Numerical Finance - User Guide

Paris-Dauphine University
MSc 203

Faune Blanchard Alban Magerand

April 2025

Contents

Chapter 1

Introduction

This project addresses the pricing of Bermudan basket options within a multi-asset Black-Scholes framework, a challenging problem in numerical finance due to path dependency and early exercise features. The primary objective is to develop a simulation-based approach that accurately estimates the option price while minimizing variance through advanced variance reduction techniques.

Bermudan basket options are financial derivatives that give the holder the right to exercise at several predetermined dates, based on the performance of a group of underlying assets. They are commonly used in structured products and risk management because they offer a balance between flexibility and cost. By combining exposure to multiple assets with limited early exercise rights, these options allow investors to respond to market movements at key times while keeping premiums lower than fully American-style options. Their complexity, however, requires advanced numerical methods for accurate pricing.

The report is structured into four main sections. We begin with a presentation of the numerical tools developed for the code, including matrix inversion via LU decomposition, p-adic arithmetic operations, and Cholesky decomposition to generate correlated Gaussian variables. The second section delves into the theoretical background, covering the properties of correlated random vectors, Monte Carlo and Quasi-Monte Carlo methods, and the role of variance reduction techniques. This is followed by an overview of the program architecture, outlining the modular design and user interface. Finally, we present simulation results with detailed commentary, illustrating the impact of each method on variance reduction and computational efficiency.

Chapter 2

Numerical Tools

2.1 Cholesky decomposition

In order to model the assets in the basket, we need to simulate a multi-dimensional Brownian motion with correlated components. This means generating correlated Gaussian random variables from independent standard normals. This transformation requires computing a matrix B such that $\Sigma = BB^T$, where Σ is the correlation matrix of the Brownian increments. The Cholesky decomposition provides such a matrix B, under the condition that Σ is symmetric and positively defined. Mathematically, if $\Sigma \in \mathbb{R}^{n\times n}$ satisfies these properties, there exists a unique lower triangular matrix L with positive diagonal entries such that

$$\Sigma = LL^T. \tag{2.1}$$

This result is guaranteed by the Cholesky Decomposition theorem in linear algebra, which states that every symmetric positively defined matrix admits a Cholesky decomposition.

To implement this in our simulation framework, the decomposition is computed once during the initialization of the multi-dimensional Brownian motion class. The constructor takes as input a correlation matrix and immediately factorizes it into a lower triangular matrix L, which will be used to transform the vectors of independent standard normal variables into correlated ones. By precomputing L, we avoid redundant calculations during each simulation step, improving both speed and clarity in the numerical workflow.

The decomposition is performed element by element, looping over each row and column. For each entry L_{ij}, the code subtracts the inner product of the previously computed elements:

$$L_{ij} = \begin{cases} \sqrt{\Sigma_{ii} - \sum_{k=0}^{j-1} L_{ik}^2} & \text{if } i = j \\ \frac{1}{L_{jj}} \left(\Sigma_{ij} - \sum_{k=0}^{j-1} L_{ik} L_{jk} \right) & \text{if } i > j \end{cases} \tag{2.2}$$

This recursive process builds the matrix row by row, which ensures numerical correctness. An exception is thrown if any diagonal term L_{ii} becomes non real, which means that the input matrix is not positively defined. This means that we cannot use Cholesky decomposition for this matrix. This exception protects against potentially flawed or badly defined correlation matrices, and therefore preserves the stability and reliability of the simulation.

2.2 P-adic operations

In our project, we use p-adic decomposition to support quasi-Monte Carlo methods, which help to improve the accuracy of our simulations. Quasi-Monte Carlo relies on low-discrepancy sequences designed to fill the space more evenly than standard random number generators. To construct these sequences, we need to work with numbers in base p. By representing numbers that way, we can generate deterministic points that cover the unit interval with much less clustering. This will be presented further later in section 3.2.

This is useful when pricing basket options, because we need to simulate multiple correlated assets across time. In high-dimensional settings, bad or clustered point distribution can cause the results to fluctuate and converge slowly, which might result in a biased price.

To perform arithmetic in a base-p system with high precision, we represent numbers using their p-adic expansion. A p-adic number in the interval $[0, 1)$ can be written as an infinite series of the form:

$$x = \sum_{i=1}^{\infty} a_i p^{-i}, \quad \text{where } a_i \in \{0, 1, \ldots, p-1\}. \tag{2.3}$$

In practice, we truncate this expansion at a fixed precision N, storing only the first N coefficients. The implementation constructs a p-adic number system by taking a base p and a chosen precision, while ensuring that $p > 1$ is valid. The inputs are in $[0, 1)$, they get decomposed into their base-p digits by iterative multiplication and extraction of integer parts, and then stored as vectors of digits.

We then need another tool in order to get a real number from a p-adic number, i.e. the inverse operation. The code interprets the list of base-p digits as a weighted sum using negative powers of p:

$$x \approx \sum_{i=1}^{N} a_i p^{-i}. \tag{2.4}$$

This approximation is useful when visualizing or comparing p-adic numbers in the real number system. Because the precision is finite, the result is a rational approximation of the original real number, reconstructed from its p-adic representation up to the specified depth.

In addition to encoding and decoding, our code also supports the addition of two real numbers via their p-adic expansions. Each number is converted to its digit form, then the digits are added coordinate by coordinate with carry handling, similar to addition in elementary arithmetic. If the sum of two digits exceeds $p - 1$, a carry is propagated to the next higher digit:

$$a_i + b_i + \text{carry} = c_i \mod p, \quad \text{with carry}_{i+1} = \left\lfloor \frac{a_i + b_i + \text{carry}}{p} \right\rfloor. \tag{2.5}$$

The resulting digit list is then transformed back into a real approximation using the reconstruction formula presented above. This method yields precise addition in a base-p system, which is particularly useful in number simulations and certain Monte-Carlo methods, where normal arithmetic may introduce biases.

2.3 Prime numbers

In order to generate low-discrepancy sequences across multiple dimensions, we need a distinct base for each coordinate. A common approach is to use the first d prime numbers, where d is the number of dimensions. This ensures that each base is co-prime to the others, which is important for preserving the uniformity properties of sequences like Kakutani. Our code implements this with "brute force" by looking through each integer, checking if it is prime, and then collecting the first d primes into a list.

The primality test used is simple and effective for small values of d (otherwise, it starts taking a long time to run). For each number, it simply checks whether it is divisible by any integer up to its square root. If no such divisor is found, the number is considered prime and added to the result. This process continues until we have collected enough primes to match the desired dimensionality. These primes are later used as the bases for generating quasi-random numbers in each dimension, forming the foundation of our Kakutani-based sampling method.

2.4 Matrix inversion

In order to invert a square matrix efficiently, we use LU decomposition, which factors a matrix A into the product of a lower triangular matrix L and an upper triangular matrix U, such that:

$$A = LU. \tag{2.6}$$

This decomposition is valid when A is a nonsingular square matrix and often requires row pivoting to ensure numerical stability.

Once we have $A = LU$, we can compute the inverse A^{-1} by solving:

$$AA^{-1} = I \quad \Rightarrow \quad LUA^{-1} = I, \tag{2.7}$$

where I is the identity matrix. We do this by solving $LUx = e_i$ for each column e_i of the identity matrix. This involves two steps: first solving $Ly = e_i$ using forward substitution, then $Ux = y$ using back substitution.

By repeating this process for each column of I, we build up the full inverse matrix one column at a time. This method is more efficient and numerically stable than computing the inverse directly, and is widely used in applications like solving systems of equations or simulating multivariate models.

2.5 Random number generation

2.5.1 Ecuyer Combined

The Ecuyer Combined generator is a pseudo-random number generator that combines two linear congruential generators (LCGs), to produce a sequence of uniform random numbers, with an extended period and improved statistical properties. By carefully choosing the parameters of two independent generators and combining their outputs, the resulting sequence achieves better uniformity and independence.

Each component generator is defined by the recurrence:

$$X_{n+1} = (aX_n + c) \mod m, \tag{2.8}$$

where a is the multiplier, c the increment (here set to zero), and m the modulus. The Ecuyer Combined generator uses two such LCGs:

- The first with multiplier $a_1 = 40014$ and modulus $m_1 = 2{,}147{,}483{,}563$,
- The second with multiplier $a_2 = 40692$ and modulus $m_2 = 2{,}147{,}483{,}399$.

