Paris Dauphine University

Numerical Finance Project

Pricing of Bermudan Basket Options

Romain CASTELLARNAU, Maxime GARRIGA, Lukas SAULIS

Dauphine | PSL
UNIVERSITÉ PARIS

Abstract

Bermudan options are financial derivatives that resemble to American options, but have a limitation: they only allow the holder to exercise the option on certain pre-determined dates. This feature makes the pricing tasks more complex and requires additional steps compared to European options.

In this project, our objective is to price such options, on baskets of assets using Monte Carlo methods, assuming the assets follow a multi-dimensional diffusion model similar to Black-Scholes. To achieve this, we developed both pseudo and quasi-random number generators to simulate asset dynamics through both Euler and Milstein schemes. Additionally, we implemented various variance reduction techniques, including the utilization of quasi-random numbers to reduce simulation randomness, control variates to leverage known variable values for variance reduction, and antithetic variates to mitigate estimation errors through negatively correlated variable pairs. These methods significantly enhanced the computational efficiency of our Monte Carlo simulations.

The implementation of this project was done using the language Python, the source code is available to the reader at this address: (Source Code). This report presents our findings and the significant results achieved through the different variance reduction methods. Additionally, it includes a user guide explaining the code architecture and its functionalities.

Contents

1 Introduction

History: Bermudan options are a type of financial derivative that was created in the late 20th century. They were created in order to offer a product with more flexibility than European options (which can only be exercised at maturity) and less flexibility than American options (which can be exercised at any time before maturity).
Bermudan options allow the holder to exercise the option on specific dates, providing a balance between European and American options. This feature makes Bermudan options attractive for investors who want more control over when they can exercise the option but also don't want to pay the expensive premium of American options.

Objective: The main goal of this paper is to compare different variance reduction techniques to see how they affect the pricing of Bermudan basket options.
We want to understand which methods can result in more accurate and more stable pricing. By using these techniques, we hope to find the best ways to reduce the uncertainty (variance) in the simulation results, which is very important for accurate pricing of these options.

Plan: After the introduction, we will provide a user guide which will help understand the basic concepts discussed in this report, as well as some guidelines for not getting lost in our code. The following section explores various pricing and variance reduction methods for European options, including the use of Monte Carlo simulations with quasi-random numbers and other techniques like control and antithetic variates.
Then, we apply these techniques to Bermudan options specifically, using the Longstaff-Schwartz Monte Carlo algorithm, and comparing the outcomes in different scenarios.
Finally, the conclusion summarizes the findings as well as the limits of our report, and suggests areas for improvement.

Results: Our results will show that while the simple estimator converges without bias towards the estimated value, the precision and the speed of convergence can be improved up to a certain level without increasing the computation costs. We will see that some techniques show great results but require much more computational power. There will be a balance between the following notions: speed of convergence, precision, number of simulations and numerical costs.

Parameters used: For the purpose of this document, we will put the theory to the test by considering a real basket including three American stocks. The basket we will consider is constituted of the following stocks: Apple (AAPL), Amazon (AMZN), & Tesla (TSLA). We chose our basket to be equally weighted for the nominal of each assets, meaning that we want to be exposed similarly on each asset. By doing so, the spot and scale of each asset has no importance and this allows us to benefit from each asset's dynamics.

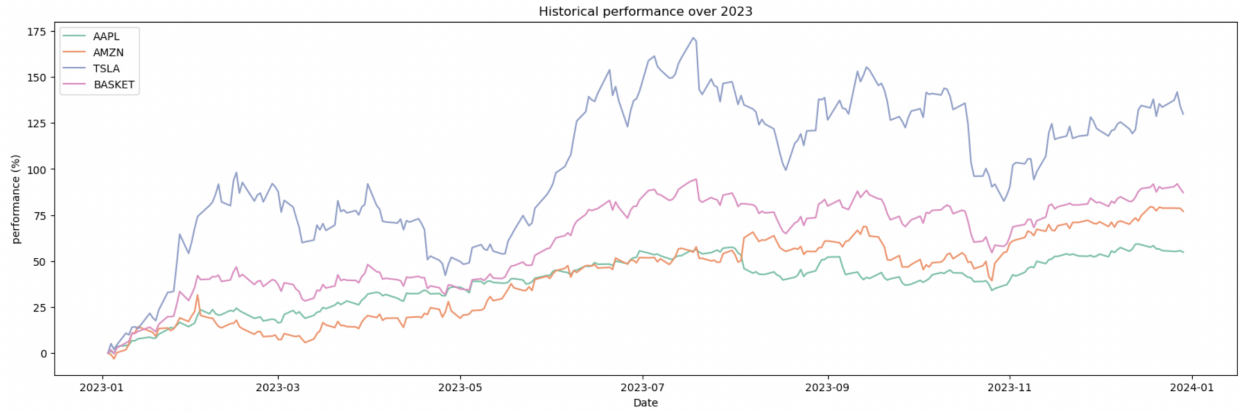


Figure 1: Stocks performance during 2023 - base: 2023-01-01

The three stocks are not trading on the same scales at the time of the writing of this report[1], AAPL is worth 169$, AMZN 179$ and TSLA 180$. Later in this document, we will have to simulate the basket dynamic which require to know the volatilities and correlations of each asset. To price and simulate, we will use the annual realized volatility[2] of each asset based on their values from the year 2023.

Below are the observed realized volatility and correlations computed from the historical data of the three assets.

Table 1: 2023 Realized annual volatility

AAPL	AMZN	TSLA
19.95%	33.02%	52.65%

Here is the correlation matrix we will use throughout the document based on the observed realized values from 2023.

Table 2: 2023 Realized correlation Matrix

	AAPL	AMZN	TSLA
AAPL	100%	88.47%	82.75%
AMZN	88.47%	100%	79.33%
TSLA	82.75%	79.33%	100%

Finally, in order to be exposed at 30% on each asset's performance, we need to adjust this weight by each asset's spot price at the time of the writing. This leads to the following weights table. We multiplied the weights by two to achieve a basket initial size of 200$.

Table 3: Adjusted weights table[3]

AAPL	AMZN	TSLA
39.38%	37.24%	37.04%

[1] user remark: time of the writing and today's values are fixed as 2024-05-01

[2] the implied volatility inducing a forward vision for the volatility and not backward is normally used in such works

[3] Assets spots: APPL. $169, AMZN. $179, TSLA $180.

2 User Guide

Before diving into the methods and the results, you will find a guide to our work presenting the structure behind our ideas.

A quick word for the users:

For this work, we decided to give the user the ability to do almost everything as specified rather than simply using switches and booleans that would be set in a function.
For example, the algorithm applying MCLS to an European payoff works well even if it has no objective of providing a better result. For example, we also allow the user to provide any function A, h_0 (as we will define later) that are being used as Monte Carlo variance reduction techniques. There is no need to worry as there also exists a way to get existing ways without much effort.

Arborescence:
Here is the architecture of our work. It is separated into three main folders **Generators**, **Results** and **SDE**.

The graphical results of our work are contained in three notebooks in the result class.

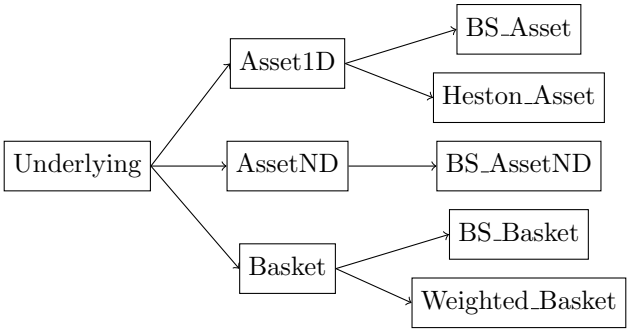
The functions used to simulate the random variables i.e. normal distribution for our assets (and many more other distributions!) are contained within the Generators folder.
Finally, the different objects used when pricing are in the SDE section of our work.

```
my_work/
├── README.md
├── Generators/
│   ├── Continuous_Generators/
│   │   ├── ContinuousGenerator.py
│   │   ├── Exponential.py
│   │   └── Normal.py
│   ├── Discrete_Generators/
│   │   ├── Bernouilli.py
│   │   ├── Binomial.py
│   │   ├── DiscreteGenerator.py
│   │   ├── FiniteSet.py
│   │   ├── HeadTail.py
│   │   └── Poisson.py
│   ├── Uniform_Generators/
│   │   ├── EcuyerCombine.py
│   │   ├── LinearCongruential.py
│   │   ├── PseudoGenerator.py
│   │   ├── QuasiGenerator.py
│   │   ├── UniformGenerator.py
│   │   └── VanDerCorput.py
│   └── RandomGenerator.py
├── Results/
│   ├── main_notebook.ipynb
│   ├── mcls_variance_reduction_results.ipynb
│   ├── variance_reduction_notebook_results.ipynb
│   └── random_generator_results.ipynb
└── SDE/
    ├── basket.py
    ├── bs.py
    └── heston.py
```

```
├── montecarlo.py
├── pricer.py
├── underlying.py
└── utils/
    ├── matrix.py
    ├── montecarlo_utils.py
    ├── random_generation.py
    └── yfinance_functions.py
```

The key part of this project, which is related to pricing, is contained in the SDE folder, with the pricing related function contained within the two files *"montecarlo.py"* and *"pricer.py"*. The remaining file of the folder are used to build the object used by the pricer.

Figure 2: Architecture of class Underlying



Above is the structure set between our objects, which has to be analyzed with the attributes found in the table below.

Table 4: Use of Class Underlying

Class attributes			
Underlying	Asset1D	AssetND	Basket
initial_spot	vol	assets	weights
dimension	rf	correl_matrix	assetND
generator_function	nb_brownians_per_sim		nb_brownians_per_sim
			aggregate_function

Walking through the different attributes, most of the attributes speak for themselves with their names. However, here are some key elements that might be useful and you can play with to act with agility and make the code do exactly what you'd want.

- **generator_function**: When simulating the path or a single value of an asset, we need to first simulate (whether it is inside or outside the function and seen or not by the user) some Gaussian values which will be used later on by the function computing the dynamic of an asset. Using a value *is_live* in a constructor, you can set the seed or not of this generator function. There also exists for each underlying object a class function called *generate_random_values_in_required_format* which has per role to return a function which allows to simulate the given number of simulations in a format that is expected by a *simulate_path_with_brownians*-type function. The **generator_function** is called by *generate_random_values_in_required_format*.

```
# set a function simulating for 10 periods
simulator_function = my_undl_object.generate_random_values_in_required_format(nb_periods=10)
simulator_function(1000)  # simulates 1000 sets of N(0,1) under format expected for such object
```

The trick by initiating the object with live_seed= True is to fix the results and be able to show the gain in variance across different estimators for a comparable set of simulations (see graph similar to**??**).

