Project Report on Monte Carlo Methods for Option Pricing

Aditya Naskar(BSD-CC-2402)| Kolkata Center
Tanish Ghosh(BSD-BG-2420)| Bangalore Center
Md Ali Mumtaz(BSD-CC-2410)| Kolkata Center
Faaz Noushad(BSD-BG-2405)| Bangalore Center

May 13, 2026

Contents

Abstract

Option pricing lies at the core of modern financial mathematics, where the value of a derivative security is determined by the uncertain future behavior of one or more underlying assets. While standard European options can often be priced using closed-form formulas such as the Black–Scholes model, many practically relevant contracts are path-dependent, or otherwise analytically intractable. Examples include barrier options, lookback options and Asian options. In such settings, Monte Carlo simulation offers a flexible and powerful computational framework for estimating option values by generating and averaging a large number of possible price paths. This project studies the financial interpretation of options, the mathematical ideas underlying classical pricing theory, the role of Monte Carlo methods in pricing contracts for which exact formulas are unavailable or impractical and comparisons and convergence analysis between various methods for Monte Carlo Simulations.

1 Introduction

Option pricing is one of the central topics in financial mathematics and computational finance. An *option* is a derivative contract that gives its holder the right, but not the obligation, to buy or sell an underlying asset at a specified price within a specified time period or at maturity. The two basic forms are *call options* and *put options*. A call option gives the right to buy the asset, whereas a put option gives the right to sell it.

The simplest and most widely studied contracts are European options. For a European call option with strike price K and maturity T, the payoff at maturity is

$$(S_T - K)^+ = \max(S_T - K, 0),$$

where S_T denotes the price of the underlying asset at time T. Similarly, the payoff of a European put option is

$$(K - S_T)^+ = \max(K - S_T, 0).$$

These expressions show that option values are fundamentally linked to uncertainty in future asset prices.

For standard European options, the Black–Scholes framework provides a celebrated closed-form pricing formula. However, many contracts encountered in practice are more complex. Barrier options depend on whether the asset price crosses a prescribed level during the life of the option; lookback options depend on the maximum or minimum attained by the asset price; Asian options depend on an average of prices over time. Since these contracts are often path-dependent or multi-dimensional, exact pricing formulas are frequently unavailable or difficult to apply directly.

Monte Carlo simulation provides a natural and flexible alternative. Rather than solving the pricing problem analytically, one generates a large number of possible future price paths, computes the payoff for each path, and then averages the discounted payoffs. This makes Monte Carlo methods especially useful for exotic options, where the payoff depends on the entire trajectory of the asset price instead of only its terminal value.

2 Mathematical and Financial Background

Option pricing is built on a simple market model consisting of a risk-free asset and a risky asset. The risk-free asset, or bond, evolves according to

$$dB_t = rB_t\,dt, \qquad B_t = B_0 e^{rt},$$

where r is the constant risk-free rate. The risky asset, usually interpreted as a stock price, is modeled by

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

where μ is the expected return, σ is the volatility, and W_t is Brownian motion. This model captures both the average growth of the stock price and its random fluctuations.

To determine the value of a derivative contract $V(S,t)$, *Itô*'s formula is applied to relate the evolution of the option price to the evolution of the underlying stock. Under the no-arbitrage principle, one can construct a hedged portfolio whose random component vanishes. This leads to the Black–Scholes partial differential equation

$$\frac{\partial V}{\partial t} + \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 V}{\partial S^2} + rS\frac{\partial V}{\partial S} - rV = 0.$$

This equation states that a fair derivative price must evolve so that a riskless hedged position grows at the risk-free rate.

The PDE must be supplemented by terminal and boundary conditions. For a European call option,

$$V(S,T) = (S-K)^+,$$

and for a European put option,

$$V(S,T) = (K-S)^+.$$

These conditions specify the payoff at maturity. Boundary conditions describe the limiting behavior of the option price when the stock price becomes very small or very large.

For pricing purposes, it is often convenient to work under the risk-neutral drift

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

In this setting, the discounted asset price does not allow arbitrage, and the option price can be estimated by averaging discounted payoffs. This is the basis for Monte Carlo pricing.

3 Option Types & Payoff Structures

3.1 European Options

European options are the simplest contracts and serve as the baseline for the project.

For a European call,

$$\text{Payoff} = (S_T - K)^+.$$

For a European put,

$$\text{Payoff} = (K - S_T)^+.$$

These payoffs depend only on the terminal asset price, which is why they are the easiest to analyze.

3.2 Asian Options

Asian options are based on an average price rather than the terminal price alone. If the asset is observed at times $t_1, \ldots, t_n$, the arithmetic average is

$$\bar{S} = \frac{1}{n} \sum_{i=1}^{n} S(t_i).$$

A standard Asian call option has payoff

$$(\bar{S} - K)^+,$$

while an Asian put has payoff

$$(K - \bar{S})^+.$$

Asian options may be monitored discretely or continuously. In practice, discrete monitoring is often used because it is easier to implement numerically. Since the payoff depends on a time average, Monte Carlo simulation is particularly well suited for valuation.

3.3 Barrier Options

Barrier options are path-dependent contracts whose payoff depends on whether the asset price reaches a prescribed level during the contract period. Let b denote the barrier.

For a down-and-out option,

$$\text{Payoff} = (S_T - K)^+ \mathbf{1} \left\{ \min_{0 \le t \le T} S_t > b \right\}.$$

For a down-and-in option,

$$\text{Payoff} = (S_T - K)^+ \mathbf{1} \left\{ \min_{0 \le t \le T} S_t \le b \right\}.$$

Similarly, an up-and-out option can be written as

$$\text{Payoff} = (S_T - K)^+ \mathbf{1} \left\{ \max_{0 \le t \le T} S_t < b \right\},$$

and an up-and-in option as

$$\text{Payoff} = (S_T - K)^+ \mathbf{1} \left\{ \max_{0 \le t \le T} S_t \ge b \right\}.$$

The indicator function simply checks whether the barrier condition has been satisfied. If the specified event occurs, the indicator equals 1; otherwise it equals 0. In practical terms, the option either remains alive or becomes inactive depending on whether the price path crosses the barrier.

Barrier options may be monitored continuously or at discrete observation times. Continuous monitoring checks the barrier at every instant, whereas discrete monitoring checks only at selected dates. This distinction is important because a path may cross the barrier between two observation times without being detected in a discrete scheme.

3.4 Lookback Options

Lookback options depend on the extreme values attained by the underlying asset over the entire life of the contract. They are therefore strongly path-dependent.

A fixed-strike lookback call may be written as

$$\text{Payoff} = \left(\max_{0 \le t \le T} S_t - K \right)^+,$$

while a fixed-strike lookback put may be written as

$$\text{Payoff} = \left(K - \min_{0\le t\le T} S_t\right)^+.$$

In floating-strike versions, the strike itself is replaced by an extreme value of the path, such as

$$S_T - \min_{0\le t\le T} S_t \quad \text{or} \quad \max_{0\le t\le T} S_t - S_T.$$

The main idea is that the contract depends on the full trajectory of the stock price, not merely on its final value.

4 Monte Carlo for Pricing Options and The Big Picture

Option pricing is about valuing a contract (*option*) today by looking at many possible future outcomes of the Stocks and asking what the contract is worth on average. This is where Monte Carlo simulation enters the picture. Under a standard no-arbitrage pricing idea, the fair value of an option today is the expected payoff at maturity discounted to today. Monte Carlo is used to approximate that expectation by simulating many possible future stock paths. Additionally, for European options, closed-form pricing may be available, but for barrier, lookback and Asian options, the payoff depends on the path itself. In such cases, simulation is often the most practical approach to proceed.

4.1 The main idea

Suppose an option expires at time T and has payoff Payoff at T. If we simulate many possible future stock paths, then for each simulated path we can calculate the payoff of the option at maturity. If the simulated payoffs are $\text{Payoff}_1, \text{Payoff}_2, \ldots, \text{Payoff}_N$, then the Monte Carlo estimate of the price of the option today is

$$V_0 \approx e^{-rT} \frac{1}{N} \sum_{i=1}^{N} \text{Payoff}_i.$$

Here: - V_0 is the price of the option today, - r is the risk-free interest rate, - T is the time of maturity, - the factor e^{-rT} discounts the future average payoff back to the present.

This discounted average payoff is interpreted as the **fair price** of the option. In other words, the price paid today for the option contract is the amount that makes the contract fair under the no-arbitrage principle. This can be understood as, what amount V_0 should one invest in bank today, where it gets compounded over time, so that at maturity T, the compounded amount $V_0 e^{rT}$ is equal to expected Payoff. This is why we need to discount the expected Payoff to today to get the Fair Price of the Option today.