The combined output is calculated by taking the difference of the two generators' states:

$$X_n = (X_n^{(1)} - X_n^{(2)}) \mod (m_1 - 1), \tag{2.9}$$

and the final uniform number on the interval $[0, 1)$ is obtained by:

$$U_n = \begin{cases} \frac{X_n}{m_1} & \text{if } X_n > 0, \\ \frac{X_n}{m_1} + 1 & \text{if } X_n < 0, \\ \frac{m_1 - 1}{m_1} & \text{if } X_n = 0. \end{cases} \tag{2.10}$$

This construction ensures a longer period than either LCG individually and improves the statistical independence between successive values. The resulting sequence of uniform numbers is suitable for high-dimensional Monte Carlo simulations, such as those required for this project (option pricing). The implementation in our code initializes both generators independently and maintains internal state variables. This allows it to deliver reliable and reproducible uniform samples across repeated calls.

2.5.2 Box-Muller Transform

The Box-Muller transform is a widely used method for generating independent standard normal random variables from uniform ones. It is based on the idea that the joint distribution of two independent normal variables can be represented more conveniently in polar coordinates. Specifically, let U_1 and U_2 be two independent random variables uniformly distributed on the interval $(0, 1)$. The Box-Muller transform produces two independent standard normal variables Z_1 and Z_2 as follows:

$$Z_1 = \sqrt{-2 \ln U_1} \cdot \cos(2\pi U_2), \quad Z_2 = \sqrt{-2 \ln U_1} \cdot \sin(2\pi U_2). \tag{2.11}$$

This construction relies on the fact that the transformation from Cartesian to polar coordinates preserves the distribution of the bi-variate normal distribution. The radius component $R = \sqrt{-2 \ln U_1}$ comes from the exponential distribution of the squared radius in polar form, while the angle $\Theta = 2\pi U_2$ is uniformly distributed over $[0, 2\pi)$. The resulting variables Z_1 and Z_2 are both standard normal, and independent by construction.

The Box-Muller transform is not an approximation, and it is especially useful in simulations where samples that closely match the theoretical Gaussian distribution and maintain statistical independence, all the while ensuring accurate and reliable simulation results are required (such as Monte Carlo methods). It is particularly well suited for generating Brownian motion increments in the Euler scheme.

2.5.3 Central Limit Theorem

One way to simulate a standard normal distribution is by leveraging the Central Limit Theorem (CLT). According to the CLT, the sum of a sufficiently large number of

independent and identically distributed (i.i.d.) random variables converges in distribution to a normal distribution, regardless of the original distribution.

More formally, let $U_1, U_2, \ldots, U_n$ be i.i.d. random variables with expected value μ, variance σ^2, and let $S = \sum_{i=1}^{n} U_i$. Then the normalized sum

$$\frac{S - n\mu}{\sigma\sqrt{n}} \xrightarrow{\text{law}} \mathcal{N}(0,1) \tag{2.12}$$

as $n \to \infty$. In practice, convergence is often sufficient for small values of n (typically around 10).

To apply this method, we can choose uniform random variables on the interval $[0, 1]$, where

$$\mathbb{E}[U] = \frac{1}{2}, \quad \text{Var}(U) = \frac{1}{12}. \tag{2.13}$$

If we sum 12 independent uniform variables:

$$\sum_{i=1}^{12} U_i, \tag{2.14}$$

the resulting variable has expectation:

$$\mathbb{E}\left[\sum_{i=1}^{12} U_i\right] = 12 \cdot \frac{1}{2} = 6, \tag{2.15}$$

and variance:

$$\text{Var}\left[\sum_{i=1}^{12} U_i\right] = 12 \cdot \frac{1}{12} = 1. \tag{2.16}$$

Therefore, we can approximate a standard normal variable $X \sim \mathcal{N}(0,1)$ using the expression:

$$X = \left(\sum_{i=1}^{12} U_i\right) - 6. \tag{2.17}$$

This provides a simple way to generate approximated normal samples using only uniform random variables. We will later explain how this is used in the context of this project. However, this significantly worsens compute time since we have to generate a large amount of numbers.

Chapter 3

Theoretical Background

3.1 Correlated Random Variables

In many cases, we have to account for the correlation between assets to obtain a more realistic representation of our universe. In our framework, since we adopt a Geometric Brownian Motion representation to model the evolution of our asset prices:

$$\frac{dX_t^i}{X_t^i} = \mu^i dt + \sigma dB_t^i, \forall i \in [1, d] \tag{3.1}$$

this sums up to defininining a correlation matrix between the Brownian motions (B_t^i). Let $\Sigma \in \mathbb{R}^{d \times d}$ denote the above correlation matrix. We first computes the lower-triangular Cholesky decomposition L of Σ such that:

$$\Sigma = LL^T \tag{3.2}$$

At each time step t_i, a vector $Z^i \in \mathbb{R}^d$ of i.i.d. standard normal variables is sampled and scaled by dt, from which we can obtain a vector accounting for the correlation structure by defining:

$$Z^{\text{corr},i} = LZ^i \tag{3.3}$$

For computational efficiency, it is highly recommended to perform only once the Cholesky decomposition for the given matrix Σ and to re-use across the various simulations.

3.2 Quasi Monte Carlo Methods

Quasi-random number sequences, also known as low-discrepancy sequences, are generated through entirely deterministic algorithms that do not seek to replicate the statistical properties of independent and identically distributed uniform random variables. Instead, these sequences are constructed to fill the unit hypercube in d dimensions more uniformly, minimizing the gaps and clustering that typically arise with pseudo-random numbers. This characteristic leads to a significant advantage in the context of Monte Carlo simulations, where pseudo-random numbers can induce high variance in the estimated results. By contrast, quasi-random sequences offer superior space-filling properties, resulting in more stable and accurate estimates due to their inherent variance reduction capabilities. The core strength of quasi-random sequences lies in their low discrepancy, a measure of how uniformly the points are distributed, which underpins their effectiveness

in high-dimensional numerical integration tasks. There exists various ways to construct such quasi-random sequences (Van der Corput and Halton, Faure, ...) but we decide to focus on a Kakutani sequences since we put on a strong emphasis on this method during the course.

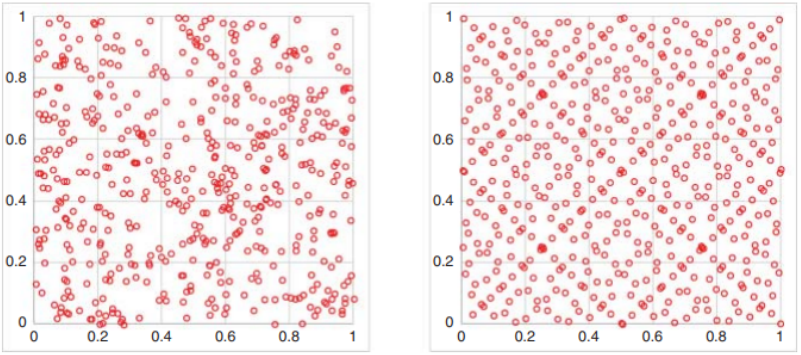

Figure 3.1: A random and a Sobol sequence. Source: [6]

Kakutani sequences offer a simple, flexible way to generate low-discrepancy sequences using p-adic arithmetic, making them easy to adapt dimension-by-dimension with custom shifts. Their main advantage is conceptual simplicity and the ability to extend sequences without recomputing from scratch. However, they tend to be slower due to digit-based arithmetic To build such sequences, we first need to define the p-adic addition, which is a binary operation defined on the set of p-adic expansions. For a given number $x \in [0, 1[$ having a p-adic representation:

$$x = \sum_{k=0}^{r} a_i p^i \text{ where } a_r \neq 0 \text{ and } a_i \in \{0, \dots, p-1\} \tag{3.4}$$

we define the p-adic addition in the common base p as:

$$(x \oplus_p y)_k = (x_k + y_k)\, \mathbf{1}_{\{x_{k-1}+y_{k-1} \leq p-1\}} + (1 + x_k + y_k)\, \mathbf{1}_{\{x_{k-1}+y_{k-1} \geq p\}} \tag{3.5}$$

with the convention $x_1 = y_1 = 0$. Finally, Kakutani sequences are usually defined as:

$$\xi_n = \left(x_i \oplus_{p_i}^{n-1} y_i\right)_{1 \leq i \leq d} \tag{3.6}$$

where we have initially defined $x_i = \frac{1}{p_i}$ and $y_i = \frac{1}{p_i} + \frac{1}{p_i^2}$ which seem to be common choices. By doing so, we effectively define nbSteps sequences of size d of $\mathcal{U}[0, 1]$ numbers, which have all been built deterministically. This yields an issue for Quasi Monte Carlo sequences, as we would obtain the same sequences (and therefore paths) across all simulations. To palliate this, we introduce a shifted version of Kakutani sequences, enabling us to maintain unity of the paths across simulations:

$$\xi_n = \left(x_i \oplus_{p_i}^{n-1+k\times n} y_i\right)_{1 \leq i \leq d}, k \in [1, \text{nbSims}] \tag{3.7}$$