How to define any custom basket:

- **.aggregate_function**: a Basket is composed of an assetND object which contains the list of assets that composed our basket. the assetND type is useful to us to simulate the dynamic between the d-assets and then gather their value from d dimensions to 1 to represent the basket value. This is the role of the "aggregate_function" with signature Callable[[Iterable],float].
  In the case of a **Weighted_basket** for example, the aggregate_function is set to numpy.sum which will apply the weighted average when considering the basket weights. You are free to build your own basket by doing such command :

```
# example of creating a Best-of basket
my_BO_basket = BS_Basket(weights,[asset_1,asset_2],correl_matrix,aggregate_function=np.max)

# alternatively, class for weighted basket already exist
my_basket = Weighted_Basket(weights,[asset_1,asset_2],correl_matrix)
```

Note that the class "Weighted_Basket" already exist so you don't have to define again the aggregate function.

- **nb_brownians_per_sim**: corresponds to the number of Brownian required for one simulation of an asset. In our case is 1 in BS model. There is a challenging aspect to our code to use it with = 2 when simulating Heston model
- **scheme**: (in BS_Asset) when initiating an asset of type *BS_Asset*, by setting the scheme in the constructor to "euler" or "milstein", it will simulate the underlying using one technique or the other.

Pricing object: To obtain a price, we set the rule that we should have the following elements:

- an option of type (Option)
- an underlying of type (Underlying)
- a pricing configuration of type (PricingConfig)

The two classes Option and PricingConfig are located in the file **"pricer.py"**. They own the following attributes :

Table 5: Use of Class Option & PricingConfiguration

Class attributes	
Option	PricingConfiguration
_strike: float	_nb_simulations: int = 1000
_terminal_payoff_function: Callable	_pricing_model_name: str = "CF"
_actualised_payoff_function: Callable	_A_transformation: Callable = None
_actualisation_rate: float	_h0_function: Callable = None
_maturity: float	_m_value: float = None
_exercise_times: List[float]	_is_antithetic: bool = False
_payoff_name: str = None	_is_control_variate: bool = False
_exercise_type: str = "european"	_is_quasi_random: bool = False
pricing_configuration: PricingConfiguration	

The role of the PricingConfiguration class is to provide information that will be used later by the pricing function. Those information as it is explicit with the attributes is to attribute the model, the number of simulations or the function which will be used during the process of variance reduction methods.

- **.set_predefined_pricing_reduction_techniques**: The class Option has a method that takes an underlying as input. Depending on the type of the underlying and the exercise periods, it fills the parameters "_m_value", "_h0_function", and "_A_transformation" to predefined functions. For example, function A: $x \mapsto -x$ is set for every underlying, but the function $h0$ introduced later in this document will only be set to the object pricing_configuration of the option if the object is of type Weighted_Basket.

How to achieve different models of pricing:

There is an example of how to use such methods in our notebook located in "ResultsLaTeX main_notebook.py". There are three ways to price: using a closed form formula (denoted CF) (only available for Black and Scholes single underlying options), using Monte Carlo (MC) and using Monte Carlo Longstaff Schwartz (MCLS). Finally, you can achieve a price using the following lines of code:

```
# assume my_asset defined previsouly

# Creating ATM option
call_option_object = Option(
            strike = 100,
            actualisation_rate= 0.05,
            maturity= 1,
            payoff_name= "call"     # allows not to give payoff but not mandatory
            )

### EXEMPLE OF PRICING USING CLOSED FORM FORMULA ###
# pricing_model_name = "CF" is set by default when not providing pricing config

print(f"CF European price is: {call_option_object.Price(my_asset,display_info=True)} \n")

######### EXEMPLE OF PRICING USING MC ##############
# 1/ build pricing config
pricing_config = PricingConfiguration(nb_simulations=1000000,pricing_model_name="MC")
# 2/ set pricing config to option object
call_option_object.set_pricing_configuration(pricing_config)
# 3/ price with according underlying
print(f"MC European price is: {call_option_object.Price(my_asset,display_info=True)} \n")

######## EXEMPLE OF PRICING USING MCLS ##############

pricing_config = PricingConfiguration(nb_simulations=1000000,pricing_model_name="MCLS")
call_option_object.set_pricing_configuration(pricing_config)
print(f"MCLS European price is: {call_option_object.Price(asset_AAPL,display_info=True)} \n")
```

Note that when pricing according to the method of your choice, you have to follow the steps :
1/ create the pricing object,
2/ set the pricing config to the object,
3/ price with the corresponding underlying.

How to activate variance reduction methods:

The last bit of effort asked for the user is to provide or not a variance reduction method. This has to be done by setting a parameter and a function to the PricingConfiguration object type.
Our code is very flexible and you can easily provide any function A or h0 (as defined later in the document). In order to avoid the user to provide a method, there exists a function called "**.set_predefined_pricing_reduction_techniques**" This loads some functions for a given underlying type and exercise way.

```
### EXEMPLE OF PRICING USING PRE DEFINED REDUCTION TECHNIQUE FOR MCLS ###
pricing_config = PricingConfiguration(nb_simulations=10000,pricing_model_name="MCLS")
call_option_object.set_pricing_configuration(pricing_config)

# **load existing variance techniques**
call_option_object.set_predefined_pricing_reduction_techniques(my_basket_object)
# activate antithetic pricing
call_option_object.set_pricing_antithetic()
# activate control variate pricing
call_option_object.set_pricing_control_variate()
# price
print(f"MCLS Bermudean price is: {call_option_object.Price(my_basket_object,display_info=True)} \n")
```

How to create custom-made basket and price while using variance reduction:

As mentioned, using the aggregate_function element from a Basket object, you can create any basket you like. This makes us wonder how to price a best-of option on a basket. The transformation $A : x \mapsto -x$ is known to be a solution for variance reduction in such framework. Here is how you should do to perform in such situation.

```
# setting function max as aggregate to custom a BO basket
my_BO_basket = BS_Basket(weigths,[asset_AAPL,asset_AMZN],correl_matrix,
live_seed=True,
aggregate_function=np.max)

# 1/ define option
call_option_object = Option(strike,actualisation_rate,maturity= 1,
            exercise_type= "european",
            payoff_name= "call")
# 2/ define pricing config
pricing_config = PricingConfiguration(pricing_model_name="MC",nb_simulations= 100000)
# 3/ set pricing config to option
call_option_object.set_pricing_configuration(pricing_config)
# 4/ give mannually variance reducition techniques
call_option_object.set_pricing_reduction_techniques(A=A = lambda x: -x)
# 5/ price
print(f"MC Best of option w. Antithetique is: {call_option_object.Price(my_BO_basket,display_info=True)}
```

Congratulations ! You successfully completed the training "pricing & classes basics".

3 Pricing & Variance Reduction methods for European basket options

In this section, we will explore how to price European basket options and how different variance reduction techniques can change this pricing. We will use Monte Carlo simulations and combine it with variance reduction methods like antithetic variates and control variates. Each of these methods has its own benefits in making our estimates more accurate. By comparing them with each other, we will show which combination of these methods can be used to reduce the variance in our pricing process.

3.1 Introduction to European Basket Call Options

European basket call options are a type of financial derivative that gives the buyer the right, but not the obligation, to buy a basket of assets at a predetermined price on a specific expiration date. Unlike American options, which can be exercised at any time before they expire, European options can only be exercised on the expiration date, making their pricing simpler but still requiring calculations in order to predict how multiple underlying assets will behave until the expiry.

The basket in a European basket call option typically consists of stocks or indices, and the payoff depends on the combined performance of all assets in the basket.
The payoff of a European basket call option can be described by the formula:

$$C(T) = e^{-rT} * max(S(T) - K, 0) \tag{1}$$

where:
- $C(T)$ is the value of the call at expiration,
- T is the time to expiration,
- r is the risk-free interest rate,
- $S(T)$ represents the weighted sum of asset prices in the basket at time T,
- K is the strike price of the option,
- e^{-rT} is the discount factor that adjusts the future payoff to its present value.

The main challenge in pricing these options comes from the need to calculate how the different underlying assets in the basket correlate with each other over time which affects the volatility of the option. These correlations can have an important influence in pricing, as they determine if the assets in the basket move together or not under different market conditions.
To do our pricing we assumed the assets prices to follow a Black-Scholes type diffusion which can be written as:

$$dS_t^i = \mu(S_t^i)dt + \sigma(S_t^i)dB_t^i \quad S_0^i \in \mathbb{R}$$

Where (B_t) is a Brownian motion, such that $B_0 = 0$ and for all $0 \leq s \leq t$, we have that:

$$(B_t - B_s) \sim \mathcal{N}(0, t-s)$$

Hence, we use the latter property to simulate recursively the random process B_t through:

$$\begin{aligned} B_{t+dt} &= B_t + Z_{t+dt}\sqrt{dt}, \quad Z_{t+dt} \sim \mathcal{N}(0,1) \\ B_0 &= 0 \end{aligned}$$

To obtain a trajectory for the asset's price, we relied on two different schemes. The Euler Scheme which allows us to obtain recursively an asset trajectory through:

$$S_{t+dt}^i = S_t^i + \mu(S_t^i)dt + \sigma(S_t^i)(B_{t+dt} - B_t)$$

And the Milstein scheme, which enhances the stochastic term:

$$S_{t+dt}^i = S_t^i + (\mu - \frac{1}{2}\sigma^2)(S_t^i)dt + \sigma(S_t^i)(B_{t+dt} - B_t) + \frac{1}{2}\sigma^2(S_t^i)(B_{t+dt} - B_t)^2$$

To simulate this multi-dimensional Black-Scholes diffusion, i.e., the diffusion of the whole basket, we need to correlate the Brownian motions $\left(B_t^1, B_t^2, ..., B_t^d\right)$. This correlation between the different Brownian motions $d < B^j, B^k >_t = \rho_{j,k} dt$. Hence, to simulate the dynamics of the basket as a whole, we need to possess their correlation matrix, which we denote Σ. If this matrix is invertible, we apply the Cholesky decomposition, which allows us to express Σ as the product of a lower triangle matrix L and its conjugate transpose.

$$\Sigma = L \cdot L^*$$

Then, by generating $(Z_i), i = 0, 1, ..., d$ i.i.d $\sim \mathcal{N}(0, 1)$ and multiplying these d realizations of independent standard normal random variables by the matrix L, we obtain a d-dimensional Gaussian vector X with correlation matrix Σ. We are now able to simulate the d-dimensional Brownian motions vector required in the simulation of our basket by simply having a standard normal random variable generator.

