot m} \sum_{i=1}^{N} \sum_{j=1}^{m} Z_{ij},$$
where N is the number of simulated paths, and m is the number of time steps.
2. Empirical variance:
$$\hat{\sigma}^2 = \frac{1}{N \cdot m - 1} \sum_{i=1}^{N} \sum_{j=1}^{m} (Z_{ij} - \hat{\mu})^2.$$

The goal of Moment Matching is to adjust Z such that:

$$\text{Mean of Adjusted } Z = 0, \quad \text{Variance of Adjusted } Z = 1.$$

Steps for Adjustment

1. **Centralization (Zero Mean)**: Subtract the empirical mean from each sample to centralize the data:
$$Z'_{ij} = Z_{ij} - \hat{\mu}.$$
2. **Normalization (Unit Variance)**: Divide each sample by the empirical standard deviation:
$$Z''_{ij} = \frac{Z'_{ij}}{\hat{\sigma}}.$$

After adjustment, the modified samples Z''_{ij} satisfy:

$$\text{Mean of } Z'' = 0, \quad \text{Variance of } Z'' = 1.$$

Why Moment Matching Works

1. **Variance Reduction**: By enforcing theoretical mean and variance, MM reduces the noise introduced by random sampling errors, leading to faster convergence.
2. **Improved Accuracy**: MM ensures that the simulated random variables match the assumed distribution properties more closely, improving the precision of Monte Carlo estimates.

Below is the implementation of the Moment Matching algorithm:

```r
# Moment Matching Function
moment_matching <- function(Z) {
  # Step 1: Centralize the samples (adjust mean to 0)
  Z_centered <- Z - rowMeans(Z)

  # Step 2: Normalize the samples (adjust variance to 1)
  Z_normalized <- Z_centered / apply(Z_centered, 1, sd)

  # Return the adjusted random samples
  return(Z_normalized)
}
```

The `moment_matching` function ensures that the random samples used in the simulation have properties consistent with a standard normal distribution. Specifically, it adjusts the samples so that each simulated path has a mean of 0 and a variance of 1.

The function works in two steps:

1. **Centralization**: Each row of the random matrix is adjusted by subtracting the row mean. This ensures that the average value of the samples is zero.
2. **Normalization**: After centralization, each row is scaled by its standard deviation. This ensures that the variance of the samples is equal to one.

These adjustments reduce noise introduced by sampling and help the Monte Carlo simulation converge more efficiently. By aligning the samples with the theoretical distribution, the results become more stable and accurate, especially for smaller sample sizes.

```r
# Asian Call Option Pricing with Moment Matching
price_asian_call_MM <- function(S0, K, T, r, q, sigma, m, N) {
  start_time <- proc.time()
  delta_t <- T / m
  sqrt_delta_t <- sqrt(delta_t)

  # Generate random numbers and apply Moment Matching
  Z <- matrix(rnorm(N * m), nrow = N, ncol = m)
  Z <- moment_matching(Z)

  # Simulate asset price paths
  drift <- (r - q - 0.5 * sigma^2) * delta_t
  diffusion <- sigma * sqrt_delta_t * Z
  W <- t(apply(diffusion, 1, cumsum))
  log_S <- log(S0) + outer(rep(1, N), drift * (1:m)) + W
  S <- exp(log_S)

  # Calculate option price
  S_bar <- rowMeans(S)
  payoffs <- pmax(S_bar - K, 0)
  discounted_payoffs <- exp(-r * T) * payoffs
  option_price <- mean(discounted_payoffs)
  std_error <- sd(discounted_payoffs) / sqrt(N)
  CI_lower <- option_price - 1.96 * std_error
  CI_upper <- option_price + 1.96 * std_error
```

```
  comp_time <- (proc.time() - start_time)[["elapsed"]]

  return(list(
    N = N,
    Option_Price = round(option_price, 2),
    Standard_Error = round(std_error, 5),
    Confidence_Interval = c(round(CI_lower, 2), round(CI_upper, 2)),
    Computation_Time_sec = round(comp_time, 4)
  ))
}
```

The above `price_asian_call_MM` function implements the Monte Carlo simulation for pricing an Asian call option using the moment matching method to improve accuracy and efficiency. The function consists of three main steps: generating random numbers, simulating asset price paths, and calculating the option price.