As a first check to see if the generation is better behaved than with `EcuyerCombined`, we can look at the distribution of the samples across `nbSims` $= 10,000$ simulations:

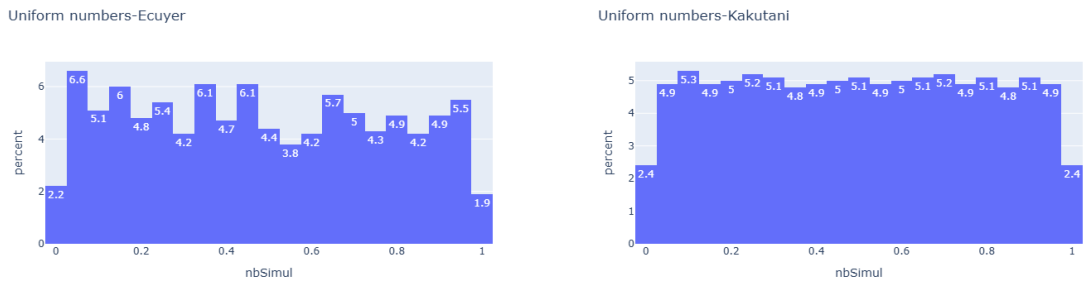

Figure 3.2: Histogram of generated numbers using Ecuyer & Kakutani

As we can observe, the Kakutani sequence is more well-behaved and has a better distribution, with less disparities in frequencies across intervals. In the tails, where both generators fail to have a uniform distribution, Kakutani still performs better with a higher frequency. To check if our implementation worked correctly, we also looked for drifts in the numbers we generated, by looking at the evolution of the mean across batches and did not notice any issues.

Equipped with such sequences, we can now generate uniformly distributed numbers from low-discrepancy sequences, which we pass into a `Normal` generator to get $\mathcal{N}(0,1)$ numbers. Since the main assumption behind the Box-Muller method is the independence of the two uniform variables, we check if the distribution obtained overs `nbSims` $= 10,000$ simulations match the requirements:

$$\begin{cases} \mu = 0 \\ \sigma = 1 \end{cases} \tag{3.8}$$

with the functions `TestMean` and `TestVariance`. It turns out that both equalities are rejected and this can be observed with the following graphs:

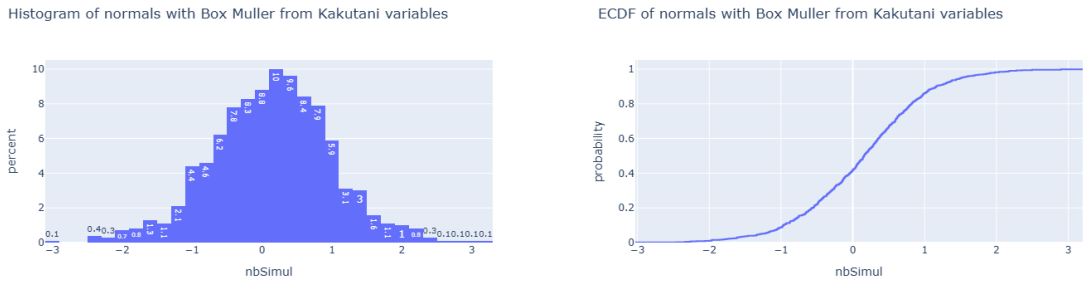

Figure 3.3: Histogram & ECDF of R.V.s generated with Box-Muller from Kakutani

Taking the summary statistics, we observe a mean $\mu = 0.13$ instead of the 0 we were expecting. From these plots, we notice that the distribution is positively skewed, and while we did not run a Kolmogorov-Smirnov test, the distribution does not seem normal at all. This has implications on our pricing, since it directly pushes the price of our forward upwards, increasing the number of paths finishing I.T.M. and therefore the option price.
As a check, we implemented the Central Limit Theorem (see section 2.5.2), which has

a better behavior (we don't reject the null about the mean anymore, but we still do regarding the variance):

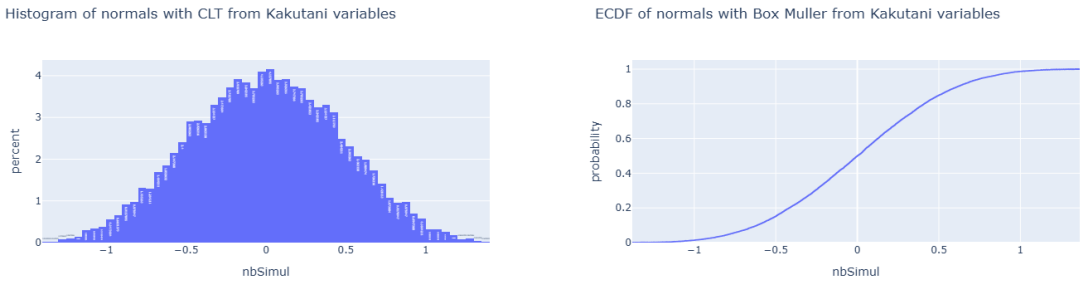

Figure 3.4: Histogram & ECDF of R.V.s generated with CLT from Kakutani

The distribution using this method looks better compared to using the Box-Muller implementation. However, the cost to pay for this method is high -too high for our local machines if we want to run a decent number of simulations- since we need to generate 12 uniform numbers to get one normally distributed variable. Therefore, aware of this shortcoming in our implementation, we revert back to Box-Muller.
To address some of the key issues we faced, namely the skewness of the normal distribution and the poor alignment of the Kakutani sequences with the Box-Muller algorithm, we followed [3] and various others, who implemented a scrambled version of this low-discrepancy sequence, by defining:

$$x'_i = (x_i + \mathcal{U}(0,1)) \mod 1 \tag{3.9}$$

Indeed, according to Niederreiter's theorem (1992), a random shift modulo 1 of a deterministic low-discrepancy sequence maintains low discrepancy in expectation. To create this random shift, we used an instance of `EcuyerCombined`, which enabled us to obtain coherent prices across various variance reduction techniques.

3.3 American Monte Carlo

Let us consider $0 = t_0 < t_1 < ... < t_n = T$ a discrete subdivision of $[0, T]$, where each date t_i represents a potential exercise date. We also define the sequence of stopping times τ_{j,t_i} for a given simulation $j \in [1, \text{nbSims}]$ at time t_i. We apply the Longstaff-Schwartz algorithm (decribed below) to price our Bermudan call basket option.

1. Simulate nbSims stocks paths $(S^j_{t_0}, ..., S^j_{t_n})$
2. Proceed backward by setting $\tau_{j,t_n} = 1$
3. Compute the sequence $\alpha^j_{k,I}$ that minimizes:

$$\frac{1}{M}\sum_{j=1}^{M}\left(e^{-r\left(\tau^*_{j,k+1}-t_k\right)}f(S^j_{\tau_j,k+1}) - \sum_{I\geq 1}^{L} P_I S^j_{t_k}\right) \tag{3.10}$$

where L represents the order of the basis function $P_I(S)$ and the function f represents the payoff of our option.