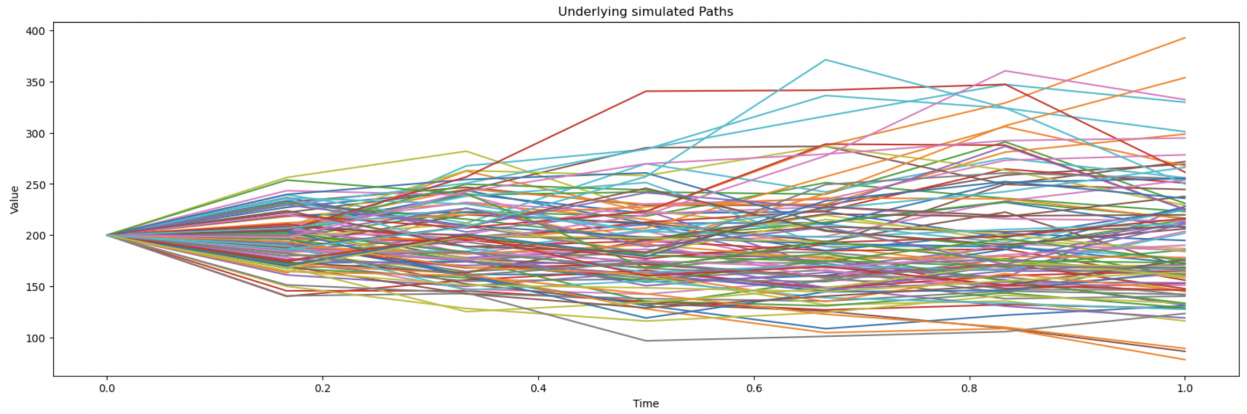


Figure 3: Basket 100 paths simulations with 5 dates[4]

3.2 Pricing using Monte Carlo Simulations and Pseudo-Random Numbers

Monte Carlo simulations are very important in finance, especially when it comes to pricing derivatives like European basket call options. These options depend on the performance of multiple underlying assets, which makes their valuation complex because of how these assets correlate with each other. The Monte Carlo method helps calculate their value by simulating possible future scenarios for underlying prices and calculating the option's payoff in each scenario. It uses the idea of the law of large numbers: by simulating many paths and averaging the results, it gives a good estimate of the option's present value.

In theory, the basic steps for using Monte Carlo simulations in basket options are the following:

1. Simulating the future price paths for each underlying in the basket using an appropriate stochastic model (often a geometric Brownian motion for stock prices).
2. Calculating the basket's value at expiration for each path, which is a weighted average from pre-defined weights of each underlying in the basket.
3. Computing the payoff for each path, which is the maximum of zero and the basket's value minus the strike price.
4. Discounting each of these payoffs back to the present value using the risk-free rate.
5. Averaging these discounted payoffs across all simulated paths to get the final basket option's price.

[4]For basket built and described in introduction.

A crucial part of Monte Carlo simulations is the use of pseudo-random numbers. These numbers are important for simulating all the different paths the underlyings might follow. In Monte Carlo simulations, these numbers usually need to follow a Gaussian distribution, specifically $\mathcal{N}(0,1)$, to accurately model how asset prices randomly change over time. The quality of these pseudo-random numbers is very important because it directly affects how accurate and stable the simulation results are. The method works best when these pseudo-random numbers are simulated correctly (independent and identically distributed), which they are in our report, in order to make sure that the simulated paths are realistic and cover the possible market conditions.

For basket options, where the payoff depends on how multiple assets performance together, the Monte Carlo method is very useful because it allows to model changing correlations between asset returns over time and different levels of volatility for each asset. In this report, we use simple pseudo-random number sequences to generate random variables following a $\mathcal{N}(0,1)$ distribution. However, as discussed by Okada et al. (2023), improving the generation of pseudo-random numbers (using advanced models to produce higher-quality numbers) could improve Monte Carlo simulations and make them more reliable in pricing models.

We then introduce the simple Monte Carlo Pseudo-Random estimator evaluating our price such that:

$$\delta = E[h(S)] = e^{-rT}E[max(S(T) - K, 0)]$$

with S being our basket. This estimator will serve as a benchmark to improve throughout this paper.

The Simple Pseudo-Random Method

Let $h : \mathbb{R}^d \mapsto \mathbb{R}$ such that $\mathbb{E}\left[h\left(X\right)\right] = \delta$, given a sequence $\left(X_n\right)_{n\geq 1}$ of i.i.d random variables following the law ν, we define the estimator:

$$\hat{\delta}_n\left(b\right) = \frac{1}{n}\sum_{k=1}^{n} h\left(X_k\right)$$

The law of large numbers unsure that this simple estimator converges to our target.

The figure below illustrates the pricing of an European basket call option using pseudo Monte Carlo simulations. The grey curves represent the evolution of the standard deviation of the price when increasing the number of simulations.

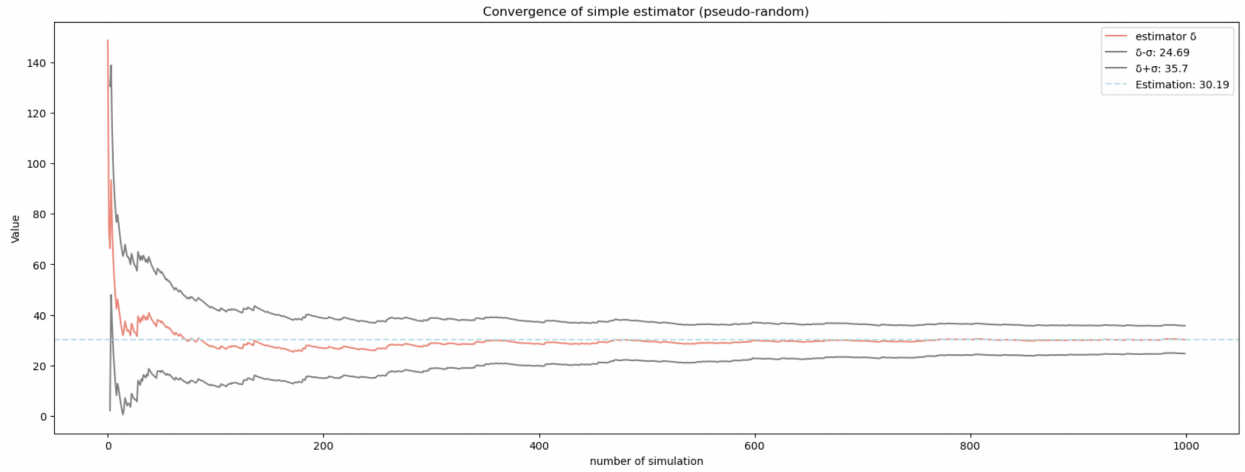


Figure 4: Pricing of European basket call option using pseudo Monte Carlo simulations

Pseudo-random Number Generation

As one understands, random number generations is of upmost importance for Monte Carlo simulations, for which we require a large number of random numbers in order to simulate all the trajectories of our different assets and to obtain a single price. Hence, it is critical to have a reliable source of random number generation. One of this method is the pseudo-random generation for which we know exactly the method used in the generation. As a result, the generation obtained through these methods can be replicated if one knows the correct set of inputs and the method used.

The Linear Congruential Method (LCM):
The Linear Congruential Method is a widely used technique that allows given a seed (X_0), a multiplier (a), an increment (c) and a modulus (m), to generate a sequence of pseudo-random numbers with a length comprised between 0 and the modulus. These numbers are generated recursively using the following formula:

$$X_{n+1} = (aX_n + c) \bmod (m) \quad n = [0, 1, ..., m-1]$$

By dividing the sequence $(X_i)_{i\in[0,m-1]}$ by the modulus, we are able to obtain random float numbers contained in the interval $[0, 1]$. As we see, this approach highly depend on the choice of the input parameters. For the generators to be reliable, we need to ensure that the sequence obtained through these parameters is uniformly distributed on $[0, 1]$, that the first two moments of the simulated sequence i.e. the mean and the variance correspond to the one of the uniform distribution and that the generated numbers are independent. Besides, as we mentioned these types of generators only allow to generate a sequence with a length equal to the modulus. As we require a large number of trajectories, with a substantial number of time steps in our Monte Carlo simulations our project required a more advanced method to generate the uniform distribution. In the LCG, we used by default the following parameters $(a, c, m) = (17, 43, 100)$ and allowed the user the option for a dynamic seed in our option pricer.

Combined Linear Congruential Generators (CLCG):
One way to solve the limitations of the pseudo-random generator previously presented is to combine multiple linear congruential generators with an increment equal to 0 to obtain a more robust uniform pseudo-random number generator. The maximum length of the sequence outputted through the combined generator is then given by:

$$p = \frac{(m_1 - 1)(m_2 - 1)...(m_k - 1)}{2^{k-1}}$$

With k the number of LCG used. A very robust CLCG was formulated by l'Ecuyer (1988), in his paper a pseudo-random uniform generator combining 2 LCG with the parameters is proposed:

$$(a_1\,,\, m_1) = (40.014\,,\, 2.147.483.563)$$
$$(a_2\,,\, m_2) = (40.692\,,\, 2.147.483.399)$$

First, we need to select the seeds of the two LCG, i.e, $X_{1,0} \in [1, m_1 - 1]$ and $X_{2,0} \in [1, m_2 - 1]$. Then we obtain the next number of the sequence in both the generators through:

$$X_{1,n+1} = (a_1 X_{1,n}) \bmod (m_1)$$
$$X_{2,n+1} = (a_2 X_{2,n}) \bmod (m_2)$$

Then we compute:

$$X_{n+1} = (X_{1,n+1} - X_{2,n+1}) \bmod (m_1 - 1)$$

Finally, our generated uniform pseudo random number is given by:

$$R_{n+1} = \begin{cases} \frac{X_{n+1}}{m_1} & \text{if } X_{n+1} > 0 \\ \frac{m_1 - 1}{m_1} & \text{if } X_{n+1} = 0 \end{cases}$$

The number R_{n+1} obtained is our uniform pseudo-random number contained in the interval $[0, 1]$. To obtain a sequence of N realization uniform random variables we iterate through this simple algorithm N times. We rely in all of our project on this method to simulate our uniform pseudo-random numbers. A dynamic seed

option is available to the user in the option pricer for both the LCG used in the Ecuyer Combined method.

Generating random normal variables:
As we have seen, our pricing task requires to eventually be able to generate accurately independent and identically distributed standard Gaussian variables on a large scale. To do so, we implemented in our project different methods each one relying on the sole existence of a uniform random variable generator, the inverse distribution, the central limit, the Box-Muller and the rejection sampling methods. We will present briefly in the following the first three.