4.2 Call option example: profit case

Suppose: - current stock price is $S_0 = 100$, - strike price is $K = 100$, - current option price is 8, - maturity stock price is $S_T = 130$.

For a call option, the payoff is

$$(S_T - K)^+ = (130 - 100)^+ = 30.$$

The net profit is

$$\text{Net profit} = \text{Payoff} - \text{Price of Option} = 30 - 8 = 22.$$

Here the buyer makes a profit if they exercise the option.

4.3 Call option example: loss case

Now suppose the same contract but the maturity stock price is $S_T = 90$.

Then the call payoff is

$$(S_T - K)^+ = (90 - 100)^+ = 0.$$

The net profit is

$$\text{Net profit} = 0 - 8 = -8.$$

Here the buyer may not exercise the option as, they can directly buy the stock at a lower price from the market. The loss is limited to the price of the option paid upfront to the dealer.

5 Monte Carlo Theory & Methodology

5.1 Monte Carlo Principle

Monte Carlo methods provide a numerical approach for approximating expectations by means of random sampling. In the context of option pricing, the fundamental objective is to compute the expected value of a payoff that depends on the future evolution of an asset price.

Let X denote the random payoff of an option at maturity. The expected value $\mathbb{E}[X]$ can be approximated using a finite number of simulated samples:

$$\mathbb{E}[X] \approx \frac{1}{N} \sum_{i=1}^{N} X_i$$

where $X_1, X_2, \ldots, X_N$ are independent realizations of the payoff obtained from simulated asset price paths. As the number of simulations N increases, the approximation converges to the true expected value by the Law of Large Numbers.

The following simple example demonstrates how sample averages converge to the true mean.

```
set.seed(123)
N <- 10000
samples <- rnorm(N, mean = 0, sd = 1)

mean(samples)   # should be close to 0
```

```
## [1] -0.002371702
```

```
var(samples)    # should be close to 1
```

```
## [1] 0.9972751
```

5.2 Brownian Motion Simulation Methods

To simulate asset prices, it is necessary to model the underlying source of randomness. This is achieved using Brownian Motion, also known as the Wiener process.

A stochastic process $\{W(t), t \geq 0\}$ is called a standard Brownian Motion if it satisfies the following properties:

- $W(0) = 0$ almost surely.
- It has increments independent of the past *i.e.* $W(t+u) - W(t)$ independent of $W(s)$ for $s < t$.
- The increments are normally distributed, i.e.,

$$W(t) - W(s) \sim \mathcal{N}(0, t-s), \quad t > s$$

- The mapping $t \to W(t)$ is a continuous function on $[0, T]$ almost surely.

Brownian Motion provides a mathematical representation of continuous random fluctuations and forms the basis for modeling uncertainty in financial systems.

We can write a stochastic process $X(t) = x + \mu t + \sigma W(t)$, where $W(t)$ is a standard Brownian Motion and x is intial value, μ is drift and σ is volatility. This can also be written as $X(t) \sim N(x + \mu t, \sigma^2 t)$.

One can get this $X(t)$ by solving the *SDE* (stochastic differential equation) $dX(t) = \mu dt + \sigma dW(t)$. But we could model the stock price by a deterministic time-varying $\mu(t)$ and $\sigma(t) > 0$, which gives us a SDE $dX(t) = \mu(t)dt + \sigma(t)dW(t)$, whose solution is $X(t) = X(0) + \int_0^t \mu(u)du + \int_0^t \sigma(s)dW(s)$, solving it further we get $X(t) \sim N(X(0) + \int_0^t \mu(u)du, \int_0^t \sigma^2(s)ds)$ with $X(t)$ having *a.s.* continuous paths and independent increments.

Random Walk Construction

To simulate $\vec{B} = (B(t_1), \cdots, B(t_n))$ from a standard Brownian Motion at fixed points $t_1 < \cdots < t_n$, notice that $\vec{B}$ can be written as a linear transformation of the random vector $(B(t_1), B(t_2) - B(t_1), \cdots, B(t_n) - B(t_{n-1}))$ of increments which we know are independent and normally distributed. Hence, $\vec{B}$ has a multivariate normal distribution. The mean and the covariance matrix of this random vector are $\vec{0}$ and $C = ((C_{i,j}))$, where $C_{i,j} = min(t_i, t_j)$, respectively. What we want now is to simulate $\vec{B}$ from $\vec{Z}$ (standard normal vector), if we can somehow express $\vec{B}$ as $A\vec{Z}$ such that $A\vec{Z} \sim \mathcal{N}(\vec{0}, AA^T) \sim \vec{B}$ then we are done. So, we can obtain the Cholesky decomposition of $C = AA^T$. Here, it turns out that

$$A = \begin{pmatrix} \sqrt{t_1} & 0 & \cdots & 0 \\ \sqrt{t_1} & \sqrt{t_2 - t_1} & \cdots & 0 \\ \cdot & \cdot & & \cdot \\ \sqrt{t_1} & \sqrt{t_2 - t_1} & \cdots & \sqrt{t_n - t_{n-1}} \end{pmatrix}$$

Then,

$$B(t_i) = A^{(i^{th} row)}\vec{Z} = \sqrt{t_1}Z_1 + \sqrt{t_2 - t_1}Z_2 + \cdots + \sqrt{t_i - t_{i-1}}Z_i$$

For computational purposes, standard Brownian Motion can be written in discrete time using increments of the form (using the above):

$$B(t_i) - B(t_{i-1}) = \sqrt{t_i - t_{i-1}}Z_i, \text{ Or,}$$
$$\Delta B = \sqrt{\Delta t} \cdot Z, \quad Z \sim \mathcal{N}(0, 1)$$

But we are rather interested in simulating general Brownian Motion X with drift μ and volatility σ at fixed times $0 = t_1 < \cdots < t_n$. Now $X(t_i) = X(0) + \mu t_i + \sigma B(t_i)$. This can be written in the following form, for simulating purpose:

$$X(t_i) = X(t_{i-1}) + \mu(t_i - t_{i-1}) + \sigma\sqrt{t_i - t_{i-1}}Z_i, \ i = 1, \cdots, n$$

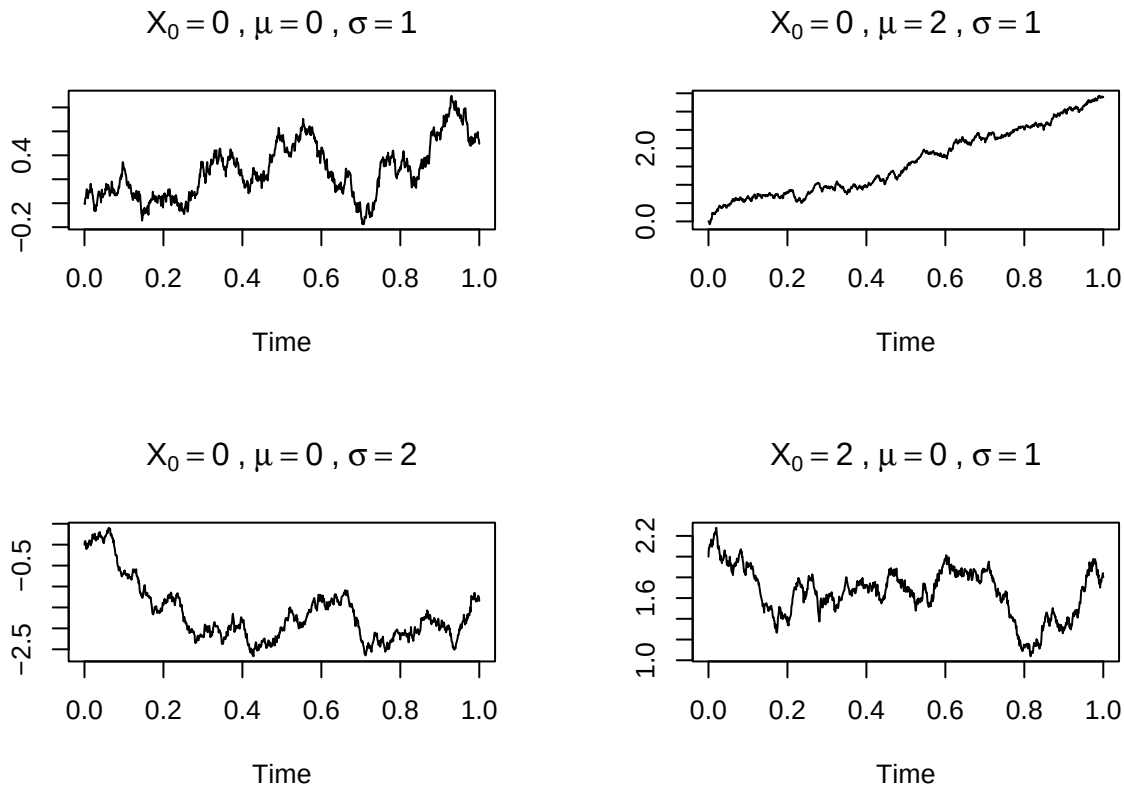
In case for time dependent parameters $\mu(t)$ and $\sigma(t)$, one can obtain:

$$X(t_i) = X(t_{i-1}) + \int_{t_{i-1}}^{t_i} \mu(s)ds + \left(\sqrt{\int_{t_{i-1}}^{t_i} \sigma^2(s)ds}\right) Z_i, \ i = 1, \cdots, n.$$