To actually find this sequence, we notice that we can obtain a system of L equations by successively applying: $\frac{\partial(3.8)}{\partial\alpha_{k,I_0}}, I_0 \in [I = 1, L]$, which can be summed up as:

$$H\alpha_k = \sum_{j=1}^{M} e^{-r\left(\tau^*_{j,k+1} - t_k\right)} f(S^j_{\tau_j,k+1}) P \tag{3.11}$$

where $H = \Phi^T\Phi$ with $\Phi = \begin{pmatrix} P_1(S^0_{t_k}) & \cdots & P_L(S^0_{t_k}) \\ \vdots & \ddots & \vdots \\ P_1(S^{\text{nbSim}-1}_{t_k}) & \cdots & P_L(S^{\text{nbSim}-1}_{t_k}) \end{pmatrix}$ and $P = \begin{pmatrix} P_1(S^j_{t_k}) \\ \vdots \\ P_L(S^j_{t_k}) \end{pmatrix}$

4. Update the stopping time sequence recursively:

$$\tau_{j,h} = t_k \mathbb{1}_{A_{j,k}} + \tau_{j,k+1} \mathbb{1}_{A^c_{j,k}} \text{where} \quad A_{j,k} := \left\{ f(S^j_{t_k}) \geq \sum_{I=1}^{L} \alpha^j_{k,I} P_I(S^j_{t_k}) \right\} \tag{3.12}$$

5. Finally get the price:

$$V_0 = \frac{1}{M} \sum_{j=1}^{M} e^{\tau_{j,0}} f(S^j_{\tau_{j,0}}) \tag{3.13}$$

Regarding the choice of the polynomial basis (P^L_I), we follow the steps of [4] and define the orthonormal basis of the Laguerre polynomials :

$$\begin{aligned} &P^0_I(x) = 1, \\ &P^1_I(x) = 1 - x, \\ &P^n_I(x) = \frac{(2n - 1 - x)\, L_{n-1}(x) - (n-1)\, L_{n-2}(x)}{n} \quad \text{for } n \in [2, L]. \end{aligned} \tag{3.14}$$

where $X = \sum_{i=1}^{d} w_i S^i$. We also tried an alternative basis defined as $P^n_I = X^n, n \in [0, L]$. but the Laguerre basis has the advantage of being orthonormal and offers better numerical stability. Indeed, one of the concerns in this setting is for the matrix H to become ill-conditioned, leading to poor inverse estimates which is essential to estimate the vector α at each timestep. To account for this potential problem, we tried to normalize X using the cross-sectional mean and standard deviation, that is, define:

$$X' = \frac{X - \mu}{\sigma} \tag{3.15}$$

with (μ, σ) being computed from basket values across all simulations for a given timestamp t_i. The results proved to be quite similar, hence why we decided to keep the simplest implementation. Regarding the order L of the regression, we chose $L = 3$ following [2] and [4] in order to avoid overfitting and potential spurious continuation values.

3.4 Static-Control Variate

Monte Carlo methods are powerful for estimating expectations under complex distributions, but they often suffer from slow convergence due to high variance. To improve efficiency, variance reduction techniques are used to reduce the statistical noise of the estimator without introducing bias. One of the simplest and most effective techniques is the *static control variate method*.

Mathematical Background

The core idea is to introduce an auxiliary random variable C, called a *control variate*, which satisfies two key properties:

- C is strongly correlated with the original estimator Y
- The expected value $\mathbb{E}[C]$ is known or easily computed

Given a payoff Y computed via Monte Carlo, we construct a new estimator

$$\hat{Y}_{\text{cv}} = Y + \lambda(\mathbb{E}[C] - C), \tag{3.16}$$

where $\lambda \in \mathbb{R}$ is a scalar parameter, often set to 1 in static control variate implementations. The expectation of $\hat{Y}_{\text{cv}}$ is:

$$\mathbb{E}[\hat{Y}_{\text{cv}}] = \mathbb{E}[Y + \lambda(\mathbb{E}[C] - C)] = \mathbb{E}[Y], \tag{3.17}$$

which confirms that the estimator is unbiased. The variance of $\hat{Y}_{\text{cv}}$ is minimized when:

$$\lambda^* = \frac{\text{Cov}(Y, C)}{\text{Var}(C)}. \tag{3.18}$$

Implementation in Basket Option Pricing

In this project, we use the control variate C as the closed form approximation of the basket call option price, under the Black-Scholes framework. This control variate captures the key dynamics of the payoff. The formula for the lognormal approximation assumes the basket behaves like a single asset with the effective dynamics:

$$X = \prod_{i=1}^{n} S_i^{\alpha_i}, \tag{3.19}$$

$$\sigma_{\text{eff}}^2 = \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j \sigma_i \sigma_j \rho_{ij}, \tag{3.20}$$

$$r_{\text{eff}} = r - \frac{1}{2} \sum_{i=1}^{n} \alpha_i \sigma_i^2 + \frac{1}{2} \sigma_{\text{eff}}^2, \tag{3.21}$$

$$d_1 = \frac{\ln(X/K) + (r_{\text{eff}} + 0.5\sigma_{\text{eff}}^2)T}{\sigma_{\text{eff}}\sqrt{T}}, \quad d_2 = d_1 - \sigma_{\text{eff}}\sqrt{T}, \tag{3.22}$$

$$\text{BasketCall}_{\text{BS}} = e^{-rT} \left(X e^{r_{\text{eff}} T} \Phi(d_1) - K \Phi(d_2) \right). \tag{3.23}$$

In our code, during each Monte Carlo iteration, the path-dependent payoff Y is computed based on the simulated terminal values of the underlying assets. The analytic approximation is simultaneously computed as C, and also the corrected estimator is obtained using the static control variate formula above. Because the analytic basket price is very correlated with the simulated payoff, the variance of the corrected estimator is significantly lower, which leads to to faster and more stable convergence of the pricing algorithm.

3.5 Antithetic Random Variables

Antithetic random variables are a variance reduction technique in Monte Carlo simulations. The central idea is to use the symmetry of the normal distribution by looking at each simulation path together with a mirrored, or "antithetic" version, constructed by taking the negative of the random input. This approach reduces variance by inducing negative correlation between paired samples.

Mathematical Background

Let $X \sim \mathcal{N}(0,1)$ be a standard normal random variable used to drive asset dynamics in a simulation. Instead of using X alone to estimate a payoff $f(X)$, we also compute $f(-X)$, which is an equally likely path due to the symmetry of the normal distribution. The antithetic estimator is then:

$$\hat{Y}_{\text{ant}} = \frac{1}{2}(f(X) + f(-X)), \tag{3.24}$$

and has the same expectation as the original Monte Carlo estimator:

$$\mathbb{E}[\hat{Y}_{\text{ant}}] = \mathbb{E}[f(X)]. \tag{3.25}$$

However, the variance is often reduced, since:

$$\text{Var}(\hat{Y}_{\text{ant}}) = \frac{1}{4}\left(\text{Var}(f(X)) + \text{Var}(f(-X)) + 2\,\text{Cov}(f(X), f(-X))\right), \tag{3.26}$$

and $\text{Cov}(f(X), f(-X))$ are typically negative for monotonic functions such as option payoffs.

Implementation in Path Simulation

In the simulation of asset paths under the multi-dimensional Black-Scholes model, this method is applied by generating both a standard Brownian increment ΔW and its negative counterpart $-\Delta W$ for each time step and each asset. This yields two trajectories: the original path and the antithetic path.

At each time step t, the asset value (S_t) evolves according to:

$$S_{t+\Delta t} = S_t + S_t\left(r\Delta t + \sigma\Delta W\right), \tag{3.27}$$

while the antithetic path uses:

$$\tilde{S}_{t+\Delta t} = \tilde{S}_t + \tilde{S}_t\left(r\Delta t - \sigma\Delta W\right). \tag{3.28}$$

Both paths are simulated in parallel. If the antithetic option is activated, the simulation stores the antithetic paths during the first execution and reuses them on the next call. This ensures that the antithetic estimator is applied consistently across batches of simulations.

The primary advantage of this method lies in its simplicity and efficiency. It requires minimal changes to the simulation code, doubles the amount of information extracted per path, and significantly reduces the variance of the estimator, especially for convex or monotonic payoffs such as those in vanilla and basket options.

Chapter 4

Code user guide

4.1 Philosophy

We started the project with an advanced foundation, providing us with a solid framework to build on and iterate. It was key to us to make our additions fit in as best as possible in the existing architecture, as this would be the expected behavior were we to join a development team in our future positions. We therefore tried to integrate seamlessly our variance reduction methods and new products (European and Bermudan Basket Options). As such, we present our implementation, as well as some tricks we noticed to improve performance.