The Inverse Distribution method:
Let $F(X) : \mathbb{R} \mapsto [0, 1]$, be the cumulative distribution function (CDF) of the random variable of law ν defined by:

$$F_X(x) = \mathbb{P}(X \leq x)$$

Hence, if F is strictly increasing, continuous and the inverse function F^{-1} of the CDF exists. Then by generating $U \sim \mathcal{U}(0, 1)$, we can generate a realization of the random variable X through $F^{-1}(U)$. This tasks is not simple for the normal distribution as we don't have an expression for F^{-1}. One must rely, on method such as the bisection to compute the CDF numerically. However, for simpler distribution such as the exponential, the task is very simple and require only one uniform random variable to obtain one realization of an exponential random variable.

The Central Limit Method:
The central limit theorem states that for a sufficiently large sequence (X_i) of i.i.d random variables such that $\forall i = 1, .., n \quad \mathbb{E}[X_i] = \mu$ and $\mathbb{V}[X_i] = \sigma^2 < \infty$. Then, we have that :

$$\frac{\sqrt{n}(\bar{X}_n - \mu)}{\sigma} \xrightarrow{d} \mathcal{N}(0, 1)$$

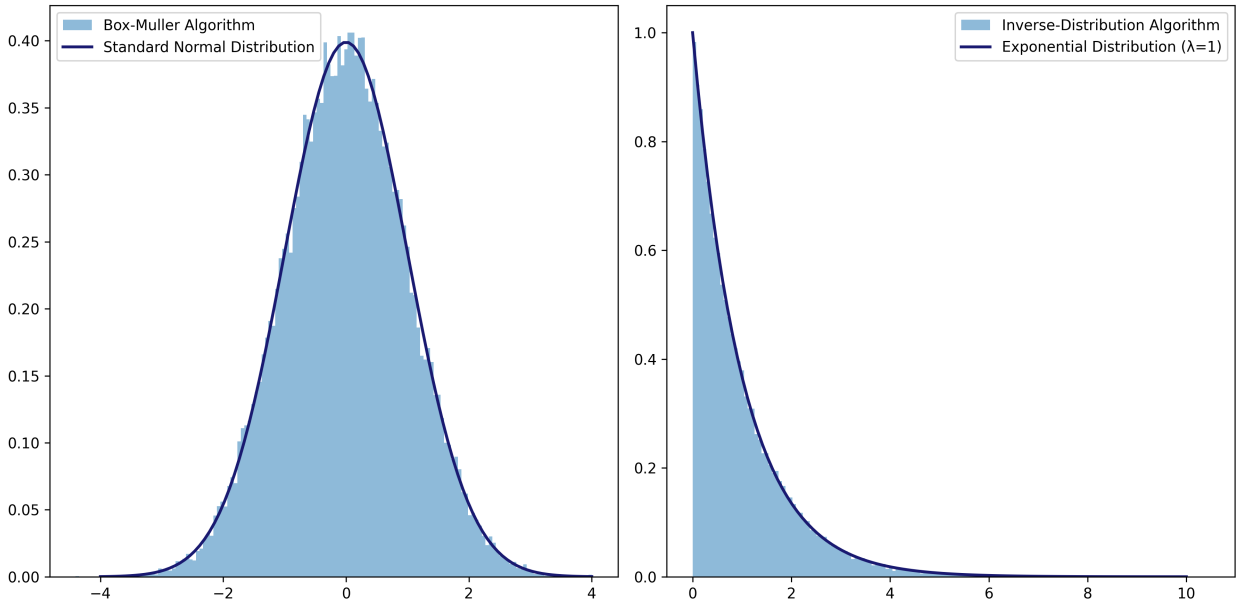
Therefore, by generating sufficiently large number of i,i,d $\sim \mathcal{U}(0, 1)$ random variables, we are able to generate a single realization of a standard normal random variable. We know note though that this task is very computation intensive as we need to have at least 6 uniform random variables to obtain a single realization of a normal and the best approximation of the normal law is obtained with more or less 30 uniform random variables.

The Box-Muller Transformation method:
Finally, a very useful method for generating normal random variables is the Box-Muller transformation method for which require only two realization of independent uniform random variables to obtain two independent realization of standard normal random variables. Given U_1 and U_2 two independent uniform random variables by applying the following formulas:

$$\begin{aligned} Z_1 &= \sqrt{-2ln\,(U_1)}\cos\,(2\pi U_2) \\ Z_2 &= \sqrt{-2ln\,(U_1)}\sin\,(2\pi U_2) \\ Z_1, Z_2 &\sim \mathcal{N}(0, 1) \end{aligned}$$

By generating to two independent uniform random variables through the Ecuyer's method we obtain very simply two independent standard normal random variables. We used the following process to generate or Brownian motions using pseudo-random number in our Monte Carlo simulations, as it proved to be the most efficient and precise method.

These methods are all implemented in the folder Generators. We also coded a fourth method called the rejection sampling method which don't present here. A generator of the exponential distribution and two algorithms used to simulate the Poisson distribution from uniform random variables are implemented as well in our project.



Variance Reduction Methods:

Variance reduction methods contain many strategies that can be used within Monte Carlo simulations to reduce the variance of our pricing process, thus improving pricing accuracy and convergence. These methods are very useful in scenarios where we price complex financial products or when there is a large number of simulations, as they help reduce the computational time and the resources used.

Various types of variance reduction techniques exist, such as antithetic variates, control variates, quasi-random numbers. This section will concentrate on the first two specific methods: antithetic variates, control variates. A comprehensive explanation of these two technique's underlying theory and its integration within the framework of Monte Carlo simulations will be provided. Numerical findings showcasing the efficacy of each approach in curbing simulation output variance and bolstering result precision will be presented later section in this section.

The Antithetic Variate

The antithetic variate method has for goal to reduce the estimation error by introducing symmetry to the problem. To do this, we need the law of the random variable of choice to possess the in-variance properties meaning symmetry and rotation. Meaning that there exist a certain measurable transformation $A : \mathbb{R}^d \to \mathbb{R}^d$ such that, $A(X) \sim \nu$.

The Antithetic Variate Method

A random variable $A(X)$ is called the antithetic variable of X and the estimator of $A(X)$ is defined by:

$$\hat{\delta}_n = \frac{1}{n} \sum_{k=1}^{n} \frac{h(X_k) + h \circ A(X_k)}{2}$$

Where $(X_n)_{n \geq 1}$ is a sequence of i.i.d random variables following the law of X.

Bias of the estimator: The random variables X and $A(X)$ being identically distributed, we have that $\mathbb{E}[h(X)] = \mathbb{E}[h \circ A(X)]$. We obtain that the estimator $\hat{\delta}_n$ has no bias, i.e. $\mathbb{E}\left[\hat{\delta}_n\right] = \mathbb{E}[h(X)]$.

Performance of the estimator: The random variables $(X_n)_{n \geq 1}$ being i.i.d $\sim \nu$ and thanks to the in-variance property (X and $A(X)$ have the same law), we obtain that the variance of the estimator is given

by:

$$\begin{aligned}\mathbb{V}\left[\hat{\delta}_n\right] &= \frac{1}{4n}\left(\mathbb{V}\left[h\left(X\right)\right] + \mathbb{V}\left[h \circ A\left(X\right)\right] + 2Cov\left[h\left(X\right), h \circ A\left(X\right)\right]\right) \\ &= \frac{1}{2n}\mathbb{V}\left[h\left(X\right)\right] + \frac{1}{2n}\mathbb{V}\left[h \circ A\left(X\right)\right] = \frac{1}{2n}\mathbb{V}\left[h\left(X\right)\right](1+\rho) \\ \text{With,} \quad \rho &= \frac{Cov\left[h\left(X\right), h \circ A\left(X\right)\right]}{\mathbb{V}\left[h\left(X\right)\right]}\end{aligned}$$

We observe that with a correlation of $\rho = -1$, we obtain an estimator with a variance that is null. Furthermore, using the fact that:

$$Cov\left[h\left(X\right), h \circ A\left(X\right)\right] \equiv \mathbb{V}\left[\hat{\delta}_n\right] < \frac{1}{2}\mathbb{V}\left[\overline{h}_n\right]$$

We get that if the correlation ρ is inferior to 0, the variance of the estimator obtained through this method is at least divided by 2 compared to the classic Monte Carlo method, or requires two times less simulations to obtain a similar result.

Application to European Basket calls: It is know that if X $\sim \mathcal{N}(0,1)$, then the opposite of X, -X equals X in law. Using this relation, own that $S_T|x$ equals in law $S_T|-x$. The problem then becomes to set here $A : x \mapsto -x$ and injecting into $\delta = \mathbb{E}[(S_{T|-x} - K)_+]$.

Control Variate Method

The variate control method consist in introducing a new variable $h_0\left(X\right)$ and solving a new Monte Carlo problem for which we know the solution.

The Control Variate Method

Let $h_0 : \mathbb{R}^d \mapsto \mathbb{R}$ such that $\mathbb{E}\left[h_0\left(X\right)\right] = m$ is relatively easy to compute and $\mathbb{V}\left[h_0(X)\right] < +\infty$. $\forall b \in \mathbb{R}$, given a sequence $\left(X_n\right)_{n\geq 1}$ of i.i.d random variables following the law ν, we define the estimator:

$$\hat{\delta}_n\left(b\right) = \frac{1}{n}\sum_{k=1}^{n}\left\{h\left(X_k\right) - b\left[h_0\left(X_k\right) - m\right]\right\}$$

Bias of the estimator: As $m = \mathbb{E}\left[h_0\left(X\right)\right]$, we obtain directly that $\mathbb{E}\left[\hat{\delta}_n\right] = \mathbb{E}\left[h\left(X\right)\right]$.

