POLITECNICO DI TORINO

Master's Degree in Mathematical Engineering

Master's Degree Thesis

Valuation Risk in Margin Loan Pricing

Supervisor:

Prof. Patrizia Semeraro

Co-Supervisors:

Dr. Diego Giovannini

Dr. Danilo Barretta

Candidate:

Ornella Elena Grassi

Academic Year 2022-2023

Summary

Margin loans have gained increasing attention as financial instruments tailored for newly listed firms such as startups or small emerging companies. These loans, provided by banks as financing instruments, utilize a specific number of company-owned shares as guarantee (aka "collateral"), valued higher than the loan amount at the beginning of the effective date. Like margin accounts, should the value of the company's shares decline, the borrower is required to provide additional shares to cover the shortfall and meet the stipulated collateral requirements. This thesis focuses on a comprehensive analysis of the financial risks associated with margin loans, with a particular emphasis on monitoring the performance of the company's shares following the loan agreement, whose price strongly depends on relevant market shocks. To achieve this, we delve into the study of renowned stochastic processes for simulating stock price movements in the stock market, specifically, Lévy processes. In this context, jump models become particularly relevant, proving invaluable for generating market scenarios to estimate the value of a shortfall event. We will conduct a comparative analysis of different pricing models: the Merton's Jump Diffusion model, Kou's Double Exponential Jump Diffusion model, and the Variance Gamma process proposed by Madan and Seneta. As part of Margin Loan pricing, whether through Monte Carlo simulations or closed formulas, an

essential aspect is the calibration of the model parameters to the market-priced option data. This involves fine-tuning the model's internal settings to accurately replicate market prices, ensuring coherence of the model with market predictions. In detail, we will use both European plain vanilla options and a unique category of exotic options known as One-Touch Knock-Out Daily Cliquet Options, which are suitable for capturing significant fluctuations in stock prices. The calibration of these parameters will be addressed as an optimization problem, and we will employ the Non-Linear Least-Squares algorithm to solve it. Tackling the subject of margin loan pricing using various models will enable us to assess the intricacies of pricing these instruments and the inherent valuation risk linked to model choices.

Keywords: Margin Loans, Lévy processes, Variance Gamma, Merton Jump Diffusion, Kou Jump Diffusion, option pricing, calibration, vanilla otpions, one-touch knock-out daily cliquet options

Acknowledgements

I would like to express my sincere gratitude to my university professor, Patrizia Semeraro, for giving me the opportunity to develop this project in such a prestigious company. Thank you for your help and endless comprehension; they were decisive in achieving my goals.

I would also like to thank my supervisors at *Intesa Sanpaolo*, Dr. Diego Giovannini and Dr. Danilo Barretta, for their assiduous guidance and mentorship, which were essential in the success of this project.

To my family

Table of Contents

List of Figures

Acronyms

APR

Annual Percentage Rate

BS

Black-Scholes

GBM

Geometric Brownian Motion

ITM

In-the-Money

KJD

Kou Jump Diffusion

LTV

Loan-To-Value

MJD

Merton Jump Diffusion

NLS

Nonlinear least squares

OTKO

One-Touch Knock-Out

OTM

Out-of-the-Money

SPV

Special Purpose Vehicle

VG

Variance Gamma

Chapter 1

Introduction

Margin loans have emerged as vital financial instruments tailored for startups and emerging companies. This thesis delves into a comprehensive analysis of the inherent financial risks associated with margin loans, with a particular focus on their evaluation and accurate pricing.

In contrast to traditional loans, where banks primarily face credit risk – the risk that the borrower may not repay the debt – margin loans shift the risk to market performance. In this scenario, the bank must contend with the post-loan agreement market performance of the beneficiary company. To address this problem, we will employ stochastic processes, specifically Lévy processes, to simulate stock price movements in the stock market. These processes, including jump models, facilitate the estimation of potential shortfall events. In Chapter 2, we provide an overview of margin loans, starting with the concept of loans themselves. We will explore how margin loan contracts work and explain the theoretical instruments necessary for our analysis. Chapter 3 explains the stochastic processes used in this thesis and contains their theoretical background and description. We emphasize their

significance in modeling stock price movements. Then, chapter 4 shows how to realize Monte-Carlo simulations of these processes, that will be needed in subsequent chapters in the context of option pricing and margin loans pricing. We thus discuss methods for pricing options using both closed-form solutions and Monte-Carlo simulations in Chapter 5. We will explain how option pricing methods allow us to calibrate models parameters to ensure accurate process simulations. In Chapter 6, we will show the results of our calibration procedure, illustrating how the outcomes strongly depend on the set of parameters chosen. Finally, Chapter 7 will explain how to price our margin loans given all the steps previously discussed, drawing conclusions in Chapter 8.