Step 1: Generate Random Numbers with Moment Matching The function first generates a matrix of standard normal random numbers Z with dimensions $N \times m$, where N is the number of simulated paths, and m is the number of time steps. The `moment_matching` function is applied to adjust the random numbers so that:

$$\text{Mean of Adjusted } Z = 0, \quad \text{Variance of Adjusted } Z = 1.$$

This adjustment ensures that the simulated random samples align more closely with the theoretical properties of the standard normal distribution, reducing noise and improving convergence.

Step 2: Simulate Asset Price Paths The adjusted random numbers are used to simulate asset price paths under the Geometric Brownian Motion (GBM) model. The GBM model for the stock price S_t is given by:

$$S_t = S_0 \exp\left((r - q - 0.5\sigma^2)t + \sigma W_t\right),$$

where: - S_0: Initial stock price, - r: Risk-free interest rate, - q: Dividend yield, - σ: Volatility, - W_t: Standard Brownian motion.

The function uses the drift and diffusion terms to construct the log-transformed price:

$$\log(S_t) = \log(S_0) + (r - q - 0.5\sigma^2)t + \sigma W_t,$$

and then converts the log prices back to the original price domain:

$$S_t = \exp(\log(S_t)).$$

Step 3: Calculate Option Price For each simulated path, the arithmetic average of the asset prices is computed:

$$\bar{S} = \frac{1}{m} \sum_{i=1}^{m} S_{t_i}.$$

The payoff of the Asian call option is then calculated as:

$$\text{Payoff}_i = \max(\bar{S} - K, 0),$$

where K is the strike price. The present value of the payoff is obtained by discounting it at the risk-free rate:

$$\text{Discounted Payoff}_i = e^{-rT} \cdot \text{Payoff}_i.$$

The option price is estimated as the average of the discounted payoffs:

$$\text{Option Price} = \frac{1}{N} \sum_{i=1}^{N} \text{Discounted Payoff}_i.$$

Statistical Outputs To assess the accuracy of the estimate, the function computes the standard error:

$$\text{Standard Error} = \frac{\text{Standard Deviation of Payoffs}}{\sqrt{N}},$$

and constructs a 95% confidence interval for the option price:

$$\text{Confidence Interval} = [\text{Option Price} \pm 1.96 \times \text{Standard Error}] .$$

The function also measures the computation time to evaluate performance. By reducing the variability in the random samples, the moment matching method ensures faster convergence of the Monte Carlo simulation and provides more stable results, especially for small sample sizes.

Testing the Moment Matching Method

To evaluate the performance of the moment matching method, we test the `price_asian_call_MM` function using different sample sizes (N) and compare the results. The table below summarizes the option price, standard error, 95% confidence interval, and computation time for each sample size.

```r
# Parameters for the Asian call option
S0 <- 100      # Initial stock price
K <- 100       # Strike price
T <- 1         # Time to maturity (in years)
r <- 0.10      # Risk-free rate
q <- 0         # Dividend yield
sigma <- 0.20  # Volatility
m <- 50        # Number of monitoring points

# Define sample sizes for testing
sample_sizes <- c(1000, 4000, 16000, 64000, 256000)

# Initialize a data frame to store results
moment_matching_results <- data.frame(
  Sample_Size = numeric(),
  Option_Price = numeric(),
  Standard_Error = numeric(),
  CI_Lower = numeric(),
  CI_Upper = numeric(),
  Computation_Time_sec = numeric(),
  stringsAsFactors = FALSE
)

# Loop through different sample sizes and compute results
for (N in sample_sizes) {
  result <- price_asian_call_MM(S0, K, T, r, q, sigma, m, N)
  moment_matching_results <- rbind(moment_matching_results, data.frame(
    Sample_Size = N,
    Option_Price = result$Option_Price,
    Standard_Error = result$Standard_Error,
    CI_Lower = result$Confidence_Interval[1],
    CI_Upper = result$Confidence_Interval[2],
    Computation_Time_sec = result$Computation_Time_sec
  ))
}
```