Performance of the estimator: The random variables $\left(Y_n\right)_{n\geq 1}$ being i.i.d, we obtain that the variance of the estimator is given by:

$$\begin{aligned}\mathbb{V}\left[\hat{\delta}_n\left(b\right)\right] = \frac{1}{n}\sigma^2\left(b\right) &= \frac{1}{n}\left\{\mathbb{V}\left[h\left(X\right)\right] + b^2\mathbb{V}\left[h_0\left(X\right)\right] - 2bCov\left[h\left(X\right), h_0\left(X\right)\right]\right\} \\ &= \mathbb{V}\left[\overline{h}_n\right]\frac{1}{n}\left\{b^2\mathbb{V}\left[h_0\left(X\right)\right] - 2bCov\left[h\left(X\right), h_0\left(X\right)\right]\right\}\end{aligned}$$

The estimator $\hat{\delta}_n\left(b\right)$ has a smaller variance than the basic estimator $\overline{h}_n$ if and only if, we are able to find b and h_0 such that:

$$b^2\mathbb{V}\left[h_0\left(X\right)\right] - 2bCov\left[h\left(X\right), h_0\left(X\right)\right] < 0$$

The minimal variance estimator denoted, $\hat{\delta}_n\left(b^*\right)$, is obtained through:

$$b^* = \underset{b\in\mathbb{R}}{\operatorname{argmin}}\mathbb{V}\left[\hat{\delta}_n\left(b\right)\right] = \frac{Cov\left[h\left(X\right), h_0\left(X\right)\right]}{\mathbb{V}\left[h_0\left(X\right)\right]}$$

Hence, we obtain that:

$$\mathbb{V}\left[\hat{\delta}_n\left(b^*\right)\right] = \mathbb{V}\left[\overline{h}_n\right]\left(1 - \rho\left(h, h_0\right)^2\right), \quad \rho\left(h, h_0\right) = \frac{Cov\left[h\left(X\right), h_0\left(X\right)\right]}{\sqrt{\mathbb{V}\left[h\left(X\right)\right]\mathbb{V}\left[h_0\left(X\right)\right]}}$$

Hence, the minimal variance estimator as a smaller variance than $\overline{h}_n$ as soon as $\rho(h, h_0) \neq 0$. The exact gain of the control variate method is dependent of the cost we bear by introducing h_0 to the problem. The correlation coefficient $\rho(h, h_0)$ must be superior in absolute value to the ratio of the cost of evaluating h_0 and the total cost of the estimation problem i.e. evaluating h and h_0.

To determine the optimal coefficient b^*, given a sample of size n, we estimate b^* and so $\hat{\delta}_n\left(\hat{b}_n^*\right)$ through:

$$\hat{b}_n^* = \frac{\sum_{k=1}^n \left(h_0(X_k) - m\right)\left(h(X_k) - \overline{h}_n\right)}{\sum_{k=1}^n \left(h_0(X_k) - m\right)^2}$$

The estimator $\hat{\delta}_n\left(\hat{b}_n^*\right)$ obtained is biased, but with an asymptotic variance equal to $\mathbb{V}\left[\hat{\delta}_n(b^*)\right]$.

Application to European Basket calls: The findings of the correct function h_0 to use in such situation can be quite challenging. We have been advised to use the following relation for $(w_1, \ldots, w_d)$ a vector of normal simulations:

$$h_0(w_1, \ldots, w_d) \mapsto \exp\left(\sum_{i=1}^d \alpha_i \log(X_T^i | w_i)\right)$$

The condition to use such function is to know a deterministic way to determine $m = \mathbb{E}[(h_0(X_1, ..., X_d) - K)_+]$ with $(X_1, ..., X_d)$ a gaussian vector of $\mathcal{N}(0, \Sigma)$.

We admit for proof that the $m = \mathbb{E}[(h_0(X_1, ..., X_d) - K)_+]$ can be given by the following relation:

$$\begin{aligned}
&m = \gamma \exp\left(\tfrac{\hat{\sigma}}{2}\right)\Phi(d_1) - K\Phi(d_2)\\
&\hat{\sigma} = \sqrt{\mu^T \Sigma \mu}\\
&\gamma = \exp\left(\textstyle\sum_{i=1}^d \alpha_i \left(\ln(S_0^i) + (r - \frac{\sigma_i^2}{2})T\right)\right)\\
&d_1 = \frac{\ln\left(\frac{\gamma}{K}\right)}{\hat{\sigma}} + \frac{\hat{\sigma}}{2}\\
&d_2 = d_1 - \hat{\sigma}
\end{aligned}$$

Remark: When implementing the method, we faced a challenge which illustrates the limit of such method. We initially scaled our basket on a basis of 100. This means that when considering some stocks trading at 300$ as we first did, their corresponding weight was around 8% to reach the 33% when adjusted by the spot. The exponential of the sum of the weighted logs comes back to a product of all the spots together at the power of their corresponding α_i. Then, for very low weights, it results in a very low value $h_0(.)$ which makes the estimator gravitate around levels close to 0. This inevitably result for the method to have no effect, hence poor variance reduction performance.
This is why we changed our basket to contain assets trading around similar values and at a higher level (200 vs 100 before) so that the weights could be higher in value and benefit from such techniques.

Results: The results obtained by this method are quite satisfying and can observed alongside others on **??**. Other results from our simulations show that for a set of 1000 simulations, we obtained a two-standard deviation wide interval of 4$ against one of 6.5$ for the simple pseudo random.
Per our parameters, we found that the precision was inferior to the antithetic. We will have a look at all the results together later in the document. We keep in mind that the performance of such method could have been more efficient for higher values of assets as we explained above.

3.3 Pricing using Quasi-Random Numbers

Background: The idea of quasi-random numbers, also known as low-discrepancy sequences, started to develop around the middle of the 20th century thanks to mathematicians Sobol and Halton. These kind of sequences are different from pseudo-random sequences because they aim to fill up space more evenly than

just random points that don't relate to each other. The main goal when using quasi-random points is to make the results of simulations more consistent and accurate compared to using pseudo-random numbers.

Advantages Over Pseudo-Random Numbers
There are multiple reasons why using quasi-random sequences could be better than using pseudo-random numbers:
Reduced Variance: Quasi-random sequences spread out more evenly across the simulation space than pseudo-random numbers do. This helps a lot in reducing the variance of results, making estimates more accurate and stable. This is really good when dealing with problems that have lots of dimensions because it ensures that every part of the space gets enough attention.
Faster Convergence: Thanks to their even spread, quasi-random numbers can speed up how quickly simulations reach their final point. This is super helpful in complex financial models (for example, when pricing Bermudan basket options), where you often deal with many underlyings and getting faster results can save a lot of time.
Improved Accuracy: In situations where you really need to be precise, like in risk analysis or pricing derivatives (as we have here), quasi-random sequences can give you better results with fewer simulation runs compared to pseudo-random numbers. This makes them very good for getting accurate data quickly.

Disadvantages Compared to Pseudo-Random Numbers
However, quasi-random numbers also have some downsides:
Not Really Random: Unlike pseudo-random numbers, which try to act like real random numbers, quasi-random sequences are actually deterministic. This means they don't have the natural randomness some simulations need, like in certain risk modeling situations where unexpected results are part of the process.
Implementation Issues: Making quasi-random sequences that stay evenly distributed in spaces with lots of dimensions is much harder than just making pseudo-random numbers. There could also be issues when dealing with very complex pricing models (when pricing exotic options or structured products for example).

Implementing Quasi-Random Numbers: In our project, we've chosen to implement quasi-random numbers through the use of Van der Corput sequences. Initially, these sequences generate numbers that follow a Uniform(0,1) distribution due to their low-discrepancy nature, filling this interval in a uniform way. We then transform the sequences to follow a Normal(0,1) distribution, which is the same distribution that pseudo-random numbers typically follow.

The transformation from a $\mathcal{U}(0,1)$ to a $\mathcal{N}(0,1)$ distribution is important because our model assumes that underlying asset returns are normally distributed. This assumption aligns with the central limit theorem, which suggests that sums of independent random variables tend toward a normal distribution.
By normalizing the output of the Van der Corput sequences to fit a $\mathcal{N}(0,1)$ distribution, we ensure that our quasi-random numbers can directly replace pseudo-random numbers in our model.

The generation of Van Der Corput sequences can be mathematically defined with the following:

Van Der Corput Sequences

The integer m represented in base b is given by:

$$m = a_k b^k + a_{k-1} b^{k-1} + \ldots + a_1 b + a_0 \tag{2}$$

where each coefficient a_i satisfies $0 \leq a_i < b$.

The m-th term of the Van der Corput sequence, denoted by $\phi_b(m)$, is computed by reversing these coefficients:

$$\phi_b(m) = \frac{a_0}{b} + \frac{a_1}{b^2} + \ldots + \frac{a_k}{b^{k+1}} \tag{3}$$

Furthermore, the graph below compares the distribution of points generated by two different methods: quasi-random numbers from the Van der Corput sequence (on the left), and pseudo-random numbers (on the right).

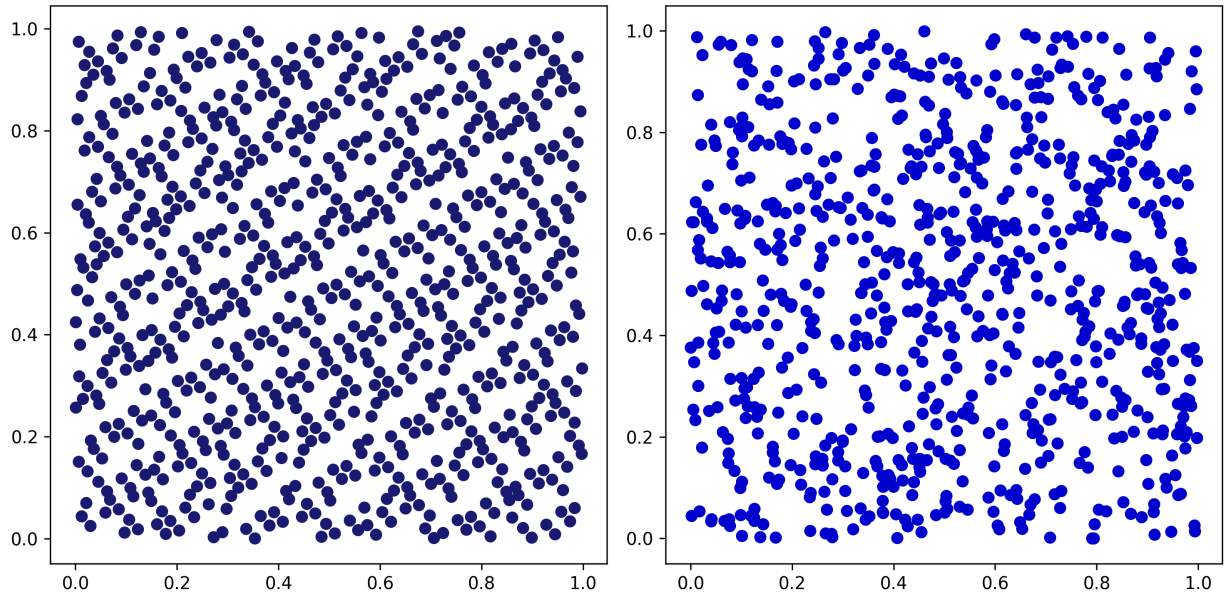


Figure 5: Comparison of Random Number Patterns of Quasi-Random Numbers vs Pseudo-Random Numbers

As we can see, the left graph (Van Der Corput) shows a pattern where points cover the space without overlapping or leaving larger empty areas. This helps reduce the error in computations where you need to simulate many different scenarios.