Chapter 2

An overview of Margin Loans

Margin loans are specialized financial instruments offered by banking institutions to corporations. They fall within the broader category of bank loans, primarily utilized by small businesses for financing projects related to financial investments and growth. This chapter serves as a foundational exploration, commencing with a general overview of loans. It aims to explain their key features, types, structural aspects, and how they are managed by lending institutions. Subsequently, our focus shifts to the central theme of this work—margin loans. We will examine their background, characteristics, and distinctive traits, showing how they involve an additional layer of complexity and risk. Furthermore, we will delve into the mathematical framework essential for assessing and effectively managing these associated risks.

2.1 Loans

A classic loan, in its essence, is a financial arrangement in which a lender, which may be a corporation, financial institution or government, lends a sum of money to an individual or other entity, creating a form of debt. This allows individuals or businesses to access capital that they may not have immediately available. In this agreement, the borrower consents to a set of specific terms, including finance charges, interest rates, repayment dates, and other conditions. Loans can take a variety of forms, including bonds and certificates of deposit (CDs), and may also request collateral to protect the loan and guarantee repayment. [1]

2.1.1 Role and Impact

A loan contract is an agreement between the lender, who provides the funds, and the borrower, who receives a specific sum known as the principal. The borrower commits to repay this principal over a specified period, often with the added cost of interest. This interest acts as compensation for the lender, accounting for both the risk associated with granting the loan and the opportunity cost of lending that money rather than investing it elsewhere. Loans have diverse applications. They can serve to finance the expansion of a business, facilitate the purchase of a residential property, or address unexpected financial needs. In effect, the existence of loans expands an economy's overall money supply while fostering competitiveness by providing funding opportunities for emerging businesses. In addition, interest and fees on loans are a major revenue for many financial institutions. From the banks' perspective, loans not only generate income, but can also help diversify the risk associated with their portfolio. In this way, banks benefit from a steady

revenue stream and a well-balanced portfolio, making lending an integral part of their business model [1].

2.1.2 Target Market

Loans are broadly classified into personal and commercial categories.

Personal loans are typically obtained by individual borrowers and could encompass mortgages, car loans, student loans, home equity lines of credit, credit cards, installment loans, and payday loans [2]. The main risk a bank face when agreeing a loan to individuals is mainly credit risk, the risk that the borrower will default on their loan repayments. This risk is generally managed by assessing an individual's creditworthiness before making a loan. Creditworthiness is determined by factors such as credit history, income, job stability, and existing debts. A good credit score and a stable financial profile generally make it easier to qualify for loans with favorable terms.

Commercial or *business loans* share similarities with personal loans, but generally involve larger amounts and different underwriting procedures. Examples are commercial mortgages, corporate bonds, and government guaranteed loans. The underwriting process is more rating-based than score-based. In this paper, we will focus on bank loans, i.e., the financing instruments that banks provide to companies for expansion and capital investment, debt refinancing, to cover short-term expenses or facilitating the acquisition of other businesses or assets.

2.1.3 Types of Loans

There are several types of loans, but they can generally be categorized based on two key criteria [3]:

1. Secured vs. Unsecured Loans:

   - *Secured Loans*: These loans require collateral, such as assets or property, to secure the loan. If the borrower fails to repay, the lender can seize the collateral.
   - *Unsecured Loans*: this type of loan is not backed by collateral, but is granted only on the basis of the borrower's creditworthiness, income, and ability to repay. Since there is no collateral, they generally have higher interest rates than secured loans.

2. Revolving vs. Term Loans:

   - *Revolving Loans*: Revolving credit lines, like credit cards, allow borrowers to access funds up to a set credit limit. Borrowers can repay and reuse the credit as needed.
   - *Term Loans*: Term loans provide borrowers with a lump sum of money upfront, which is repaid over a predetermined period with fixed installments.

2.1.4 Components of a Loan

This section aims to elucidate general key components of a loan, each of which plays a crucial role in defining the terms and conditions [4].

1. *Principal*: it is the original sum of money lent to the borrower. The principal amount is determined at the onset of the loan and it serves as the base upon which interest accumulates.

2. *Interest*: this is the cost of borrowing money, generally expressed as a percentage of the principal. It could be calculated in different ways, but the most common are simple and compound interest.