Table 3.3: Moment Matching Simulation Results

Sample_Size	Option_Price	Standard_Error	CI_Lower	CI_Upper	Computation_Time_sec
1000	4.82	0.14381	4.54	5.10	0.009
4000	4.67	0.07066	4.53	4.80	0.031
16000	4.84	0.03611	4.77	4.91	0.123
64000	4.77	0.01795	4.74	4.81	0.536
256000	4.78	0.00899	4.76	4.80	2.319

The above code tests the performance of the moment matching method by evaluating the option price under different sample sizes N. The `price_asian_call_MM` function simulates the stock price paths based on the Geometric Brownian Motion model using moment matching to adjust the random samples. Specifically, the simulation begins with generating a matrix of standard normal random variables Z, with dimensions $N \times m$, where N is the number of paths and m is the number of time steps. Moment matching ensures the random samples have a mean of zero and variance of one to align with the standard normal distribution. These adjusted samples are then used to simulate the asset prices S_t at each time step. The arithmetic average of the asset prices for each path is computed, and the option payoff is calculated as $\max(\bar{S} - K, 0)$, where $\bar{S}$ is the average price and K is the strike price. The payoff is then discounted to the present value using e^{-rT}, where r is the risk-free interest rate and T is the time to maturity. The function outputs the average option price, standard error, 95% confidence interval, and computation time.

The results in the table demonstrate the convergence of the Monte Carlo simulation with moment matching as the sample size increases. At $N = 1000$, the option price estimate is 4.82 with a relatively large standard error of 0.14381, and the confidence interval is wide, ranging from 4.54 to 5.10. As the sample size increases to $N = 4000$, the option price drops to 4.67, and the standard error decreases to 0.07066, showing a noticeable improvement in precision. With $N = 16000$, the option price estimate stabilizes at 4.84, and the standard error further reduces to 0.03611, with a narrower confidence interval of [4.77, 4.91]. When the sample size increases to $N = 64000$, the option price is 4.77, and the standard error is significantly reduced to 0.01795, confirming the expected improvement in accuracy as N increases. Finally, for $N = 256000$, the option price converges to 4.78, with a very small standard error of 0.00899 and a confidence interval of [4.76, 4.80].