One can also obtain Euler approximation to this (but it will introduce discretization error):

$$X(t_i) = X(t_{i-1}) + \mu(t_{i-1})(t_i - t_{i-1}) + \sigma(t_{i-1})\sqrt{t_i - t_{i-1}}Z_i, \ i = 1, \cdots, n.$$

```r
set.seed(123)

sim.Brownian.motion = function(X0, t, mu, sigma){
  n = length(t)
  X = rep(NA, n)
  Z = rnorm(n, mean=0, sd=1)
  X[1] = X0 + sigma*sqrt(t[1])*Z[1] + mu*t[1]
  for(i in 2:n){
    X[i] = X[i-1] + sigma*sqrt(t[i]-t[i-1])*Z[i] + mu*(t[i]-t[i-1])
  }
  return(X)
}

t = seq(0, 1, 0.001)

par(mfrow = c(2,2))

# Top Left: X0=0, mu=0, sigma=1
X1 = sim.Brownian.motion(X0=0, t=t, mu=0, sigma=1)
plot(ts(X1, start=0, frequency=1000), ylab="",
     main=expression(X[0]==0~","~mu==0~","~sigma==1))

# Top Right: X0=0, mu=2, sigma=1
X2 = sim.Brownian.motion(X0=0, t=t, mu=2, sigma=1)
plot(ts(X2, start=0, frequency=1000), ylab="",
     main=expression(X[0]==0~","~mu==2~","~sigma==1))

# Bottom Left: X0=0, mu=0, sigma=2
X3 = sim.Brownian.motion(X0=0, t=t, mu=0, sigma=2)
plot(ts(X3, start=0, frequency=1000), ylab="",
     main=expression(X[0]==0~","~mu==0~","~sigma==2))

# Bottom Right: X0=2, mu=0, sigma=1
X4 = sim.Brownian.motion(X0=2, t=t, mu=0, sigma=1)
plot(ts(X4, start=0, frequency=1000), ylab="",
     main=expression(X[0]==2~","~mu==0~","~sigma==1))
```




```
par(mfrow = c(1,1))
```

5.3 Brownian Bridge Construction

So far we have seen a sequential (random walk) construction of Brownian Motion. We may however generate the $B(t_i)$ in any order we choose, provided that at each step we sample from the correct conditional distribution given the values already generated. Because of the joint normal distribution of finite dimensional marginals of Brownian motion, the conditional distribution needed at each step is itself normal and this makes conditional sampling feasible.

Conditioning a Brownian motion on its endpoints produces a Brownian bridge. Suppose $0 < u < s < t$ and consider the problem of generating $B(s)$ conditional on $B(u) = x$ and $B(t) = y$. The joint distribution is given by

$$\begin{pmatrix} B(u) \\ B(s) \\ B(t) \end{pmatrix} \sim \mathcal{N}_3 \left(0, \begin{pmatrix} u & u & u \\ u & s & s \\ u & s & t \end{pmatrix} \right).$$

The conditional distribution of the bridge at an intermediate time t given $B(s) = b_s$ and $B(T) = b_T$ is:

$$(B(s) \mid B(u) = x,\, B(t) = y) \;\sim\; \mathcal{N}\left(\frac{(t-s)x + (s-u)y}{t-u},\; \frac{(s-u)(t-s)}{t-u} \right)$$

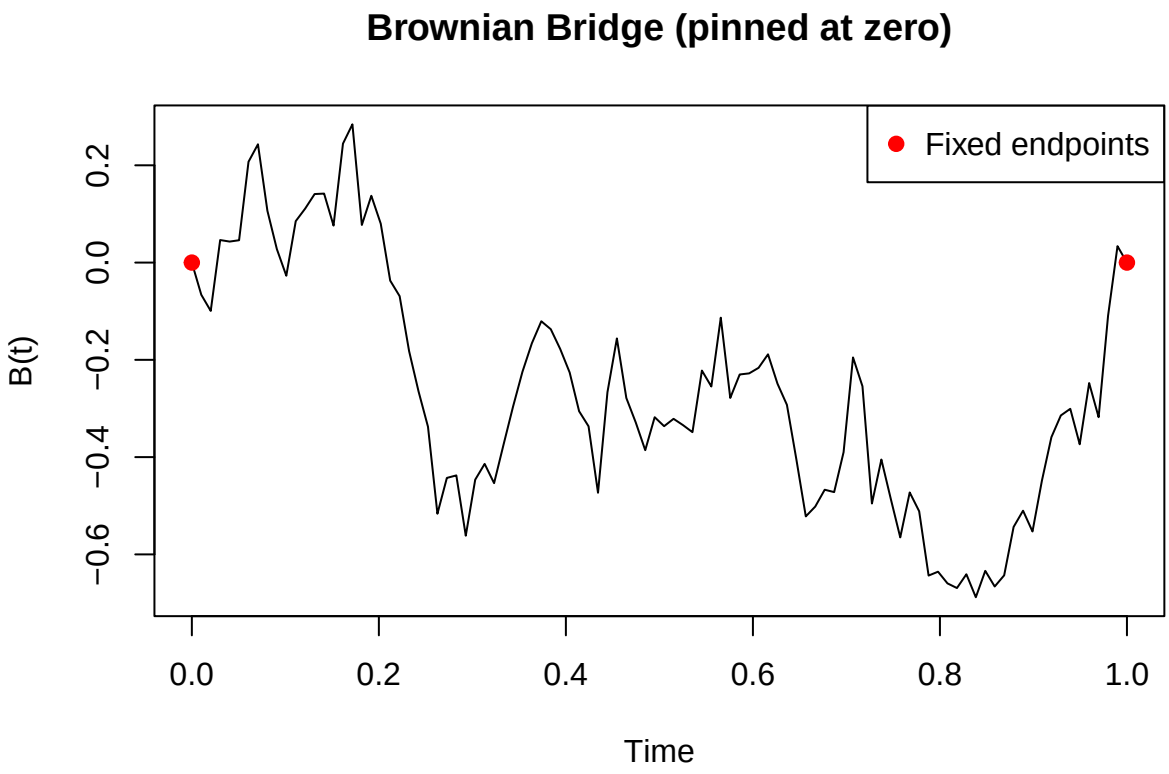
The mean interpolates linearly between the known endpoints, while the variance is maximized at the midpoint and vanishes at both ends.

More generally, if the values $B(s_1) = x_1, B(s_2) = x_2, \cdots, B(s_k) = x_k$ of the Brownian path have been determined at the times $s_1 < \cdots < s_k$ and we wish to sample $B(s)$ conditional on these values. Suppose

that $s_i < s < s_{i+1}$, then, due to Markov property of Brownian motion, we have $(B(s)|B(s_j) = x_j, j = 1, \cdots, k) = (B(s)|B(s_i) = x_i, B(s_{i+1}) = x_{i+1})$. Thus, conditioning on all $B(s_j)$ is equivalent to conditioning on the values of the Brownian path at the two times closest to s. By repeatedly using these observations, we may sample the components of the vector $(B(t_1), \cdots, B(t_n))$ in any order.