3. *Repayment Schedule*: a detailed plan of how the loan would be repaid. It includes the frequency (monthly, quarterly, annually, etc.), number of installments, and the amount of each installment. The schedule also outlines how much interest and principal is paid off with each payment.

4. *Fees*: These pertain to any additional charges associated with the loan. Common fees include origination fees, late fees, and prepayment penalties.

5. *Loan Term*: it refers to the duration over which the loan is expected to be paid back. Short-term loans usually run for less than a year, while long-term loans can last a few years to a few decades. The term typically affects the size of the monthly payment and the total cost of the loan.

6. *Collateral*: required by secured loans - a valuable asset that the borrower agrees to give to the lender if they can't repay the loan. Typical examples of collateral include cars, houses, and other valuable assets, like a company's shares.

7. *Loan Agreement*: the legal document that outlines the terms and conditions of the loan, including the obligations of the borrower and lender, and potential consequences of the violation of these terms.

2.1.5 Metrics

Interest rates

Interest rates are directly proportional to the amount of risk associated with the borrower. Interest is charged as compensation for the loss caused to the asset due to utilization, effectively indicating the cost of borrowing. Interest rates have the flexibility to be applied over varying intervals, including monthly, quarterly, or semiannually, but the prevalent practice involves them to be annualized.
Additionally, they can be fixed if they remain constant for the entire term of the loan and variable or floating if they can change periodically based on market conditions or specific indexes. They can be divided into two main categories: simple and compound interest rate [5].

Simple interest: the type of interest indicating that the total expense for borrowing money (i.e., the simple interest) is computed only on the principal amount and it remains constant throughout the loan term, according to the formula

$$SI = P \cdot r \cdot n \tag{2.1}$$

where P is the principal, r is the interest rate and n is the term of the loan.

Compound interest: it takes into account the principal and the interest earned or charged in previous periods. It results in compounding costs over time and it can be computed by

$$A = P(1 + \frac{r}{n})^{nt} \tag{2.2}$$

where A is the final amount, P is the initial principal balance, r is the interest rate, n the number of times interest applied per time period and t the number of time periods elapsed.

Interest rates for loans depend on market conditions, borrower creditworthiness, and economic factors. Market rates, influenced by benchmarks like central bank rates, affect loan rates. Borrowers with strong credit, stable income, and low debt levels typically secure lower rates. Economic conditions, including inflation and economic growth, play a role, and lender policies and competition also impact rates. Finally, global and geopolitical events can influence interest rates, particularly in international markets.

Annual Percentage Rate (APR)

This is directly related to the interest rate and it is given by the nominal interest rate and any fees or additional costs associated with the loan together.

Loan-to-Value ratio (LTV)

For secured loans LTV is a ratio that compares the loan amount with the appraised value of the collateral. A higher LTV ratio reduces the lender's risk and may lead to more favorable loan terms for the borrower. The formula to compute it is the following [6]

$$\text{LTV}_S = \frac{\text{Loan Amount}}{\text{Appraised Value}} \times 100\% \tag{2.3}$$

For an unsecured loan, the appraised value is replaced by the borrower's annual income.

2.1.6 Analytic tools

Given all the technical knowledge required, we are able to analyze the mathematical formulas that are used to calculate the key metrics of a loan agreement. The most important are observed in this section [7].

1. *Calculation of monthly payment*

   Assuming each regular payment is scheduled on a monthly basis, the *installment*, i.e. the amount of money paid regularly to cover both the principal and the interest accrued on the outstanding balance is given by

   $$P = \frac{r \cdot PV \cdot (1+r)^n}{(1+r)^n - 1} \tag{2.4}$$

   where: P is the installment, r is the monthly interest rate ($r = \frac{\text{annual rate}}{12}$), PV is the principal, n is the number of installments.

2. *Calculation of total interest paid*

   $$I = P \cdot n - PV \tag{2.5}$$

   where I is the total interest paid and P, n and PV are as defined above

3. *Calculation of outstanding loan balance* at a given time

   $$B = PV(1+r)^n - \frac{P(1+r)^n - 1}{r} \tag{2.6}$$

   where B denotes the outstanding balance, P, r, n and PV are defined above.

2.2 Margin Loans

Let us now go into detail about margin loans, a type of bank loan designed for startups and small or medium-sized companies that need liquidity to grow their business. These are listed companies, which have nothing to give in return but their own shares. The loan agreement is stipulated between a banking institution and the borrowing company.