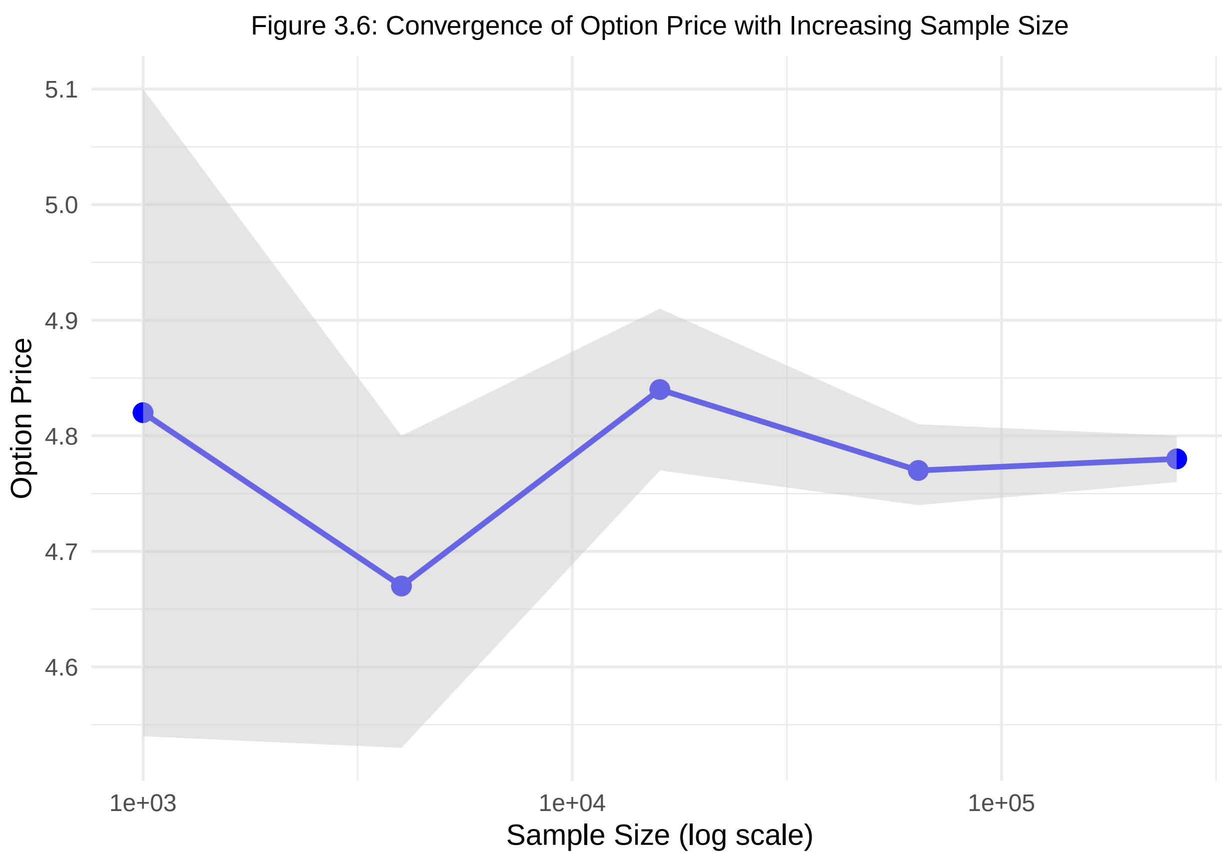
The computation time also increases with the sample size, from 0.008 seconds for $N = 1000$ to 2.416 seconds for $N = 256000$. This behavior is expected since larger sample sizes require more calculations. The results indicate that the moment matching method provides stable and accurate estimates of the Asian call option price. As the sample size increases, the standard error decreases approximately at the rate of $O(1/\sqrt{N})$, which is consistent with the theoretical behavior of the Monte Carlo method. The final estimate stabilizes around 4.78, suggesting good convergence of the simulation. Overall, the moment matching approach effectively improves the reliability of the Monte Carlo method by reducing sampling variability and enhancing the precision of the results.

To visualize the convergence of the option price as the sample size increases, we plot the option price estimates along with their 95% confidence intervals.

```r
library(ggplot2)

# Plot Option Price Convergence
ggplot(moment_matching_results, aes(x = Sample_Size, y = Option_Price)) +
  geom_line(color = "blue", size = 1) +
  geom_point(size = 3, color = "blue") +
  geom_ribbon(aes(ymin = CI_Lower, ymax = CI_Upper), fill = "grey80", alpha = 0.5) +
  scale_x_log10() +
  labs(
    title = "Figure 3.6: Convergence of Option Price with Increasing Sample Size",
```

```r
    x = "Sample Size (log scale)",
    y = "Option Price"
  ) +
  theme_minimal()+
  theme(
    plot.title = element_text(
      size = 10,
      hjust = 0.5,
      vjust = 1.2
    )
  )
```



The plot illustrates the convergence of the Asian call option price as the sample size increases. At smaller sample sizes (e.g., $N = 1000$), the option price estimate shows noticeable variability, with a wider confidence interval. This reflects higher uncertainty due to insufficient paths in the Monte Carlo simulation. As the sample size increases to $N = 16000$ and beyond, the option price stabilizes around 4.78, and the confidence interval narrows significantly. This behavior is consistent with the theoretical property of Monte Carlo methods, where the standard error decreases at a rate of $O(1/\sqrt{N})$. The convergence of the option price and the reduction in uncertainty demonstrate the effectiveness of the moment matching method in improving the reliability of the simulation results.