This construction can be useful in option pricing for estimating the probability of crossing a threshold such as in case of Barrier Options, we shall discuss this further under the Geometric Brownian Motion simulation.

```r
set.seed(123)

T       <- 1
n_steps <- 100
dt      <- T / n_steps
t_grid  <- seq(0, T, length.out = n_steps)

# Simulate a standard Brownian path
Z <- rnorm(n_steps - 1)
W <- c(0, cumsum(sqrt(dt) * Z))

# Fix terminal value and construct the bridge
W_T <- W[n_steps]
BB  <- W - (t_grid / T) * W_T   # B(t) = W(t) - (t/T)*W(T)

plot(t_grid, BB, type = "l",
     main = "Brownian Bridge (pinned at zero)",
     xlab = "Time", ylab = "B(t)")
points(c(0, T), c(0, 0), pch = 19, col = "red")
legend("topright", legend = "Fixed endpoints", pch = 19, col = "red")
```



5.4 Principal Components Construction

As noted in the Brownian Motion simulation, the vector of discretized path values

$$\vec{B} = (B(t_1), \ldots, B(t_n)) \sim \mathcal{N}_n(\vec{0}, C)$$

where $C_{i,j} = \min(t_i, t_j)$. We can use the method of principal components to simulate from standard multivariate normal vector $\vec{Z}$. If $C = A\Lambda A^\top$ where $\Lambda = \text{diag}(\lambda_i)$, then

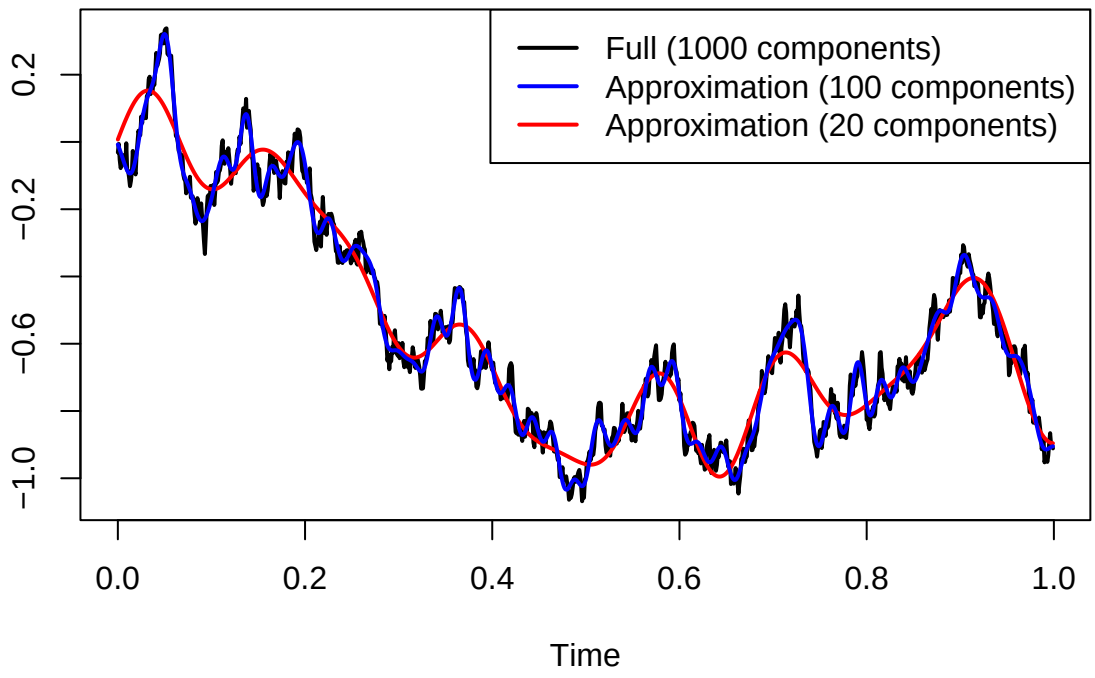
$$\vec{B} = A\Lambda^{1/2}\vec{Z} = \sum_{i=1}^{n} \vec{a_i}\sqrt{\lambda_i}Z_i$$

where $\vec{a_i}$ are the columns of the matrix A (*eigenvectors of Covariance matrix* C) and Z_i are independent standard normal random variables.

The advantage of this method is that we can obtain a good approximation to the Brownian path by considering a smaller number of standard normals, that is, if we order the vector and matrices such that λ_i are in decreasing order then, $\vec{B_n} \approx A_{k\times k}\Lambda_{k\times k}^{1/2}\vec{Z_k}$, choosing first k many eigenvectors $\vec{a_i}$ of dimension n and corresponding first k many eigenvalues λ_i and only k many Z_i standard normals.

For a 1000-step discrete Brownian path with equal time increments, the magnitudes of the eigenvalues of the covariance matrix drop off rapidly. Most of the variability of the path can be determined using far fewer Z_is.

```r
set.seed(896)
sig = matrix(0,1000,1000)
for(i in 1:1000){
  for(j in 1:1000){
    sig[i,j] = min(i,j)/1000
  }
}

u = eigen(sig)
v = cumsum(u$values)
v = v/v[1000]
z = rnorm(1000,0,1)
zz = z*sqrt(u$values)
ful = u$vectors%*%zz

app20  = u$vectors[,1:20]  %*% zz[1:20]
app100 = u$vectors[,1:100] %*% zz[1:100]

plot(ts(ful, start=0, frequency=1000), ylab="", lwd=2)
lines(ts(app20,  start=0, frequency=1000), col='red',  lwd=2)
lines(ts(app100, start=0, frequency=1000), col='blue', lwd=2)
legend("topright",
       legend = c("Full (1000 components)",
                  "Approximation (100 components)",
                  "Approximation (20 components)"),
       col    = c("black", "blue", "red"),
       lwd    = 2)
```



Smoothness of the Simulated BM Paths by PCA

Though it may look like the simulated paths are smooth but they are actually not. It turns out one can express the j^{th} component of a_i (eigenvector of C) as proportional to sinusoidal function of j and n (if it is a n-step BM path). So, the *PCA* construction in some sense aggregates finitely many such principal components which look sinusoidals evaluated at discrete points. As because the simulated path is generating over a fine-grid and each components of the principal components look sinusoidal over these discrete time, the resulting path appears smooth. However, as one can observe from the plot adding more principal components we do get rougher paths, accounting for further explainability of the variation of the BM path.

One can find further information on this in *Glasserman, Monte Carlo Methods in Financial Engineering (sec:3.1.1).*

5.5 Geometric Brownian Motion (GBM)

Asset prices are commonly modeled using Geometric Brownian Motion (*GBM*), which incorporates both deterministic growth and stochastic fluctuations. The basic reason to choose *GBM* over *BM* is that stock prices can not go negetive and we usually expect prices to compound over time.

For a *GBM* the percentage changes $\frac{S(t_{i+1})-S(t_i)}{S(t_i)}$ are independent for all $t_1 < t_2 < \cdots < t_n$. And a *GBM* process denoted as $S \sim GBM(\mu, \sigma)$ is specified by a *SDE*:

$$\frac{dS(t)}{S(t)} = \mu dt + \sigma dW(t)$$

with drift μ and volatility σ. This equation expresses the *instantaneous returns* in terms of the drift and diffusion terms.

Solving this SDE using $Itô's$ lemma, we obtain

$$S(t) = S(0)e^{[(\mu - \frac{\sigma^2}{2})t + \sigma B(t)]}$$

This can also be written as

$$S(t) = S(0)e^{X(t)},$$

$$X(t) = (\mu - \frac{\sigma^2}{2})t + \sigma B(t)$$

where $B(t)$ is standard Brownian Motion.

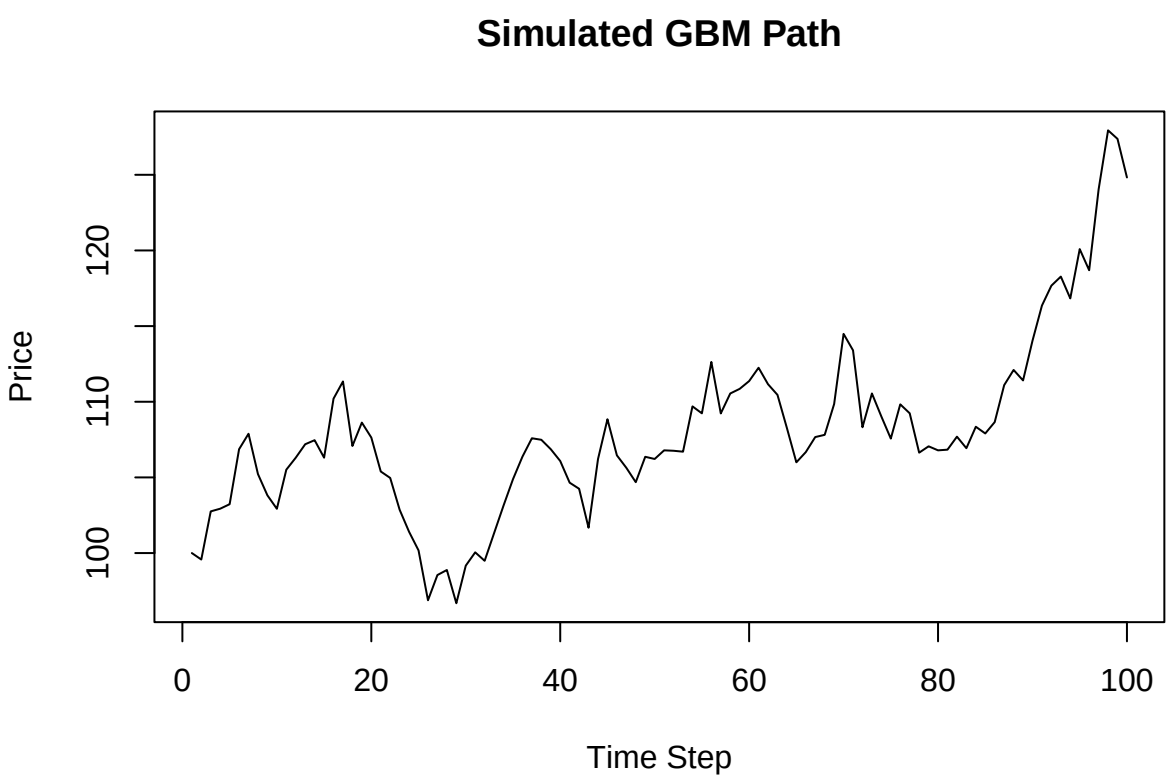
For simulation purposes, the discrete-time approximation is used:

$$S_{t+\Delta t} = S_t \cdot \exp\left[\left(\mu - \frac{1}{2}\sigma^2\right)\Delta t + \sigma\sqrt{\Delta t}Z\right]$$

This formulation ensures that asset prices remain positive and evolve according to log-normal dynamics.