We start by presenting the architecture of our code, to show the broad design decisions that we made:

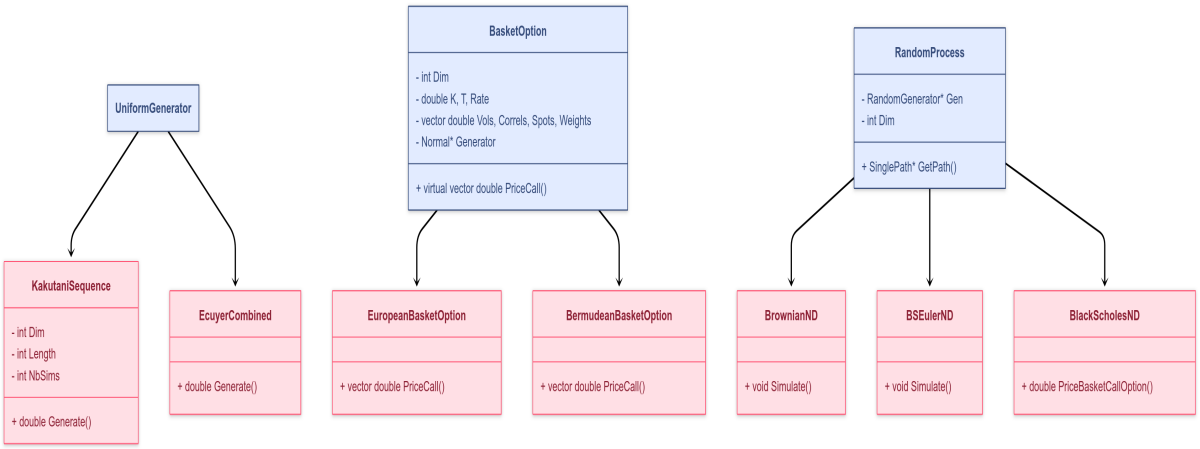

Figure 4.1: Code Architecture

4.2 Practical implementation

The first step in the extension of the code was to be able to model correlated assets (see section 3.1). To do so, we implement the `BrownianND` class, inheriting from `RandomProcess` (`BrownianND` : `RandomProcess`). For the sake of efficiency, we only compute the Cholesky decomposition once at instantiation, and re-use at each `Simulate()` call.

Equipped with this tool, we now implement in a similar fashion as the already existing

`BSEuler1D` : `BlackScholes1D` implementation the `BSEulerND` : `BlackScholesND` classes, which enable us to simulate complete asset paths over the `nbSteps`. This is also the occasion to implement our first variance reduction method, namely the antithetic one, implemented through a `bool antitheticRV` argument in the `Simulate(...)` method of the `BSEulerND` class.
Preliminary to adding the remaining variance reduction methods, we have implemented a `BasketOption` parent class, which enables us to define broadly such option, and built two sub-classes `EuropeanBasketOption` and `BermudanBasketOption`. The parameters needed to instantiate such objects are:

Name	Type	Meaning	Classes
`Dim`	`size_t`	Nb of assets	both
`K`	`double`	Strike	both
`T`	`double`	Time to Maturity	both
`Rate`	`double`	Discount factor	both
`Spots`	`vector<double>`	Initial prices	both
`Vols`	`vector<double>`	Volatilities	both
`Weights`	`vector<double>`	Assets weights in the basket	both
`Correls`	`vector<vector<double>>`	Correlation matrix	both
`Gen`	`Normal`	Generator of normally $\mathcal{N}(0,1)$	both
`L`	`size_t`	Order of the basis expansion	`Bermudan`

Table 4.1: Parameters for our pricers

Their common method `PriceCall(...)` provides us with with an intuitive and easy-to-modify interface assembling all the tools we have implemented.
We can now turn our focus on the remaining techniques to implement and begin with the control variate one (see section 3.5). For ease of use in the various experiments that we will lead in Chapter 5, we use a `bool UseControlVariate`. Through this variable, we are able to directly influence the payoff vector at the pricing level, making the various combinations of variance reduction methods easier.
Finally, we implement quasi-random number generations in the `KakutaniSequence` : `UniformGenerator` child class. By using this design, we make the switch from random to quasi-random numbers in the pricing fairly easy: we just need to change the generator `Gen` when we instantiate our options. Essentially, quasi-random methods yield sequences of $\mathcal{U}(0,1)$ distributed numbers with specific properties about their discrepancy. Given the use we make of such sequences `UniformGenerator` : (generating a number for each asset at each timestep), it made sense to fit our `KakutaniSequence` class in this framework, by keeping track internally of the simulation number, the timestep and the asset. When instantiating a `KakutaniSequence` object, a whole $\mathbb{R}^{\text{nbSims}\times\text{nbSteps}\times d}$ tensor is created, enabling us to run the whole number-generation process beforehand. We tried various approaches (e.g. regenerating a sequence after each simulation only), but the gain in computing time was significant with the currently implemented method.
It is also worth mentioning that we added several independent functions and classes (`Utils` folder), designed in order to be used throughout the whole code. They represent basic methods or objects that we had to manipulate and can be summed up as:

1. Functions to compute the mean and the variance of a `vector<double>`

2. Matrix representation of a `vector<vector<double>>`, enabling us to compute the transpose, inverse, etc ...

3. Function to write to a .csv file a `vector<vector<double>>` in order to perform data visualization on Python

Chapter 5

Results

5.1 European Basket Call Option

In this section, we present the results for the prices of a European basket call option with the following parameters :

Parameter	**Value**
Number of assets	3
Maturity T	1.0 year
Strike price K	60
Number of time steps	365
Spot prices $S_i(0)$	$\{100, 50, 60\}$
Volatilities σ_i	$\{0.10, 0.25, 0.16\}$
Risk-free rate r	0.05
Basket weights α_i	$\{0.10, 0.70, 0.20\}$
Correlation matrix Σ	$\begin{bmatrix} 1.0 & 0.1 & 0.1 \\ 0.1 & 1.0 & 0.1 \\ 0.1 & 0.1 & 1.0 \end{bmatrix}$

Table 5.1: Simulation parameters for basket option pricing

We will test three methods to obtain the price of such product, using Monte Carlo simulations. Note that we have been able to test the raw implementation (i.e. without any reduction techniques) by using the `TestSDE()` function provided, by setting a weight of 1 for a specific asset in our basket. We will then compare the results from each method:

- Basic pseudo-random sampling (no variance reduction)
- Antithetic random variables
- Static control variate
- All possible combinations

For each method ,we generated plots showing how the estimated option price evolves as the number of simulations increases. Each plot includes:

- The running mean of the estimated price

- A shaded confidence interval (explained further later)
- A dashed horizontal line representing the last price with the maximum number of simulations

We started with 1000 simulations and went up to 25 000 by steps of 500, for a total of 50 data points.

As per the course, to make the price enter a confidence interval such as :

$$[m_X - \epsilon,\, m_X + \epsilon] \tag{5.1}$$

with a confidence level of α, we needs to process a Monte Carlo simulation of size :

$$N \geq N^X(\epsilon, \alpha) = \frac{a_\alpha^2 \operatorname{Var}(X)}{\epsilon^2} \tag{5.2}$$

We assume we want a 95% confidence interval (meaning $a_\alpha^2 \approx 1.96$, with $\epsilon = 2.5\%$). We write the raw results first and then present the graphs. `Nb Sim` represents the number of simulations needed to enter the confidence interval based on the last given price with 25 000 simulations:

Technique	Price	Std	Nb Sim
Basic	3.731	39.2376	241 176
Static Control Variate	3.681	0.4792	2 945
Antithetic RV	3.649	37.9327	233 155
Antithetic + SCV	3.672	0.4624	2 842
Kakutani	3.604	37.3924	229 834
Kakutani + SCV	3.668	0.4604	2 830
Kakutani + Antithetic	3.714	39.2417	241 201
Kakutani + Antithetic + SCV	3.669	0.4540	2 790

Table 5.2: Variance reduction results

No reduction methods

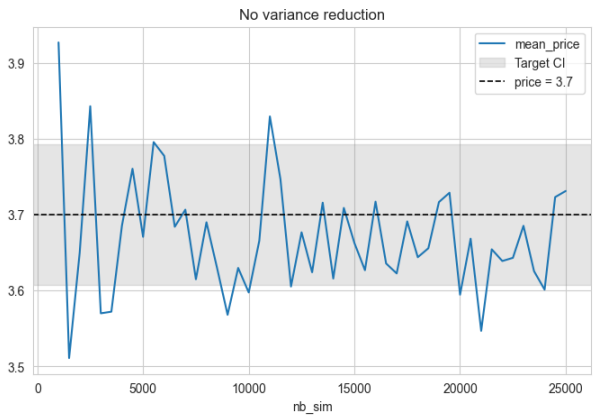

Figure 5.1: Raw simulation without any variance reduction.

The plot above shows the evolution of the Monte Carlo estimator for the basket call option price without applying any variance reduction techniques. As the number of simulations increases, the estimate begins to fluctuate around the maximum simulation (25 000) value of 3.7, indicated by the dashed horizontal line. However, the convergence is visibly noisy and irregular, particularly when the number of simulations is lowest. Despite a general trend toward stabilization, the estimator exhibits significant ups and downs. This behavior shows that standard MC when used without any variance control can be inefficient: we also see that we would need more than 20,000 simulations to enter our 95% confidence interval with $\epsilon = 2.5\%$. This means the approach is more computationally expensive and potentially unreliable for more complex pricing tasks.