On the other hand, the right graph (pseudo-random) displays a typical random scatter of points. Here, some areas contain more points while other areas have much less. This distribution which is not uniform can lead to less accurate prices and slower convergence towards the correct price.

3.4 Performance comparison

The figure below compares different estimators when using quasi-random numbers.

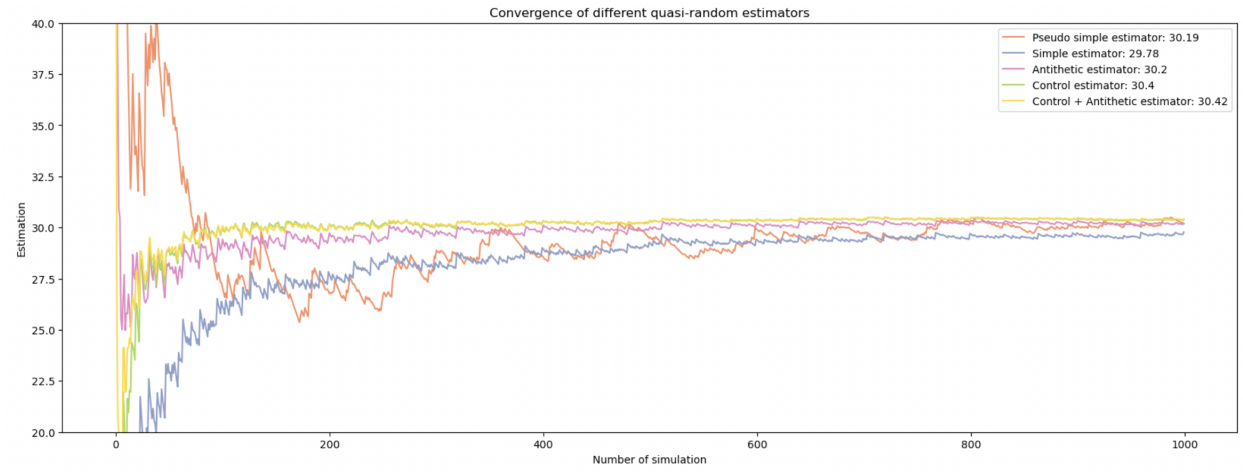


Figure 6: Comparison of Quasi-Random estimations for several methods

As we can see, our benchmark (the pseudo-random estimator) is the worst out of all the estimators since it shows high volatility and only converges to its estimated value at around 900 simulations, much more

than other estimators. We also notice that our simple quasi-random estimator is biased towards the negative values, and converges a bit slower than the other estimators while having an estimated value of 29.78, 0.5 lower than the average estimated value. As for our best estimator, we have 2 (very similar candidates): control and control + antithetic estimators which both show high initial volatility in the estimation process but then quickly converge to the estimated value of around 30.4. Control + antithetic being the best estimator is what we expected as combining multiple variance reduction techniques has a better chance in reducing variance and results in faster convergence, which is what we see here: control + antithetic takes around 200 simulations to stabilize, making it the most stable and efficient method.

Now, we will look at each estimator in more detail in the graph below.

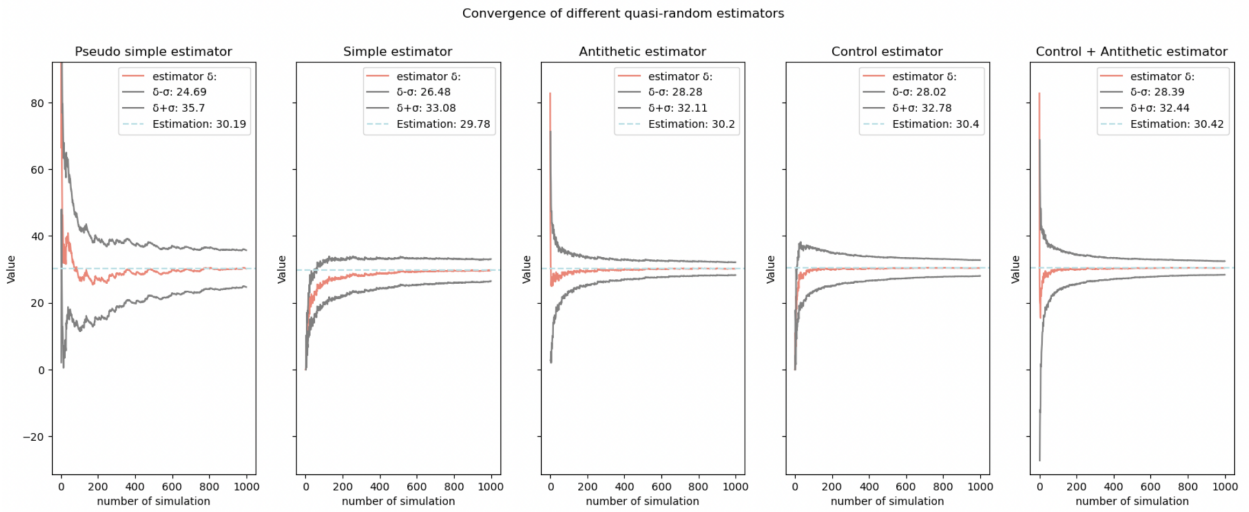


Figure 7: Comparison of Quasi-Random estimators precision given a number of simulations

Initially, the pseudo-random simple estimator demonstrates rapid convergence but exhibits some volatility and slight bias above the true value, suggesting an initial overestimation that stabilizes as the number of simulations increases. Compared to this, now looking at quasi-random numbers, the simple estimator shows smoother convergence and maintains a tighter confidence interval around the true value with less fluctuations. The antithetic estimator further improves on these aspects, quickly converging to the true value with very tight confidence intervals. The control estimator has wider confidence intervals towards the end of the simulations, meaning there is less precision compared to the antithetic estimator. Finally, the combination of control and antithetic techniques also results in tight confidence intervals, but the initial volatility is quite high. Overall, these results are consistent with expectations, as combining multiple variance reduction techniques generally results in better variance reduction in Monte Carlo simulations. Among all the estimators, the combined control and antithetic estimator seems to the best since it convergences to the estimated value very quickly and also has tight confidence intervals.

The table below shows our prices and volatilities for European basket call options using different number generators (pseudo-random and quasi-random) as well as different estimators (simple, control, antithetic and control + antithetic).

Table 6: Monte Carlo for European basket call option performance comparison[5]

Estimator	Generator	Price	Std
Simple	Pseudo-Random	30.75$	3.18$
Control	Pseudo-Random	29.22$	2.17$
Antithetic	Pseudo-Random	30.38$	1.22$
Control + Antithetic	Pseudo-Random	29.21$	1.14$
Simple	Quasi-Random	29.78$	2.82$
Control	Quasi-Random	30.4$	1.95$
Antithetic	Quasi-Random	30.2$	1.14$
Control + Antithetic	Quasi-Random	30.42$	1.24$

6

The graph serves as an illustration of the results from the table above.

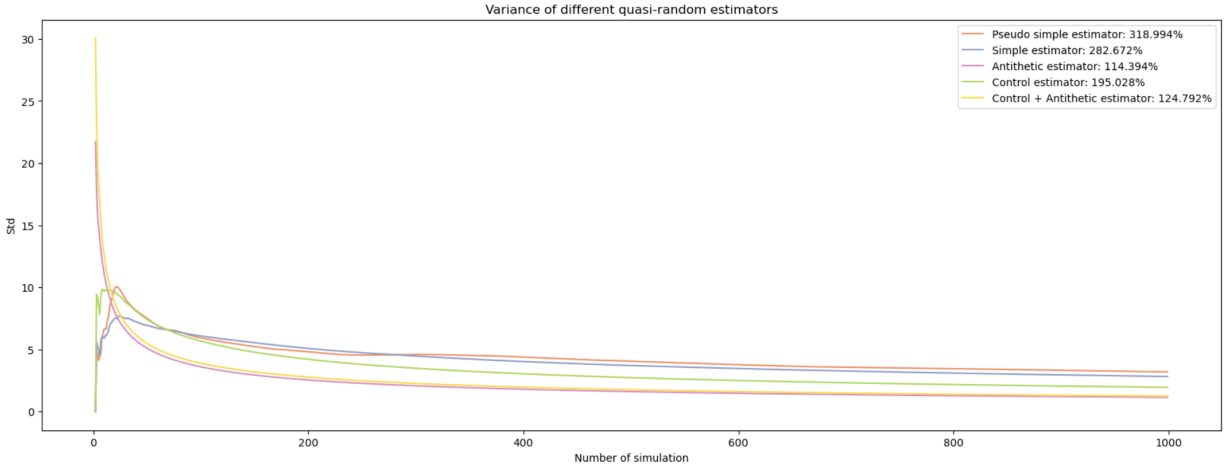


Figure 8: Variance comparison of different Quasi-Random estimators

As we can see in the graph above, the pseudo-simple estimator initially doubles its variance from 5 to 10 standard deviations within the first 50 simulations and continues to show the highest variance among all other estimators throughout the simulations. This result is indeed expected for simpler methods that lack advanced variance reduction techniques. On the other hand, the quasi-random simple estimator shows a much smaller initial variance jump than its pseudo-random counterpart, demonstrating the benefit of uniform quasi-random sequences. The quasi-random antithetic estimator, while having the second-highest initial variance, outperforms all other estimators by quickly dropping in variance from the beginning and resulting in the lowest final standard deviation after 1000 simulations at around 2.5 standard deviations. Like with other estimators, its variance stabilizes after 200 simulations. This performance is thanks to the antithetic method's nature of pairing each simulation path with a negatively correlated path, which can cancel out a lot of the randomness and significantly reduce variance. The quasi-random control estimator and the combined control + antithetic estimator also show low variance levels after 1000 simulations, with the combined estimator method slightly outperforming the control estimator.