The following code simulates one GBM path.

```r
set.seed(123)

S0 <- 100
mu <- 0.05
sigma <- 0.2
T <- 1
n_steps <- 100
dt <- T / n_steps

S <- numeric(n_steps)
S[1] <- S0

Z <- rnorm(n_steps)

for (i in 2:n_steps) {
  S[i] <- S[i-1] * exp((mu - 0.5 * sigma^2) * dt +
                       sigma * sqrt(dt) * Z[i])
}

plot(S, type = "l", main = "Simulated GBM Path",
     xlab = "Time Step", ylab = "Price")
```



5.6 Brownian Bridge Construction for Barrier Crossing in a Discrete-Time Setting

Barrier options depend on whether the underlying asset price crosses a prescribed barrier during the life of the contract. In practice, stock prices are often observed only at discrete time points $0 = t_0 < t_1 < \cdots < t_n = T$, but the barrier condition is naturally continuous in time. This creates a difficulty: even if the observed prices at two consecutive dates remain below the barrier, the true underlying path may still have crossed the barrier in between. The Brownian bridge construction is useful because it allows us to study this intermediate behavior, conditional on the two observed endpoint values.

Suppose we observe $S(t_i) = s_i, S(t_{i+1}) = s_{i+1}$, and assume both values lie below an upper barrier H. We are interested in the conditional probability that the actual path crosses the barrier somewhere in the interval $[t_i, t_{i+1}]$, namely

$$p_i = \mathbb{P}\left(\max_{u \in [t_i, t_{i+1}]} S(u) \geq H \;\middle|\; S(t_i) = s_i,\ S(t_{i+1}) = s_{i+1} \right).$$

This is the natural quantity for a barrier option.

To derive this conditional crossing probability, we use the geometric Brownian motion model $dS_t = \mu S_t\, dt + \sigma S_t\, dW_t$. It is convenient to work with the log-price $X_t = logS_t$. Over a short time interval, conditional on the endpoint values $X(t_i)$ and $X(t_{i+1})$, the process X_t behaves like a Brownian bridge. Hence, the crossing event between two observation times can be studied by simulating many Brownian bridges with the same endpoints.

Let

$$I^{(m)} = \mathbf{1}\left\{ \max_{u \in [t_i, t_{i+1}]} S^{(m)}(u) \geq H \right\},$$

where $S^{(m)}(\cdot)$ is the m-th simulated bridge between the fixed endpoints s_i and s_{i+1}. If we simulate M such independent bridges, then the sample proportion $\hat{p}_{i,M} = \frac{1}{M}\sum_{m=1}^{M} I^{(m)}$ is an estimator of p_i.

Since the indicators $I^{(1)}, I^{(2)}, \ldots$ are *i.i.d.* Bernoulli random variables with success probability p_i, the Strong Law of Large Numbers gives $\hat{p}_{i,M} \longrightarrow p_i$ almost surely as $M \to \infty$. Thus, the simulated proportion converges to the true conditional crossing probability.

In the context of a barrier option, this means that the path on $[t_i, t_{i+1}]$ can be judged probabilistically rather than only by the two observed endpoint values. If the estimated crossing probability is large, then the barrier event is likely to have occurred in that interval; if it is small, then the path likely stayed on the safe side of the barrier. Then one can construct a decision rule that if such estimated probability is large enough then the option may be exercised or not, depending on some cut-off vale for the estimated probability.

5.7 Simulation of Asset Price Paths

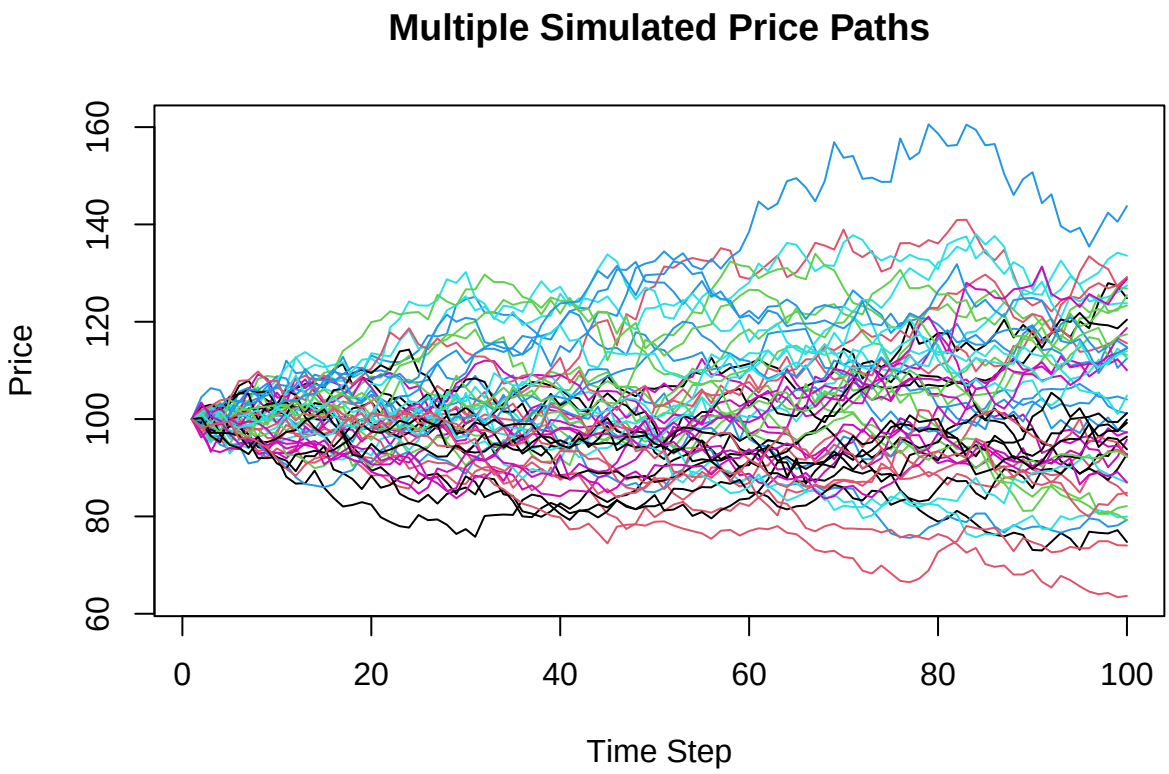
Monte Carlo simulation involves generating multiple realizations of the asset price over time. Each simulated path represents a possible future trajectory of the asset.

The procedure is as follows:

1. Initialize the asset price at S_0
2. Divide the time interval $[0, T]$ into small steps of size Δt
3. At each step, generate independent standard normal random variables
4. Update the asset price using the GBM discretization.
5. Repeat the process to generate a large number of independent paths

Monte Carlo simulation requires generating multiple independent price paths.

```r
set.seed(123)

n_paths <- 50
n_steps <- 100
S_paths <- matrix(0, nrow = n_steps, ncol = n_paths)

for (j in 1:n_paths) {
  Z <- rnorm(n_steps)
  S <- numeric(n_steps)
  S[1] <- 100

  for (i in 2:n_steps) {
    S[i] <- S[i-1] * exp((0.05 - 0.5 * 0.2^2) * (1/n_steps) +
                         0.2 * sqrt(1/n_steps) * Z[i])
  }

  S_paths[, j] <- S
}

matplot(S_paths, type = "l", lty = 1,
        main = "Multiple Simulated Price Paths",
        xlab = "Time Step", ylab = "Price")
```



5.8 Payoff Evaluation

For each simulated price path, the payoff of the option is computed based on its structure.