However, this baseline provides a useful reference for evaluating the improvements achieved through the application of variance reduction techniques in the rest of our experiments.

Antithetic Random Variables

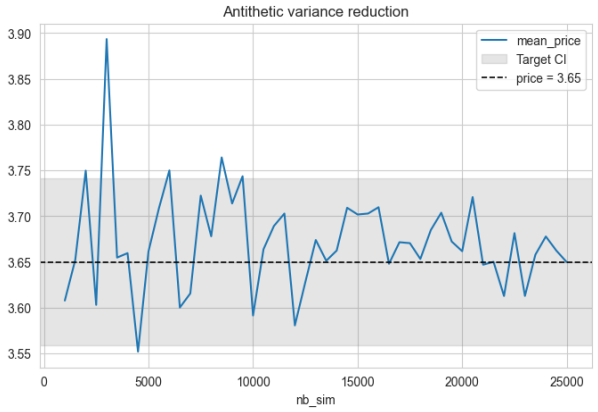

Figure 5.2: Antithetic variance reduction.

This plot shows the results of using antithetic random variables to reduce the variance. The price estimate still fluctuates significantly, especially with low simulation: there is not much improvement compared to no reduction although it is still slightly better. The method's effect on variance seems to be limited when used alone.

Static Control Variate

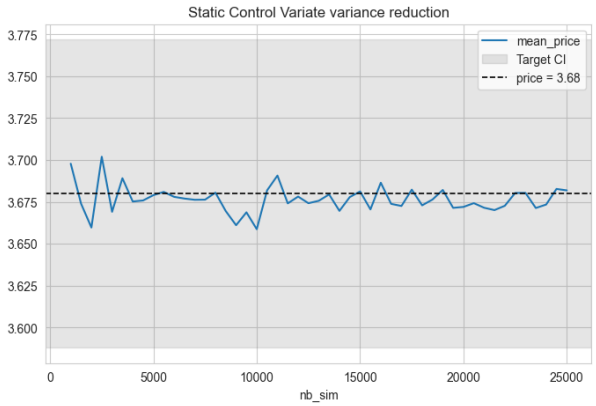

Figure 5.3: Static Control Variate variance reduction.

This plot shows the effect of using a static control variate for variance reduction. The estimator is much more stable and is close to the max simulation price throughout the simulation. The convergence is visibly faster, as confirmed by the number of simulations needed to enter the interval (showed in table 5.1). This confirms that using a control variate based on an analytic approximation (such as the lognormal formula for the basket option) is highly effective when the correlation between the payoff and the control is strong.
Such a result is striking, and can be understood by the very nature of the operation performed with this technique. Across all paths, independently of the moneyness, the payoffs vector consists in the theoretical price, setting a baseline. Then, depending on the moneyness, we add the difference of the Monte-Carlo payoff of the actual product and of the control variate, which is supposedly small since the two are highly correlated. This method therefore introduces a level (the theoretical price), around which we observe small fluctuations, which necessarily lowers the variance.
We remind the reader that the scale on the y-axis has changed for better visibility compared to the previous graphs. This is why the confidence interval appears larger. This means that the reduction in variance would appear even more drastic to the eye with the same scale, largely outperforming the previous two graphs.

SCV + Antithetic Variance Reduction

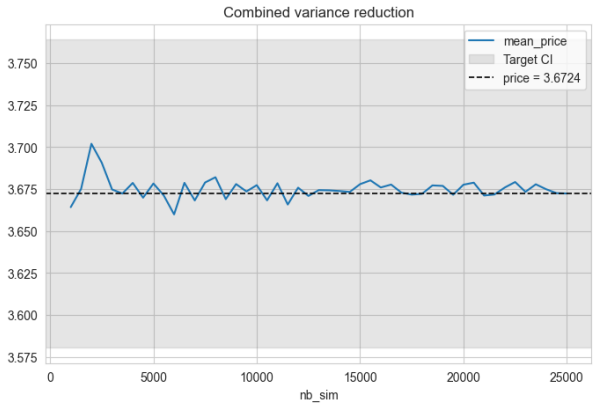

Figure 5.4: SCV and Antithetic variance reduction.

This last plot combines both techniques together, antithetic variables and static control variate. This yields the most stable and most accurate result. The estimated price converges quickly and remains within the confidence interval throughout all simulation steps. This shows that combining variance reduction methods can significantly amplify their individual benefits, leading to a more efficient and reliable estimator. However, we can see that compared to static control variate alone there is not necessarily much improvement.

Kakutani Reduction

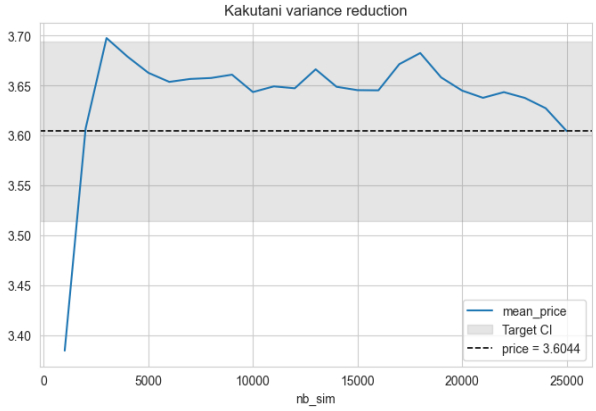

Figure 5.5: Kakutani variance reduction.

We can see that although the entire curve is more distant to the final value, the curve itself looks smoother, except for outliers when there is a low number of simulations.

Kakutani + Antithetic Reduction

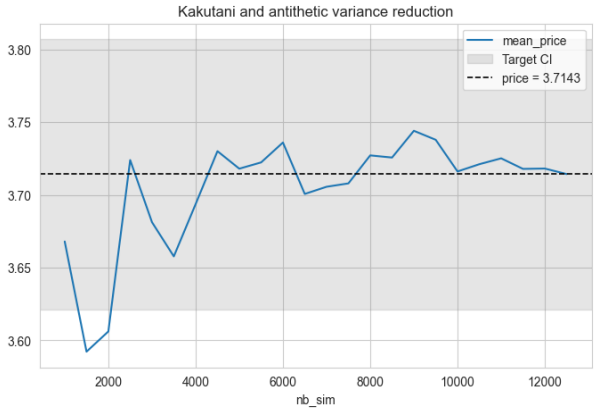

Figure 5.6: Kakutani + Antithetic variance reduction.

Here, we observe that the curve is relatively smooth and close to the final price, but with almost a positive trend in the curve. The results are slightly worse than with Kakutani alone in term of standard deviation, which is a topic mentioned in [3]. Indeed, it is often not advised to combine antithetic methods with low-discrepancy sequences, since the symmetry interferes with the uniformity of the original low-discrepancy sequence. Instead of improving coverage, it duplicates structure in a way that breaks the constructed distribution, and increases correlation between points.

Kakutani + SCV Reduction

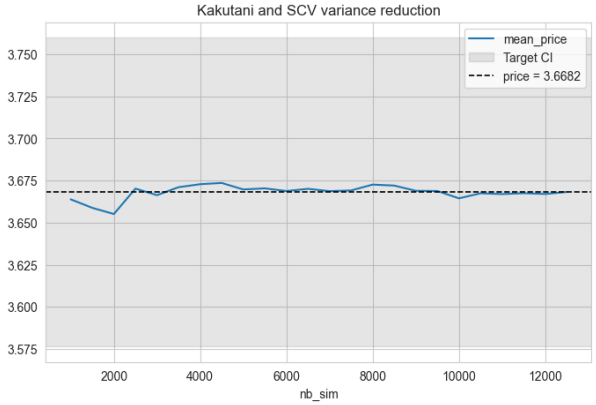

Figure 5.7: Kakutani + SCV variance reduction.

This looks to be the most drastic improvement yet, with a smooth curve that closely follow the final price and never leaves the confidence interval.