[6]the number of simulations used to achieve the results is 1000

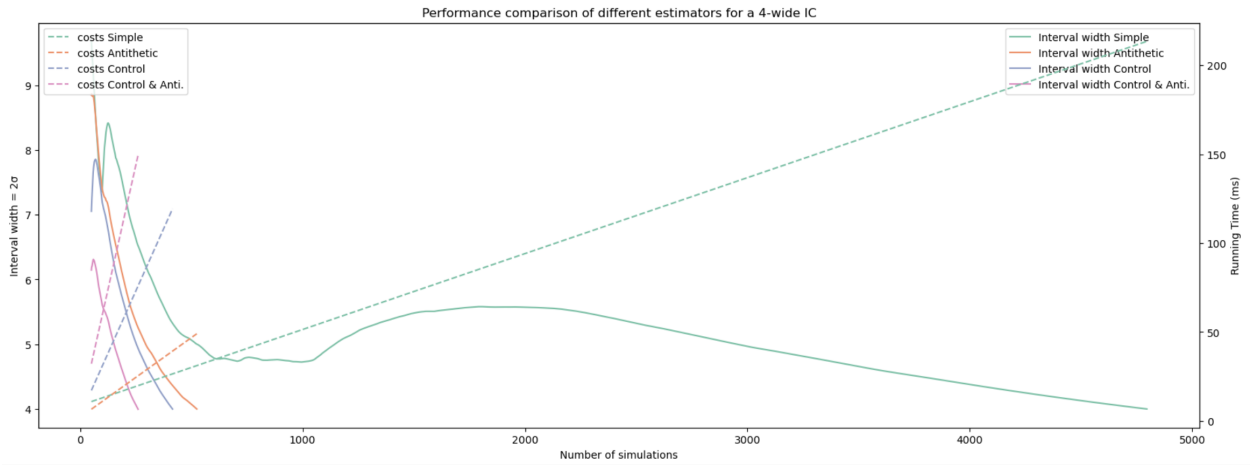


Figure 9: Cost and performance comparison between different estimators

When looking at the results above, we found that some techniques are much more efficient at reaching a particular level of precision for a given amount of simulations. However, the graph above proves that this precision given a limited amount of simulations has a cost, that is a computational cost.
The graph represents the computational cost (here the running time in milliseconds) to reach a an interval of 2 standard deviations in the results that is less than 4 in our case here.

The results are very interesting and as we could expect. The least precise method (basic Monte Carlo) needs around 5000 simulations to reach this interval where the best method (control + antithetic variate) needs only 200, hence being 25 times faster in convergence in our case. However, the same function is the one that takes the longest time to run given a number of simulations. Indeed, given the antithetic property, it has to repeat the process twice on the given set of simulations and the one shifted by the function A. Then, the control variate side of the estimator computes the value with function h_0 and estimate the best optimal control parameter b with the so-simulated variables.

There is definitely a trade-off between the growing linear cost of the best estimation method and one that reaches the correct amount of variance given a number of simulations such as antithetic method, providing the best variance-to-cost ratio.

4 Pricing & Variance Reduction methods for Bermudan basket options

Now, we will see how to price Bermudan basket options and how the variance reduction techniques we saw previously will influence this pricing. This time, instead of using simple Monte Carlo simulations like before, we will use the Longstaff-Schwartz method, proposed by Longstaff and Schwartz in 2001 (which we call Monte Carlo Longstaff-Schwartz). This pricing method is mainly used for pricing American options. Also, instead of cumulatively including quasi-random numbers, control variates and antithetic variates, we will combine all the methods right away and analyze the impact this combination has on both the price and the variance.

4.1 Introduction to Bermudan Basket Call Options

Bermudan basket call options are more complex financial derivatives than European basket call options. Like European basket options, they give the right to buy a mix of assets, but with an important difference: Bermudan options can be used on certain dates before they end, not just when they end. This makes them more flexible and generally more expensive than European options, but also harder to price.

Like for European options, the basket in a Bermudan basket call option typically consists of stocks or indices, and the payoff depends on the combined performance of all assets in the basket.
The payoff of a Bermudan basket call option can be described by the formula:

$$C(t) = e^{-r(T-t)} * max(S(t) - K, 0) \tag{4}$$

where:
- $C(t)$ is the value of the call at any allowable exercise time t
- t represents any of the predetermined dates on which the option can be exercised,
- T is the final expiration date,
- r is the risk-free interest rate,
- $S(t)$ is the weighted sum of the asset prices in the basket at time t,
- K is the strike price of the option.
- $e^{-r(T-t)}$ is the factor which discounts the payoff at time $t \leq T$ to its present value.

This flexibility to choose the exercise time makes the pricing of Bermudan options more complex as it requires optimization over multiple potential exercise dates.

4.2 Pricing using Longstaff-Schwarz Monte Carlo Simulations and Pseudo-Random Numbers

When holding the exercise right to a Bermudan/American option, an investor holds the right to earn money without having to reach maturity of the option.
In mathematical terms, this means that:

$$\forall t_i \in t_0, ..t_n, V_{t_i} = max(\Phi(S_{t_i}), E[V_{t_{i+1}}|S_{t_i}])$$

Meaning that the value of the option at a time is the maximum between exercising the right at the corresponding period and not taking it. As a result, the objective is to determine at a period whether it is optimal to exercise the right, hence the problem comes back to estimate the value $E[V_{t_{i+1}}|S_{t_i}]$ being the expected gain value at the next step knowing the spot today.

The objective of the Longstaff-Schwartz method is to estimate the value of the next optimal decision time using linear regression. In a very basic way, the algorithm intends to fit the value using basic polynomials.

In our work, we set L the maximum degree of the polynomial to 2 and the polynomial being:

$$Px, i \mapsto x^i$$

with x the value and i the degree of the polynomial from 0 to L included.

Here is how the algorithm works in detail

Table 7: Brownian Motion Paths

Period	0	1	2	3	4
Path 1	200	157	193	189	146
Path 2	200	191	219	161	271
Path 3	200	170	162	212	284
Path 4	200	227	155	246	292
Path 5	200	253	181	299	212

The algorithm iterate backwards given a set of paths as in 7, starting on the last values simulated for each asset path and iterating until t_0.

Table 8: Terminal payoff function applied to Brownian Motion Paths

Period	0	1	2	3	4
Path 1	0	0	0	0	0
Path 2	0	0	19	0	71
Path 3	0	0	0	12	84
Path 4	0	27	0	46	92
Path 5	0	53	0	99	12

At maturity, the value we set is simply the payoff applied to the terminal node. The algorithm is to be applied at each time of the exercises periods and keep iteratively a set of optimal decisions values that we iterate on.

Let's consider the concrete example of the iteration process at time t=3: We first by applying the final payoff to each of our node (especially on the period we are looking hence t=3) and obtain the In-the-money table above 8.

From there, we extract the sub matrix by keeping only the assets values that are in the money. This corresponds in our example to paths 3, 4 and 5.

Table 9: Linear regression for optimal decision at time $t = t_3$[7]

Path	S_{t_3}	$S_{t_3}^2$	$e^{-rT}V_{t_4}$
3	212	44944	70
4	246	60516	77
5	299	89401	1

Using the three paths we found, we then build the sequence of value using our polynomial sequence up to L dimension. In the meantime, we actualise the expected payoff at next step, which represents in today's money what is the expected profit we are likely to obtain.

[7] r is supposed to be 5% in our examples.

$$\underbrace{\begin{bmatrix}1 & 212 & 44944\\ 1 & 246 & 60516\\ 1 & 299 & 89401\end{bmatrix}}_{X}\underbrace{\begin{bmatrix}\alpha_0\\ \alpha_1\\ \alpha_2\end{bmatrix}}_{\alpha} = \underbrace{\begin{bmatrix}70\\ 77\\ 1\end{bmatrix}}_{Y} \longrightarrow \underbrace{\begin{bmatrix}\text{-156.662}\\ 0.426\\ 0.000058\end{bmatrix}}_{\alpha^*}$$

Then, as mentioned previously, the objective is to estimate the expected value at the next time given the value at our node. This comes back to estimate the value of the vector Y denoted above in 4.2, given the polynomial applied to the current spot values resulting in matrix X. By running the ordinary least squares methods on the problem, we find the following coefficients stored in $\alpha*$

Table 10: Continuation decision values at $t = t_3$

Path	$\Phi(S_{t_3})$	$E[\Phi(S_{t_4})\|S_{t_3}]$	$e^{-rT}V_{t_4}$
1	12	$(-63)_+$	70
4	46	$(-47)_+$	77
5	99	$(-23)_+$	1

The last step comes to finally make a decision regarding the highest value between the $\Phi(S_{t_i})_{i>1}$ and the $E[\Phi(S_{t_4})|S_{t_i}]_{i>1}$. To compute the the vector of expected value we simply multiply matrix X by $\alpha*$. In our example, it means that it is never optimal to stay within the product and that you should exercise your right of option. When integrating, the value at each node represents the maximum between the two. And we repeat the process up to $t = 0$.

Please note that the values of OLS are only indicative here as we took only three examples and such situations should only be evaluated for a high number of simulations.

4.3 Pricing combining different Variance Reduction techniques

We implemented different variance reduction methods for Bermudan options, namely antithetic and control variate methods.

Antithetic: The Antithetic transformation for the Bermudan options operates similarly as before: obtaining a mirror diffusion for the asset, which for a same number of gaussian variables simulated gives us twice the number of simulations. The process is the following: when we simulate the two sets of paths, we operate the Longstaff-Schwarz algorithm described in the above section for the two sets of data and we end by aggregating the results. The results are visible in the table below and show great reductions results.

Control Variate: The Control variate method has been more challenging to implement in the framework of Bermudan assets. With poor assistance from the literature on the subject, we decided to apply the $x \mapsto -b(h_0(x) - m)$ shift to each value obtained after iterating backward down to t_0. One solution could have been, similarly to the antithetic method, to run the ordinary least square approach on the log path dynamic for our assets. However, knowing that due to convexity they are below in value, it makes less sens to base an optimal decision on this processes. This is why we only applied the shift at t_0. Analysing the results from the graphs and table below challenge us to give a new approach to this method as the results are improving the simple estimator.

4.4 Performance comparison

Table 11: Monte Carlo for Bermudan option performance comparison[8]

Estimator	Generator	Price	Std
Simple	Pseudo-Random	30.96$	3.37$
Control	Pseudo-Random	30.07$	5.99$
Antithetic	Pseudo-Random	32.63$	1.74$
Control + Antithetic	Pseudo-Random	30.07$	5.99$

[9]

The figure below compares different estimators when using pseudo-random numbers.