Examples include:

- European options: payoff depends only on the terminal price S_T
- Asian options: payoff depends on the average price over time
- Barrier options: payoff depends on whether certain price levels are crossed
- Lookback options: payoff depends on the maximum or minimum price observed

This step translates simulated price paths into corresponding financial outcomes.

Once paths are generated, payoffs can be computed. For a European call option:

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

where S_T is the asset price at maturity and K is the strike price.

```
K <- 100

S_T <- S_paths[n_steps, ]
payoffs <- pmax(S_T - K, 0)
```

```r
head(payoffs)
```

```
## [1] 24.82235  0.00000 25.43958  0.00000 27.49067  0.00000
```

5.9 Estimation of Option Price

Once the payoffs are computed for all simulated paths, the option price is estimated as the discounted average payoff:

$$\text{Option Price} = e^{-rT} \cdot \frac{1}{N} \sum_{i=1}^{N} \text{Payoff}_i$$

where r is the risk-free interest rate. The exponential term accounts for the time value of money and converts future payoffs into present value.

```r
r <- 0.05
T <- 1

price_estimate <- exp(-r * T) * mean(payoffs)
price_estimate
```

```
## [1] 9.621958
```

6 Pricing Functions for Exotic Options

6.1 Asian Option Pricing

```r
price_asian_call <- function(S0, K, r, sigma, T, steps, n_paths) {
  S <- simulate_paths(S0, r, sigma, T, steps, n_paths)

  avg_price <- rowMeans(S[, -1])
  payoff <- pmax(avg_price - K, 0)

  return(exp(-r * T) * mean(payoff))
}

price_asian_put <- function(S0, K, r, sigma, T, steps, n_paths) {
  S <- simulate_paths(S0, r, sigma, T, steps, n_paths)

  avg_price <- rowMeans(S[, -1])
  payoff <- pmax(K - avg_price, 0)

  return(exp(-r * T) * mean(payoff))
}
```

6.2 Barrier Option Pricing

```
price_barrier_call <- function(S0, K, r, sigma, T, B,
                               barrier_type = "do",
                               steps, n_paths) {

  S <- simulate_paths(S0, r, sigma, T, steps, n_paths)

  breached <- apply(S, 1, function(path) {
    if (barrier_type == "do") return(any(path <= B))
    if (barrier_type == "di") return(any(path <= B))
    if (barrier_type == "uo") return(any(path >= B))
    if (barrier_type == "ui") return(any(path >= B))
  })

  ST <- S[, ncol(S)]
  payoff <- pmax(ST - K, 0)

  # Apply barrier logic
  if (barrier_type == "do" || barrier_type == "uo") {
    payoff[breached] <- 0
  } else if (barrier_type == "di" || barrier_type == "ui") {
    payoff[!breached] <- 0
  }

  return(exp(-r * T) * mean(payoff))
}
```

6.3 Lookback Option Pricing

```
price_lookback_call <- function(S0, K, r, sigma, T,
                                strike_type = "fixed",
                                steps, n_paths) {

  S <- simulate_paths(S0, r, sigma, T, steps, n_paths)

  if (strike_type == "fixed") {
    max_price <- apply(S, 1, max)
    payoff <- pmax(max_price - K, 0)

  } else if (strike_type == "floating") {
    min_price <- apply(S, 1, min)
    ST <- S[, ncol(S)]
    payoff <- pmax(ST - min_price, 0)
  }

  return(exp(-r * T) * mean(payoff))
}

price_lookback_put <- function(S0, K, r, sigma, T,
                               strike_type = "fixed",
```

```
                                steps, n_paths) {

  S <- simulate_paths(S0, r, sigma, T, steps, n_paths)

  if (strike_type == "fixed") {
    min_price <- apply(S, 1, min)
    payoff <- pmax(K - min_price, 0)

  } else if (strike_type == "floating") {
    max_price <- apply(S, 1, max)
    ST <- S[, ncol(S)]
    payoff <- pmax(max_price - ST, 0)
  }

  return(exp(-r * T) * mean(payoff))
}
```

7 Results & Analysis

7.1 Pricing Outputs

We first define **simulate_paths (Random Walk / Cholesky-based GBM simulation)**, the shared backbone called internally by all Section 5 pricing functions, then price all contracts under a common parameter set.

```
simulate_paths <- function(S0, r, sigma, T, steps, n_paths) {
  dt <- T / steps
  S  <- matrix(0, nrow = n_paths, ncol = steps + 1)
  S[, 1] <- S0
  for (i in 2:(steps + 1)) {
    Z      <- rnorm(n_paths)
    S[, i] <- S[, i-1] * exp((r - 0.5*sigma^2)*dt + sigma*sqrt(dt)*Z)
  }
  return(S)
}
```

```
set.seed(42)

S0      <- 100
K       <- 100
r       <- 0.05
sigma   <- 0.2
T       <- 1
steps   <- 252
n_paths <- 10000
B       <- 85

bs_call <- function(S0, K, r, sigma, T) {
  d1 <- (log(S0/K) + (r + 0.5*sigma^2)*T) / (sigma*sqrt(T))
  d2 <- d1 - sigma*sqrt(T)
  S0*pnorm(d1) - K*exp(-r*T)*pnorm(d2)
```

```
}

bs_put <- function(S0, K, r, sigma, T) {
  d1 <- (log(S0/K) + (r + 0.5*sigma^2)*T) / (sigma*sqrt(T))
  d2 <- d1 - sigma*sqrt(T)
  K*exp(-r*T)*pnorm(-d2) - S0*pnorm(-d1)
}

bs_call_price <- bs_call(S0, K, r, sigma, T)
bs_put_price  <- bs_put(S0, K, r, sigma, T)
```

```
# European: direct simulate_paths (no Section 5 wrapper exists)
S       <- simulate_paths(S0, r, sigma, T, steps, n_paths)
ST      <- S[, steps + 1]
eu_call <- exp(-r*T) * mean(pmax(ST - K, 0))
eu_put  <- exp(-r*T) * mean(pmax(K - ST, 0))

# All exotic options via Section 5 functions
asian_call    <- price_asian_call(S0, K, r, sigma, T, steps, n_paths)
asian_put     <- price_asian_put(S0, K, r, sigma, T, steps, n_paths)

barrier_do    <- price_barrier_call(S0, K, r, sigma, T,
                                    B = B, barrier_type = "do",
                                    steps = steps, n_paths = n_paths)

lb_call_fixed <- price_lookback_call(S0, K, r, sigma, T,
                                     strike_type = "fixed",
                                     steps = steps, n_paths = n_paths)
lb_put_fixed  <- price_lookback_put(S0, K, r, sigma, T,
                                    strike_type = "fixed",
                                    steps = steps, n_paths = n_paths)


results <- data.frame(
  Option   = c("European Call", "European Put",
               "Asian Call", "Asian Put",
               "Barrier (Down-and-Out Call)",
               "Lookback Call (Fixed Strike)",
               "Lookback Put (Fixed Strike)"),
  MC_Price = round(c(eu_call, eu_put, asian_call, asian_put,
                     barrier_do, lb_call_fixed, lb_put_fixed), 4),
  BS_Price = round(c(bs_call_price, bs_put_price,
                     NA, NA, NA, NA, NA), 4)
)
print(results)
```

```
##                         Option MC_Price BS_Price
## 1                European Call  10.4532  10.4506
## 2                 European Put   5.5739   5.5735
## 3                   Asian Call   5.8225       NA
## 4                    Asian Put   3.2846       NA
## 5  Barrier (Down-and-Out Call)  10.1481       NA
## 6 Lookback Call (Fixed Strike)  18.2625       NA
```

```
## 7  Lookback Put (Fixed Strike)  11.8296       NA
```

European call and put prices closely match their Black–Scholes benchmarks, validating the simulation setup before proceeding to convergence analysis.