Kakutani + SCV + Antithetic Reduction

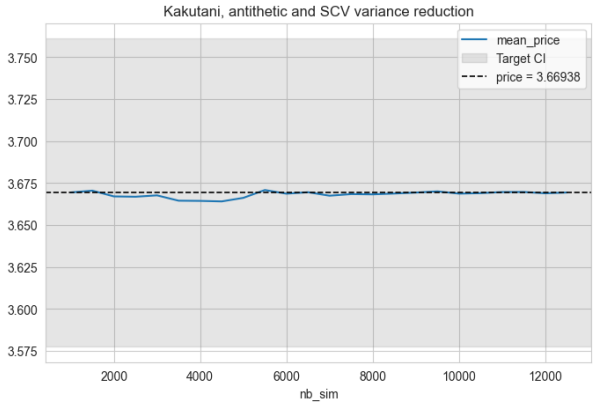

Figure 5.8: Kakutani + SCV + Antithetic variance reduction.

Once again these results are very positive as the curve looks almost like a straight line. The effect of the antithetic seems marginal, but it diminishes the fluctuation around the final value, which is a desired characteristic.

Conclusion

In general, we find that variance reduction significantly improves the performance of Monte Carlo methods. While antithetic sampling helps modestly, the static control variate brings a major improvement. Combining both methods leads to the best results and is highly recommended, especially in high-dimensional problems or when computational efficiency is critical. We can also see that Kakutani smooths the overall curve (with less oscillation around the final value), but with values that go further away from the confidence interval when there is a little amount of simulations. Finally, the best method is the one that combines all three variance reduction methods, where the curve is almost a straight line around the final price of approximately 3.67.

5.2 Bermudan Basket Call Option

In this section, we present the results for the pricing of Bermudan basket call options. We use the same parameters as for the European basket call, and we test the same variance reduction methods. The Bermudan basket option extends on the European basket option by allowing the holder to exercise at several predetermined dates before maturity.

We first present the raw results. Note that since the Kakutani method takes a very long time to compute, we only have half of the maximum number of simulations (12500 instead of 25 000), with steps of 500 in both cases.

We immediately notice that the variance reduction is nowhere near as drastic as with the European call, when we apply variance reduction techniques. With the European call, we went from $\approx$ 30 to 40 to $\approx$ 0.4, which was huge. Now the variance is only diminished by 3 when combining antithetic and static control variate, the most important reduction. We also therefore notice that we need a lot more simulations in order to enter the confidence interval.

Technique	Price	Std	Nb Sim
Basic	3.2529	18.776	115 407
Static Control Variate	3.14578	16.3642	100 583
Antithetic RV	3.22927	19.307	118 671
Antithetic + SCV	3.19829	15.5293	95 451
Kakutani	3.21184	18.4952	113 681
Kakutani + SCV	3.16187	15.1198	92 934
Kakutani + Antithetic	3.22044	18.7504	115 250
Kakutani + Antithetic + SCV	3.11827	16.3056	100 223

Table 5.3: Variance reduction results

No variance reduction

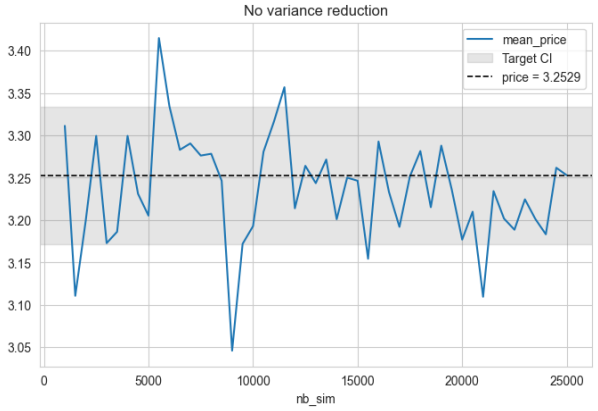

Figure 5.9: Combined variance reduction.

As per the European call, we notice a lot of variance in the results during this step. But, most noticeably, the final estimated price is slightly lower than the corresponding European basket call option with the same parameters. This result is theoretically incorrect. Bermudan options, by design add flexibility to the holder and should therefore make the option more expensive. In fact, the Bermudan price must lie exactly between the European and American prices:

$$\text{European Price} \leq \text{Bermudan Price} \leq \text{American Price} \tag{5.3}$$

This means that the lower value probably indicates a problem in implementation of the early exercise feature, or others. Therefore, while the simulation process looks stable, the result seems to indicate a possible logical error in the Bermudan pricing code, maybe particularly in how optimal stopping or early exercise is handled (since this is the main difference with the European basket call). Another theory relies on the filtering of the O.T.M. paths, which we decided not to implement. Indeed in his seminal paper, [4] only keeps I.T.M. paths to avoid biasing downwards the continuation values, therefore avoiding regressing noise and focusing on relevant exercise scenarios. This implies several design choices, especially when there are none or few I.T.M. paths, making the estimation of the α coefficients unstable. In our implementation, we noticed that continuation values

could sometime be negative, leading to early exercise of options which were not even in the money, which should not be possible in theory.

Static Control Variate Variance reduction

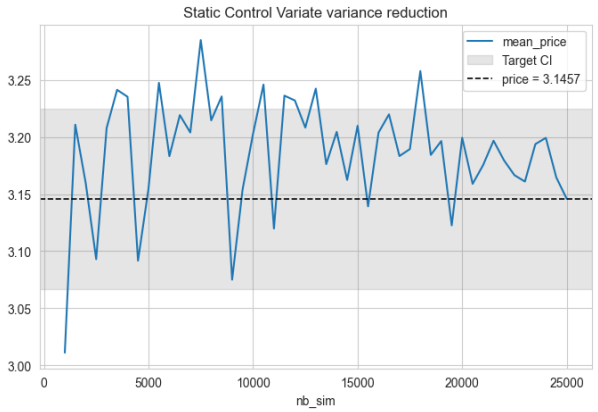

Figure 5.10: SCV variance reduction.

Here, the results are much less encouraging than for the European call option. We have an oscillating curve with a lot of values outside of the condidence interval, even for a high number of simulations.

Antithetic Variance reduction

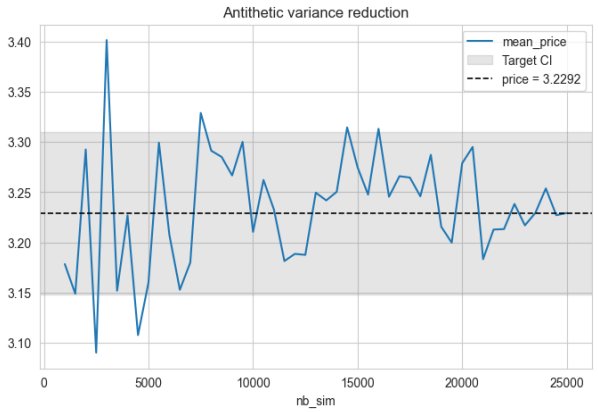

Figure 5.11: Antithetic variance reduction.

This plot shows noisy results that are difficult to interpret. Both previous result seem to yield no significant improvement compared to the pricing without any variance reduction method.

Antithetic + Static Control Variate variance reduction

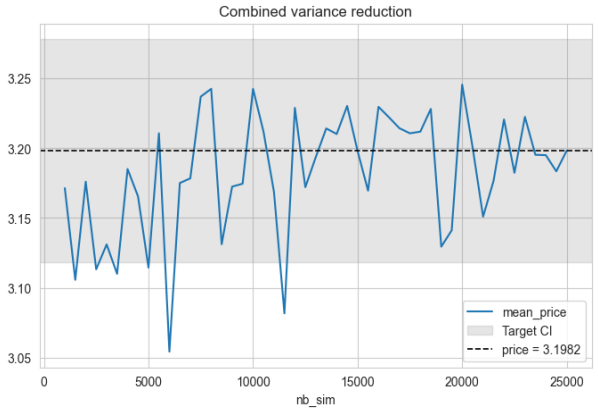

Figure 5.12: Combined variance reduction.

The variance is visibly higher compared to the equivalent reduction for the European basket call. We again notice that the price is lower than the price of the European basket call (when it should be higher).
This high variance could be attributed to the intrinsic complexity of Bermudan options : their early exercise feature introduces path dependency and decision-based discontinuities in the payoff structure. This increases sensitivity to simulation noise and reduces the effectiveness of variance reduction techniques.

Kakutani Variance reduction

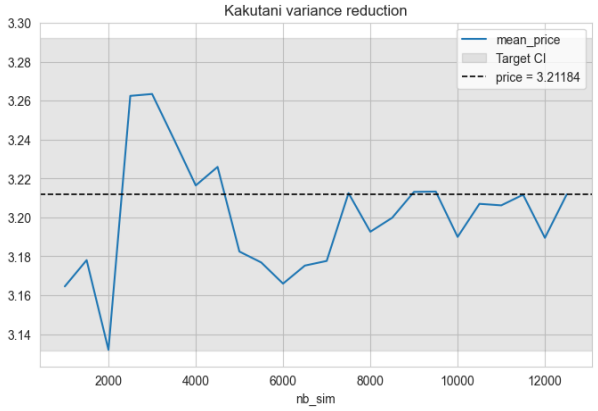

Figure 5.13: Kakutani variance reduction.