To analyze the reduction in standard error as the sample size increases, we plot the standard error against the sample size on a log-log scale.

```r
# Plot Standard Error Convergence
ggplot(moment_matching_results, aes(x = Sample_Size, y = Standard_Error)) +
```

```r
geom_line(color = "red", size = 1) +
geom_point(size = 3, color = "red") +
scale_x_log10() +
scale_y_log10() +
labs(
  title = "Figure 3.7: Convergence of Standard Error with Increasing Sample Size",
  x = "Sample Size (log scale)",
  y = "Standard Error (log scale)"
) +
theme_minimal()+
theme(
  plot.title = element_text(
    size = 10,
    hjust = 0.5,
    vjust = 1.2
  )
)
```

The plot shows the convergence of the standard error as the sample size increases, presented on a log-log scale. As expected, the standard error decreases approximately linearly with the logarithm of the sample size, confirming the theoretical rate of convergence for Monte Carlo simulations. Specifically, the standard error follows the relationship:

$$\text{Standard Error} \propto \frac{1}{\sqrt{N}},$$

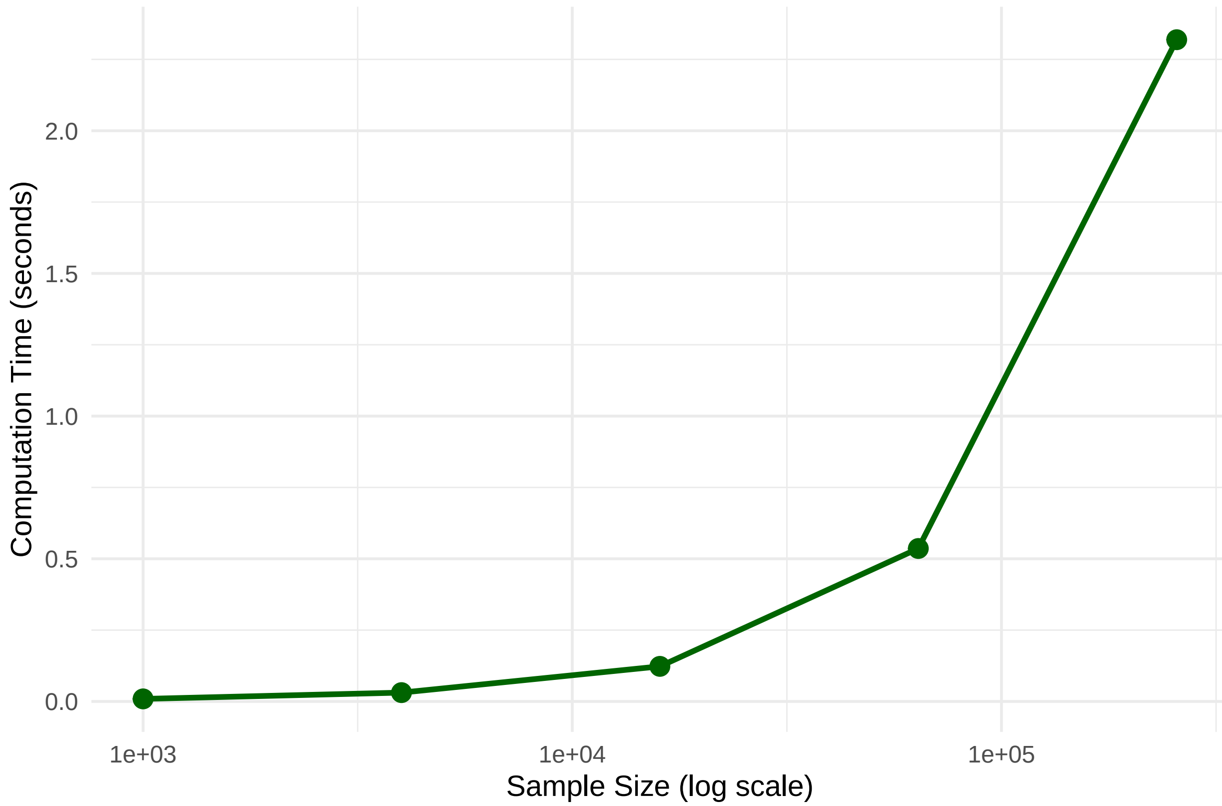
where N is the sample size. This behavior is evident in the graph, as the data points form a straight line on the log-log scale, indicating that the standard error decreases proportionally to $O(1/\sqrt{N})$.