7.2 Convergence Analysis

7.2.1 Convergence with Respect to Number of Paths N

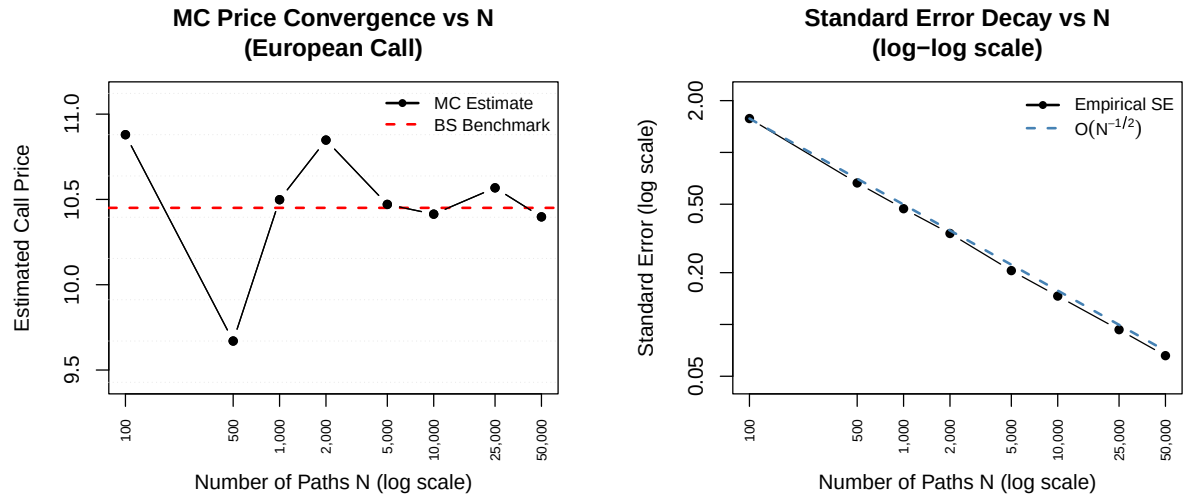
By the Law of Large Numbers the estimator $\hat{V}_N = e^{-rT} N^{-1} \sum_{i=1}^{N} \text{Payoff}^{(i)}$ converges to the true price as $N \to \infty$, with standard error decaying at rate $N^{-1/2}$:

$$\text{SE}(\hat{V}_N) = \frac{\sigma_{\text{payoff}}}{\sqrt{N}}$$

We study this for the European call, using `simulate_paths` directly since no Section 5 wrapper exists for it.

```
set.seed(42)

path_sizes <- c(100, 500, 1000, 2000, 5000, 10000, 25000, 50000)
mc_eu      <- numeric(length(path_sizes))
mc_eu_se   <- numeric(length(path_sizes))

for (idx in seq_along(path_sizes)) {
  N             <- path_sizes[idx]
  S_conv        <- simulate_paths(S0, r, sigma, T, steps, N)
  pv_eu         <- exp(-r*T) * pmax(S_conv[, steps + 1] - K, 0)
  mc_eu[idx]    <- mean(pv_eu)
  mc_eu_se[idx] <- sd(pv_eu) / sqrt(N)
}

price_range <- range(c(mc_eu, bs_call_price))
price_pad   <- diff(price_range) * 0.2
se_ref      <- mc_eu_se[1] * sqrt(path_sizes[1]) / sqrt(path_sizes)
se_range    <- range(c(mc_eu_se, se_ref))

par(mfrow = c(1, 2), mar = c(6, 5, 4, 2))

# Plot 1: Price convergence
plot(path_sizes, mc_eu,
     type = "b", log = "x", pch = 16, col = "black",
     xlab = "Number of Paths N (log scale)",
     ylab = "Estimated Call Price",
     main = "MC Price Convergence vs N\n(European Call)",
     ylim = c(price_range[1] - price_pad, price_range[2] + price_pad),
     xaxt = "n")
axis(1, at = path_sizes,
     labels = format(path_sizes, big.mark = ",", scientific = FALSE),
     las = 2, cex.axis = 0.7)
abline(h = seq(price_range[1] - price_pad,
               price_range[2] + price_pad, length.out = 8),
```

```
       col = "grey90", lty = 3, lwd = 0.8)
abline(h = bs_call_price, col = "red", lwd = 2, lty = 2)
lines(path_sizes, mc_eu, type = "b", pch = 16)
legend("topright",
       legend = c("MC Estimate", "BS Benchmark"),
       col = c("black", "red"), lty = c(1, 2),
       lwd = 2, pch = c(16, NA), bty = "n", cex = 0.85)

# Plot 2: SE decay (log-log)
plot(path_sizes, mc_eu_se,
     type = "b", log = "xy", pch = 16, col = "black",
     xlab = "Number of Paths N (log scale)",
     ylab = "Standard Error (log scale)",
     main = "Standard Error Decay vs N\n(log-log scale)",
     ylim = c(min(se_range) * 0.7, max(se_range) * 1.4),
     xaxt = "n")
axis(1, at = path_sizes,
     labels = format(path_sizes, big.mark = ",", scientific = FALSE),
     las = 2, cex.axis = 0.7)
lines(path_sizes, se_ref, col = "steelblue", lty = 2, lwd = 2)
legend("topright",
       legend = c("Empirical SE", expression(O(N^{-1/2}))),
       col = c("black", "steelblue"), lty = c(1, 2),
       lwd = 2, pch = c(16, NA), bty = "n", cex = 0.85)
```




```
par(mfrow = c(1, 1), mar = c(5, 4, 4, 2))
```

The left panel confirms that the Monte Carlo price converges toward the Black–Scholes benchmark as N grows. The right panel demonstrates the characteristic $O(N^{-1/2})$ decay of the standard error on a log–log scale, where the empirical curve closely tracks the theoretical reference line.

7.2.2 Error vs Benchmark and Confidence Intervals

A 95% confidence interval for the Monte Carlo estimator is constructed as

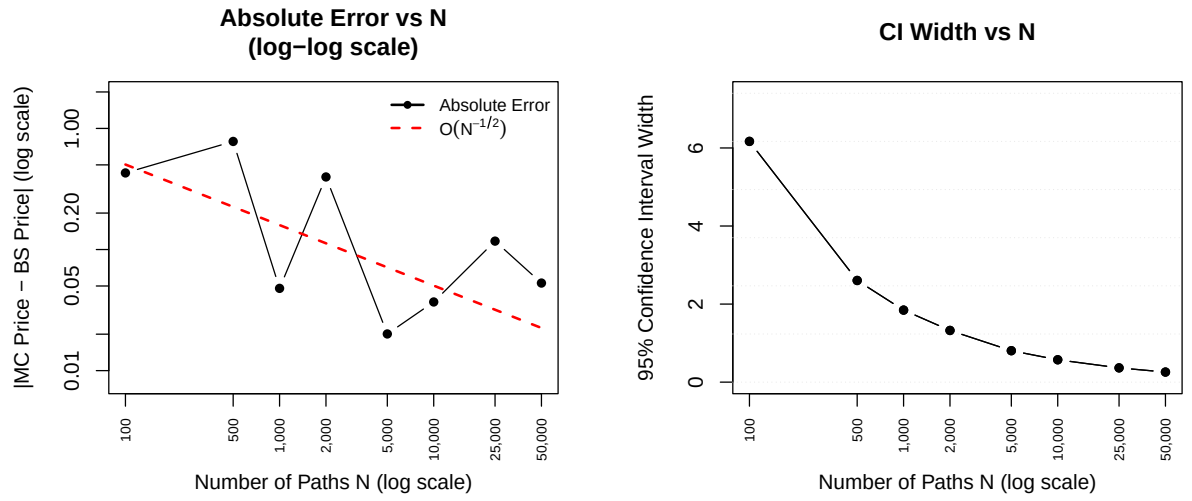
$$\hat{V}_N \pm 1.96 \cdot \frac{\hat{\sigma}}{\sqrt{N}}$$

```
errors <- abs(mc_eu - bs_call_price)

ci_data <- data.frame(
  N       = path_sizes,
  Price   = round(mc_eu, 4),
  SE      = round(mc_eu_se, 5),
  CI_low  = round(mc_eu - 1.96*mc_eu_se, 4),
  CI_high = round(mc_eu + 1.96*mc_eu_se, 4),
  Width   = round(2*1.96*mc_eu_se, 5),
  Abs_Err = round(errors, 5)
)
print(ci_data)
```

```
##       N   Price      SE  CI_low CI_high   Width Abs_Err
## 1   100 10.8795 1.57349  7.7954 13.9635 6.16809 0.42891
## 2   500  9.6699 0.66452  8.3675 10.9724 2.60490 0.78064
## 3  1000 10.4984 0.47085  9.5755 11.4212 1.84573 0.04778
## 4  2000 10.8479 0.33788 10.1857 11.5102 1.32449 0.39732
## 5  5000 10.4707 0.20554 10.0678 10.8735 0.80571 0.02009
## 6 10000 10.4137 0.14623 10.1271 10.7003 0.57321 0.03690
## 7 25000 10.5679 0.09326 10.3851 10.7507 0.36559 0.11732
## 8 50000 10.3977 0.06586 10.2686 10.5268 0.25818 0.05286
```

```
# Anchor reference line on median to avoid near-zero instability
ref_anchor <- median(errors) * sqrt(median(path_sizes))
err_ref    <- ref_anchor / sqrt(path_sizes)
err_ylim   <- c(min(errors[errors > 0]) * 0.4,
                max(errors) * 2.5)
width_ylim <- c(0, max(ci_data$Width) * 1.2)

par(mfrow = c(1, 2), mar = c(6, 5, 4, 2))

# Plot 1: Absolute error log-log
plot(path_sizes, errors,
     type = "b", log = "xy", pch = 16, col = "black",
     xlab = "Number of Paths N (log scale)",
     ylab = "|MC Price - BS Price| (log scale)",
     main = "Absolute Error vs N\n(log-log scale)",
     ylim = err_ylim,
     xaxt = "n")
axis(1, at = path_sizes,
     labels = format(path_sizes, big.mark = ",", scientific = FALSE),
     las = 2, cex.axis = 0.7)
lines(path_sizes, err_ref, col = "red", lty = 2, lwd = 2)
legend("topright",
       legend = c("Absolute Error", expression(O(N^{-1/2}))),
```

```
       col = c("black", "red"), lty = c(1, 2),
       lwd = 2, pch = c(16, NA), bty = "n", cex = 0.85)

# Plot 2: CI width
plot(path_sizes, ci_data$Width,
     type = "b", log = "x", pch = 16, col = "black",
     xlab = "Number of Paths N (log scale)",
     ylab = "95% Confidence Interval Width",
     main = "CI Width vs N",
     ylim = width_ylim,
     xaxt = "n")
axis(1, at = path_sizes,
     labels = format(path_sizes, big.mark = ",", scientific = FALSE),
     las = 2, cex.axis = 0.7)
abline(h = seq(0, max(ci_data$Width) * 1.2, length.out = 7),
       col = "grey90", lty = 3, lwd = 0.8)
lines(path_sizes, ci_data$Width, type = "b", pch = 16)
```




```
par(mfrow = c(1, 1), mar = c(5, 4, 4, 2))
```