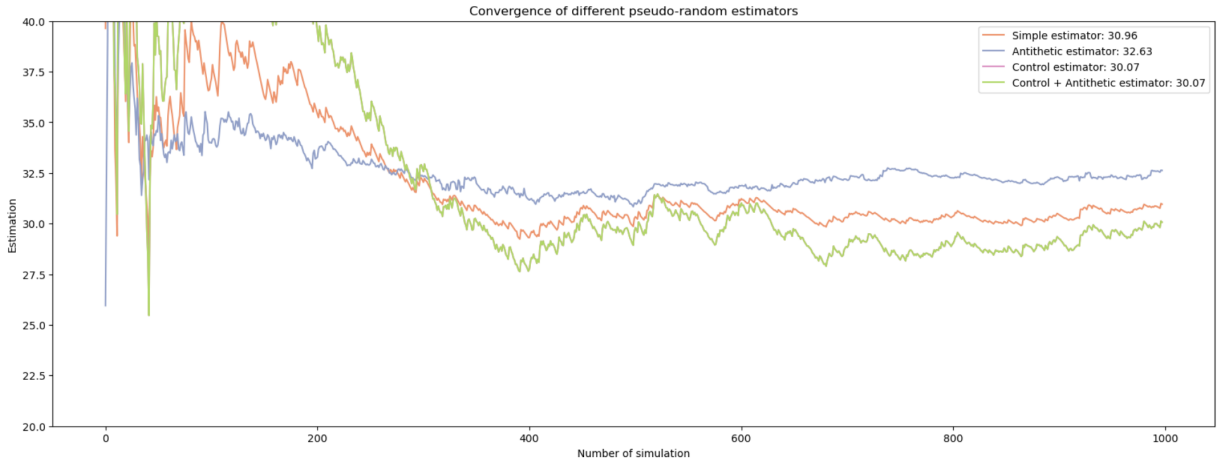


Figure 10: Comparison of Pseudo-Random estimations for Bermudan Basket Call

We notice that the simple estimator shows significant initial volatility, normal for methods that don't use variance reduction techniques, and stabilizes only as the number of simulations increases. In contrast, the antithetic estimator demonstrates the most stable and accurate convergence among all the estimators due to its effective use of negatively correlated simulation paths that cancel out random fluctuations, resulting in a smoother and more consistent approximation of the true value. Surprisingly, both the control estimator and the combined control + antithetic estimator overlap completely and display very slow initial convergence and stay quite volatile, stabilizing only at around 800 simulations and converging to a lower final estimation value than both the simple and antithetic estimators. This overlap suggests that adding antithetic variance reduction to the control variate does not enhance its performance, potentially because combining the two techniques can be challenging when using Monte Carlo Longstaff-Schwartz to price Bermudan basket call options.

Now, we will look at each estimator in more detail in the graph below.

[9] the number of simulations used to achieve the results is 1000

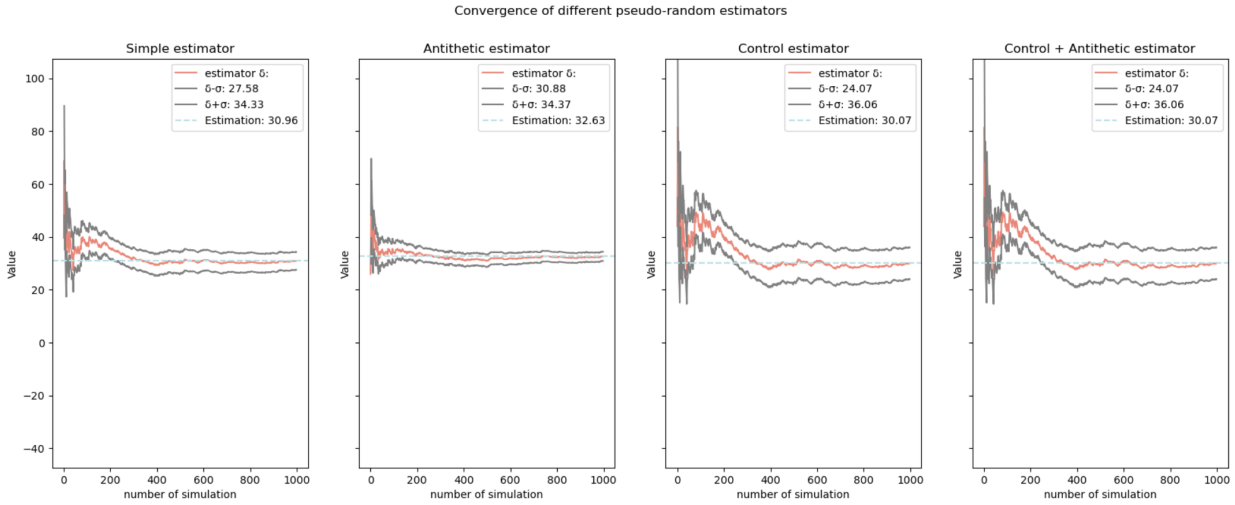


Figure 11: Comparison of Pseudo-Random estimators given a number of simulations

For the Bermudan basket calls, the simple estimator shows rapid convergence with high volatility early in the simulation. This volatility reduces as the number of simulations increases. The antithetic estimator for Bermudan calls demonstrates a much better performance over the simple estimator with quicker stabilization and tighter confidence intervals. However, its convergence to the true value isn't as smooth as seen with the European basket calls, illustrating the higher pricing complexity of Bermudan options. The control estimator, while eventually converging to a value close to the true estimated value, shows much more volatility and wider confidence intervals than the antithetic estimator. Also, the combined control and antithetic estimator displays (disappointing) results similar to what we saw with the control estimator. This indicates that combining these two techniques doesn't reduce the pricing variance, which is most likely caused by the complex payoff structure and multiple exercise dates of Bermudan options. Overall, the results don't exactly align with our expectations since combining multiple variance reduction techniques doesn't offer the best results in our Monte Carlo simulations. Actually, the antithetic estimator is the best performer, offering rapid convergence and maintaining precision across the simulations.

We now compare the variance of different pseudo-random estimators within our MCLS framework.

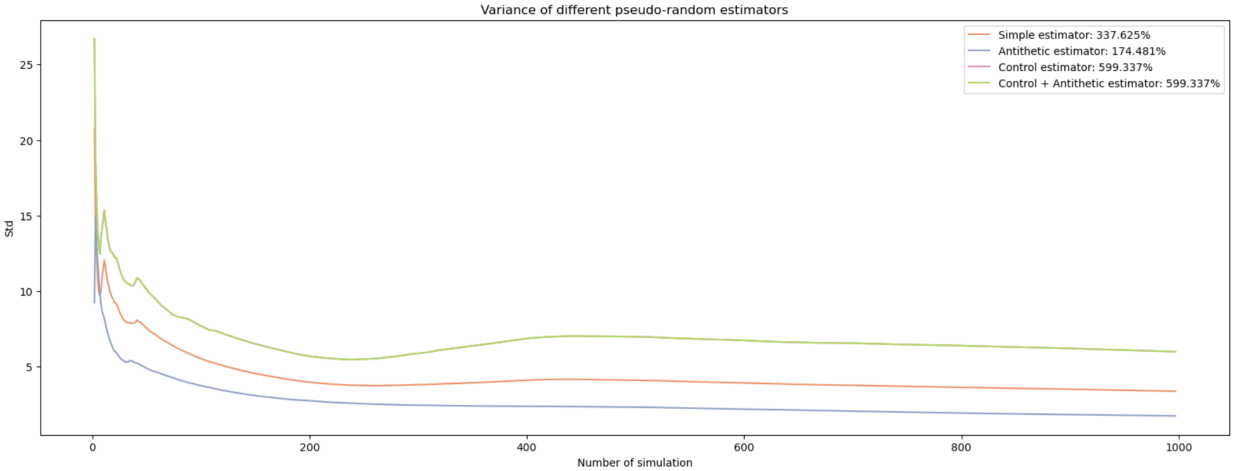


Figure 12: Variance of different pseudo-random estimators

We observe that the simple estimator starts with a high variance, showing significant initial fluctuations, but it quickly decreases and stabilizes as we continue with simulations. The antithetic estimator demonstrates the lowest initial variance and a smoother, quicker convergence compared to all the other estimators, indicative better variance reduction. Interestingly, both the control estimator and the combined control +

antithetic estimator overlap and they show the highest variance at the start. This outcome is what we saw in the previous graph, and it is not consistent with expectations, as combining these estimators should result in better variance minimization. Finally, the identical performance of the control and combined estimator suggests that adding antithetic variance reduction to the control estimator does not further reduce variance in this scenario.

5 Conclusion

5.1 Summary of Results

Difference between European & Bermudan basket call options: Contrary to single stock options, the call of a basket option is not the same whether it's an European exercise or an American/Bermudan exercise. In a framework with a single assets, we can benefit from time effect by being fully exposed through a call option. However, in a multi dimensional framework with assets correlated to each other, we could have expected a more significant difference between the European and the Bermudan option prices.

The correlation effect: Another aspect we barely mentioned throughout this study is the impact of correlation. The more the stocks are correlated, the more expensive the option becomes. The option is long correlation, partially due to the correlation effect in the basket volatility. The higher the correlation, the higher the volatility of the basket is and by the long vega exposure on the call. The basket we chose is composed of U.S. large companies with a high realized correlation in 2023 (between 80% and 90% between each pair), which we used as a pricing parameter and which can may differ from the implied correlation. The payoff being only exposed on the upside (downside limited to premium) and the exposition of the basket to correlation makes the results more meaningful as an investor.

5.2 Limitations and Potential Improvements

Pseudo-random numbers: In our project, we use a simple method to generate pseudo-random numbers by taking numbers from a standard normal distribution $N(0, 1)$. While this approach is simple and works well, it could be improved by making it "more random" which could significantly improve the accuracy of our Monte Carlo simulations.
For instance, a study by Gafsi et al. (2022) shows how using "chaotic maps" can generate high-quality pseudo-random numbers. Another approach by Okada et al. (2023) involves using machine learning methods to produce statistically robust pseudo-random numbers. These studies suggest that using more advanced methods in generating pseudo-random numbers can improve the Monte Carlo simulations by making sure that the random sequences better resemble the randomness of the stock market in the real world, thus improving the pricing of financial models.

Increasing the number of simulations: The results presented in tables and graphs above are only containing a relatively small number of simulations set to 1000. This number was a good trade-off between achieving representative results, a relatively limited cost of computations and a good visual representation with the cumulative standard deviation plotted. When running the tests, we elevated the number of simulations up to 1,000,000 and observed a significant convergence between the different methods, with the deterministic price function when pricing in one dimension from the Black & Scholes model as a benchmark.

6 Bibliography

[1] L'Ecuyer P. (1988): *Efficient and portable combined random number generators*, Communications of the ACM, Vol. 31(6), Pages 742-751.

[2] Gafsi M., Abbassi N., Amdouni R., Hajjaji M. A. and Mtibaa A. (2022): *Hardware implementation of a strong pseudo-random numbers generator with an application to image encryption*, 2022 IEEE 9th International Conference on Sciences of Electronics, Technologies of Information and Telecommunications (SETIT), Pages 510-515.

[3] Okada K., Endo K., Yasuoka K. and Kurabayashi S. (2023): *Learned pseudo-random number generator: WGAN-GP for generating statistically robust random numbers*, PLoS One, Vol. 18(6).

[4] Stoehr J. (2022): *Méthodes de Monte Carlo*, Département MIDO, Master 1 Mathématiques, Pages 15-25.