At smaller sample sizes (e.g., $N = 1000$), the standard error is relatively high (above 0.1), reflecting greater variability in the option price estimates. As the sample size increases to $N = 256000$, the standard error reduces significantly to approximately 0.01, demonstrating improved precision. This result highlights the efficiency of increasing the sample size to achieve more accurate estimates, while also validating the effectiveness of the moment matching method in reducing variance in the Monte Carlo simulation.

To analyze the computational cost, we plot the computation time against the sample size to observe how the time scales with increasing sample size.

```r
# Plot Computation Time vs Sample Size
ggplot(moment_matching_results, aes(x = Sample_Size, y = Computation_Time_sec)) +
  geom_line(color = "darkgreen", size = 1) +
  geom_point(size = 3, color = "darkgreen") +
  scale_x_log10() +
  labs(
    title = "Figure 3.8: Computation Time with Increasing Sample Size",
    x = "Sample Size (log scale)",
    y = "Computation Time (seconds)"
  ) +
  theme_minimal()+
  theme(
    plot.title = element_text(
      size = 10,
      hjust = 0.5,
      vjust = 1.2
    )
  )
```

Figure 3.8: Computation Time with Increasing Sample Size



The plot shows the computation time as a function of the sample size on a log scale. As expected, the computation time increases with the sample size. For small sample sizes, such as $N = 1000$, the computation time is negligible, close to 0.008 seconds. As the sample size increases to $N = 16000$, the computation time begins to rise gradually, reaching 0.122 seconds. For larger sample sizes, the computation time increases more significantly, reaching approximately 2.4 seconds for $N = 256000$. This behavior reflects the linear scaling of computation time with the sample size, as each additional path requires extra calculations for the asset price simulation and payoff evaluation.

The sharp increase in computation time at higher sample sizes highlights the trade-off between accuracy and computational cost in Monte Carlo simulations. While increasing the sample size reduces the standard error and improves precision, it also leads to higher computational expenses. The moment matching method efficiently reduces variance, allowing for accurate results even with moderate sample sizes, which helps balance precision and performance.

In this part of the project, we implemented the moment matching method to improve the accuracy and efficiency of Monte Carlo simulations for pricing an Asian call option. By adjusting the random samples to have a mean of zero and a variance of one, the moment matching method ensures that the simulated paths align more closely with the theoretical properties of the standard normal distribution. This adjustment reduces sampling variability and enhances the convergence rate of the simulation.

The results demonstrate that the option price estimates converge as the sample size increases, with the confidence intervals narrowing significantly. For smaller sample sizes, the estimates exhibit greater variability, as shown by wider confidence intervals. However, as the sample size grows, the option price stabilizes around 4.78, and the standard error decreases approximately at the expected rate of $O(1/\sqrt{N})$. This is consistent with the theoretical properties of Monte Carlo methods and confirms the effectiveness of moment matching in variance reduction.

The analysis of computation time reveals a linear relationship between the sample size and the time required for simulation. While larger sample sizes lead to more accurate results, they also result in higher computational costs. For instance, the computation time increases from 0.008 seconds for $N = 1000$ to 2.4 seconds for $N = 256000$. This trade-off highlights the importance of balancing precision and efficiency when selecting an appropriate sample size for practical applications.

Overall, the moment matching method proves to be a robust variance reduction technique, significantly improving the precision of Monte Carlo simulations without introducing excessive computational overhead. By reducing the standard error, it allows accurate estimates to be achieved with moderate sample sizes, making it an efficient tool for pricing complex financial derivatives like Asian options.

Future work could explore combining moment matching with other variance reduction techniques, such as the control variate method or quasi-Monte Carlo approaches, to further enhance simulation efficiency. Additionally, the applicability of moment matching to other exotic options or high-dimensional financial models could be investigated to broaden its practical use in quantitative finance.