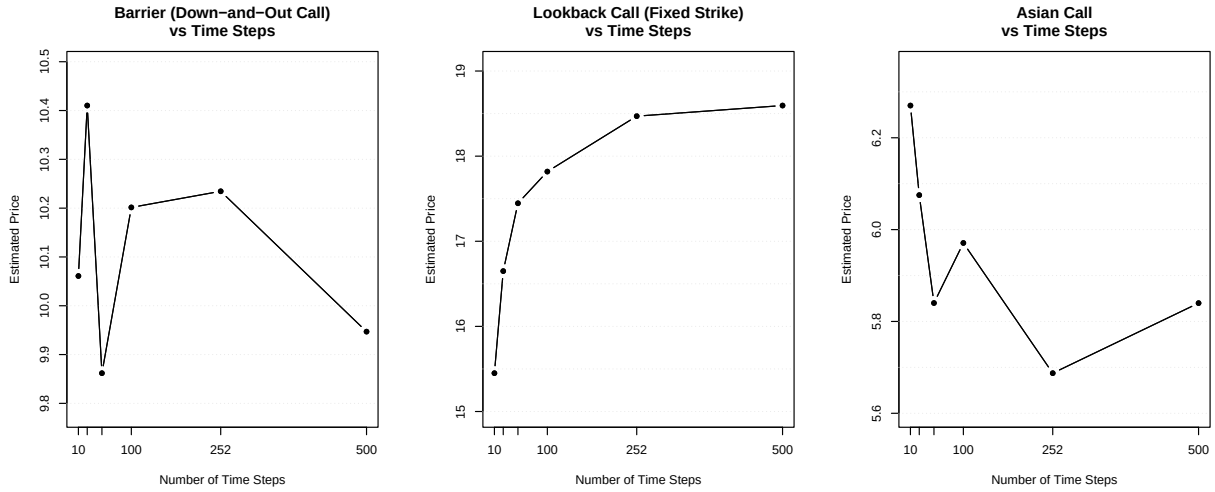
Both the absolute error and CI width follow the $O(N^{-1/2})$ envelope closely. At $N = 50{,}000$ the interval width is negligibly small, confirming that the simulation is practically unbiased relative to the analytical benchmark.

7.2.3 Convergence with Respect to Time Discretization (Steps)

Path-dependent options are sensitive to the number of time steps: coarser grids miss barrier crossings and misrepresent path extremes. We call the Section 5 pricing functions directly, varying only `steps`.

```r
set.seed(42)

step_sizes      <- c(10, 25, 50, 100, 252, 500)
barrier_prices  <- numeric(length(step_sizes))
lookback_prices <- numeric(length(step_sizes))
asian_prices    <- numeric(length(step_sizes))

for (idx in seq_along(step_sizes)) {
  m <- step_sizes[idx]
  barrier_prices[idx]  <- price_barrier_call(S0, K, r, sigma, T,
                                             B = B, barrier_type = "do",
                                             steps = m, n_paths = n_paths)
  lookback_prices[idx] <- price_lookback_call(S0, K, r, sigma, T,
                                              strike_type = "fixed",
                                              steps = m, n_paths = n_paths)
  asian_prices[idx]    <- price_asian_call(S0, K, r, sigma, T,
                                           steps = m, n_paths = n_paths)
}

# Padded ylim helper
ylim_pad <- function(x, frac = 0.15) {
  rng <- range(x)
  pad <- diff(rng) * frac
  if (pad < 1e-6) pad <- abs(rng[1]) * 0.05
  c(rng[1] - pad, rng[2] + pad)
}

par(mfrow = c(1, 3), mar = c(5, 5, 4, 2))

plot(step_sizes, barrier_prices,
     type = "b", pch = 16, col = "black",
     xlab = "Number of Time Steps",
     ylab = "Estimated Price",
     main = "Barrier (Down-and-Out Call)\nvs Time Steps",
     ylim = ylim_pad(barrier_prices),
     xaxt = "n")
axis(1, at = step_sizes)
abline(h = pretty(barrier_prices, n = 6),
       col = "grey90", lty = 3, lwd = 0.8)
lines(step_sizes, barrier_prices, type = "b", pch = 16)

plot(step_sizes, lookback_prices,
     type = "b", pch = 16, col = "black",
     xlab = "Number of Time Steps",
     ylab = "Estimated Price",
     main = "Lookback Call (Fixed Strike)\nvs Time Steps",
     ylim = ylim_pad(lookback_prices),
     xaxt = "n")
axis(1, at = step_sizes)
abline(h = pretty(lookback_prices, n = 6),
       col = "grey90", lty = 3, lwd = 0.8)
lines(step_sizes, lookback_prices, type = "b", pch = 16)
```

```r
plot(step_sizes, asian_prices,
     type = "b", pch = 16, col = "black",
     xlab = "Number of Time Steps",
     ylab = "Estimated Price",
     main = "Asian Call\nvs Time Steps",
     ylim = ylim_pad(asian_prices),
     xaxt = "n")
axis(1, at = step_sizes)
abline(h = pretty(asian_prices, n = 6),
       col = "grey90", lty = 3, lwd = 0.8)
lines(step_sizes, asian_prices, type = "b", pch = 16)
```




```r
par(mfrow = c(1, 1), mar = c(5, 4, 4, 2))
```

All three prices stabilize as the number of steps increases. The barrier option shows the steepest sensitivity at low step counts: a coarse grid fails to detect intra-period barrier breaches, causing the price to be overestimated. The lookback and Asian options exhibit milder but consistent step-size dependence, reflecting the smoother averaging nature of their payoffs.

---

8 Conclusion

8.1 Limitations

The pricing framework developed in this project is based on the Black–Scholes assumptions of constant volatility and interest rates, which may not fully capture real market behavior. Since path-dependent options are simulated using discrete time grids, discretization bias is introduced, particularly for barrier options where crossings between observation times may go undetected. Monte Carlo simulation also converges relatively slowly, with error decreasing at the rate $O(N^{-1/2})$, requiring a large number of simulations for high precision. In addition, the framework mainly considers single-asset models, while higher-dimensional derivatives significantly increase computational complexity.

8.2 Future Directions

Several extensions can improve both realism and computational efficiency. More advanced asset models such as stochastic volatility or jump-diffusion processes may better capture market dynamics. Brownian bridge corrections can reduce discretization bias in barrier option pricing. Variance reduction techniques including antithetic variates, control variates, and quasi-Monte Carlo methods can accelerate convergence. The framework may also be extended to multi-asset derivatives and American options using methods such as Least-Squares Monte Carlo. Finally, parallel and GPU-based computation could make large-scale simulations substantially faster.

---

9 References

Sen, R., Das, S., *Computational Finance with R*, Springer Nature.

Glasserman, P., *Monte Carlo Methods in Financial Engineering*. Springer.

Yuh-Dauh Lyuu, Principles of Financial Computing, NTU.