Immediately, we can see that the results look much smoother and that all values stay within the confidence interval. However, the curve presents bigger "bumps" rather that oscillations (as per previous results), which means that we might be further off if we select an unlucky number of simulations to use.

Kakutani + SCV Variance reduction

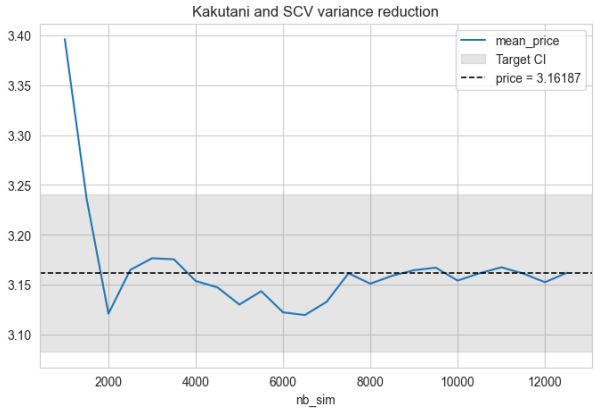

Figure 5.14: Kakutani + SCV variance reduction.

Here, we observe a smooth curve with limited bumps, and only a very large value when there are not enough simulations. However, after about 8000 simulations, we seem to have a very consistent price. This corroborates the results presented in the table earlier where this combination yielded the lowest variance.

Kakutani + Antithetic Variance reduction

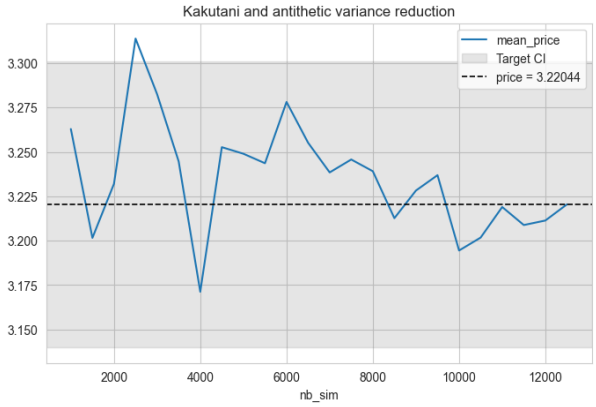

Figure 5.15: Kakutani + Antithetic variance reduction.

The results here are less satisfying than previously, with bigger and sharper jumps in the curve, which goes out of the confidence interval.

Kakutani + Antithetic + SCV Variance reduction

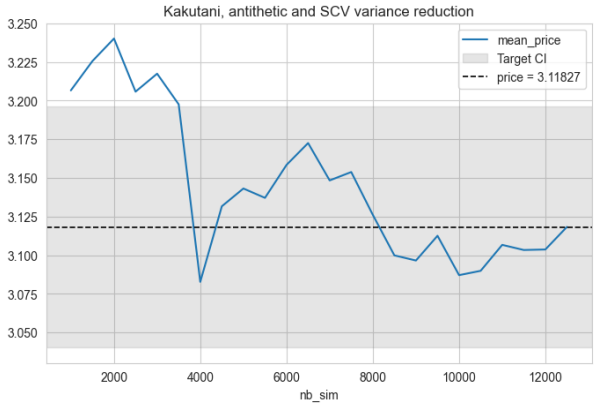

Figure 5.16: Kakutani + Antithetic + SCV variance reduction.

Similarly to what we observed with the European call, the combination of the three methods yields a satisfying result, although in this case it is clearly outperformed by the Kakutani and SCV combination. It seems that adding antithetic variance reduction actually worsens the results, as we explained earlier. This might be because the technique assumes a certain level of symmetry and smoothness in the payoff function that may not hold in this context.

Antithetic sampling works best when the payoff is a monotonic or smooth function of the underlying random variables, such as in European options with terminal payoffs. By pairing each sample with its "mirror" reducing variance. However, in a Bermudan option, the value depends on optimal early exercise decisions taken at multiple points in time. These decisions are not smooth or symmetric and may change drastically depending on small changes in the path.

When antithetic paths are used in this setting, the mirrored path might lead to different exercise decisions which potentially introduce more noise instead of canceling it out. The result is that the variance across paired paths may actually increase. This can make antithetic sampling not only ineffective but counterproductive, as we can see here.

Conclusion

Here, we can see that the combined method is much less effective for Bermudan options than it is for European options. It looks as if the best combination is Kakutani and Static Control Variate, which leads to the best variance and the flattest curve. Once again we can see that Kakutani smooths the curve overall even when it doesn't seem like the gain in variance is that drastic (for example, Kakutani alone vs no variance reduction have rougly the same variance yet the curve looks much smoother but with more drastic values for Kakutani). With a high number of simulations, we can infer that Kakutani and SCV leads to the best results.

We remind the reader that Kakutani sequences are a type of quasi-random sequence designed to cover the simulation space more uniformly than standard pseudo-random numbers. In the context of Monte Carlo simulation, this means that the paths generated will explore the range of possible outcomes more evenly, reducing gaps and clustering. On top of this, the static control variate complements this by reducing variance because

the simulation adjusts each payoff to account for deviations from the expected control, which stabilizes the results. This is especially helpful in Bermudan options, where early exercise decisions add noise and make the payoff harder to approximate directly.

Together, these two techniques address different challenges in pricing. Kakutani sequences improve the quality of the input by providing well-distributed samples, while the control variate improves the quality of the output by correcting for simulation error. When used together, they create a more stable and efficient estimator, which is particularly valuable for complex options like Bermudans where both path-dependence and early decision-making increase sensitivity to noise.

Chapter 6

Conclusion

Throughout this project, we explored the pricing of options on multi-asset underlyings and looked at how different techniques can help improve the accuracy of Monte Carlo simulations. We applied control variates and antithetic sampling to reduce variance in our estimates. We also implemented quasi-random sequences and found that they significantly improved the stability and reliability of our results.options and studied the positive influence of Monte-Carlo variance enhancing features. Within a C++ framework, we applied notions of control variates and antithetic variables to further reduce the variance of the results of our simulations.

The comparison between the European and Bermudan basket call options shows that variance reduction techniques do not perform equally well in all situations. For the European option, which has a simple payoff based only on the final asset prices, the static control variate works best, and it does equally well when combined with other methods. These techniques take advantage of the smooth and predictable structure of the problem, which helps reduce noise in the simulation.

On the other hand, the Bermudan option involves early exercise decisions, which introduce path dependency and make the problem more complex. Because of this, the same variance reduction methods are less effective. The exercise feature creates jumps and non-smooth behavior in the payoff, which weakens the assumptions that these techniques rely on.

Overall, this highlights that the choice of variance reduction method should consider the structure of the option being priced. Simple methods work well for terminal payoffs like those in European options, but more advanced or tailored approaches may be needed when dealing with early exercise features or path-dependent problems. Improvements could be achieved by comparing other normal random variables generation schemes, trying other quasi-random sequences and including more products in our Pricer base.

Bibliography

[1] Russel E. Caflisch. Monte carlo and quasi-monte carlo methods. In *Acta Numerica*, volume 7, pages 1–49. Cambridge University Press, 1998.

[2] Paul Glasserman. *Monte Carlo Methods in Financial Engineering*, volume 53 of *Applications of Mathematics*. Springer, 2004.

[3] Christiane Lemieux. *Monte Carlo and Quasi-Monte Carlo Sampling*. Springer, 2009.

[4] Francis A. Longstaff and Eduardo S. Schwartz. Valuing american options by simulation: A simple least-squares approach. *The Review of Financial Studies*, 14(1):113–147, April 2001.

[5] William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. *Numerical Recipes: The Art of Scientific Computing*. Cambridge University Press, New York, 3rd edition, 2007.

[6] Antoine Savine. *Modern Computational Finance*. Wiley, 2018.

[7] Linlin Xu and Giray Ökten. High-performance financial simulation using randomized quasi-monte carlo methods. *Quantitative Finance*, 15(8):1425–1436, August 2015.

[8] Giray Ökten. Generalized von neumann–kakutani transformation and random-start scrambled halton sequences. *Journal of Complexity*, 25(4):318–331, August 2009.