2.2.1 Background

The concept of margin lending has its roots in the late 19th century in the United States, where margin loans were initially utilized as a method of financing the development of railroads and other infrastructural projects. Investors pooled their own funds and borrowed capital from banks or brokers to funnel more substantial investments into railroads and other public companies. The purchased securities were used as collateral for these loans, a method now known as *buying on margin* [8].

2.2.2 Structure

In a margin loan arrangement, the bank extends to the borrower a term or revolving loan secured by publicly traded equity securities. For this purpose, a Special Purpose Vehicle (SPV) is created to hold and manage the shares of the borrower company. This arrangement allows shared collateral management, with the SPV holding *legal* ownership of shares and the borrower retaining *beneficial* ownership. It safeguards the lending institution's interests, reduces risk by limiting the lender's exposure

to SPV assets, and offers legal protection in case of borrower default or share value decline. The structure of a margin loan is designed to retain the proceeds generated by the underlying collateral. These income is set aside to cover any early repayments before it can be withdrawn by the borrower and used for other purposes.

Special Purpose Vehicles

SPVs are legal entities that are established for specific financial purposes. They are designed to conduct financial transactions, hold assets, or manage risks without directly involving the parent company's core business. SPVs have their own legal identity and limited liability, which protects the parent company from potential financial repercussions in case of default. These vehicles are often structured to be independent from the parent company's bankruptcy proceedings, reducing the risk of being affected by them. SPVs are customized to serve specific financial objectives, such as holding mortgage portfolios or issuing asset-backed bonds.[9]

2.2.3 Borrowing base

The funding base for a margin loan is determined by examining the ratio of outstanding loans to the value of collateral deposited in the SPV. The fact that collateral is listed implies that its prices are available and updated daily without reporting delays, following a *mark to market* approach. Since the borrower is a newly listed firm, its shares are subject to greater volatility than those of well-established businesses, and this implies a greater risk of incurring severe losses for the bank. Therefore, an *overcollateralization* technique is adopted, in which the LTV ratio, that is, the ratio of the value of the loan to the current value of the

collateral, is kept very low [10]. Should the value of the collateral fall and the LTV ratio exceed a threshold set in the loan agreement, the borrower is required to increase the number of shares pledged as collateral to maintain the security of the loan. This mechanism is known as margin call.

Margin calls

The previously mentioned threshold is named *Margin call LTV* and indicates the lowest Loan-to-Value the loan can reach. This can occur if, on a market opening day, the company's stock price makes a significant downward jump, such as losing 10%. In this case, the LTV established at the contract date must be restored, which means that the borrower needs to replenish the collateral with the required number of shares. As detailed in [8], if "the borrower does not respond to the margin call by submitting additional collateral or calling in the principal by the specified due date, an event of default will occur, and the lender may declare the loan immediately due and foreclose on the collateral".

2.2.4 The quantitative approach

In such financing instruments, the risk exposure is entirely due to the underlying equity. Thus, the primary risk factor shifts significantly from credit risk, as seen in traditional loans, to market risk. Therefore, a comprehensive market analysis becomes paramount for accurately assessing and managing the risk associated. This directly influence the pricing process, which involves determining the appropriate interest rates defining the cost of investment. This pricing must be grounded in the evaluation of potential loss risks that the bank may incur if the borrower no longer has sufficient assets to provide as collateral, potentially due to a significant shortfall event. To effectively manage this risk, it is imperative to estimate the final value of the company's shares. Therefore, employing stochastic processes to simulate the trajectory of these assets becomes essential. Stochastic processes, such as Lévy processes, are indispensable tools for generating realistic market scenarios that aid in estimating potential outcomes, including severe downturns in share values. This thesis work plays a key role in the risk evaluation process by focusing on several objectives. Specifically, it aims to model price trajectories using three Lévy processes, to simulate realistic market scenarios. Additionally, the thesis aims to calibrate the parameters of these models to real-market option prices, specifically using *One-Touch Knock-Out Daily Cliquet Options*, a unique type of derivatives tailored to value significant downward movements in the underlying asset. Lastly, this work will conduct a comparative analysis of the stochastic models under consideration. This analysis aims to provide insight into and assess the inherent valuation risk associated with each model when employed for pricing margin loans.

Chapter 3

Financial Models for Stock Prices

In this work, four distinct models have been examined to capture asset price dynamics. The following sections will delve into the fundamental theories behind each of these models.

3.1 The Black-Scholes Model

In the context of financial models for pricing derivatives, Fischer Black and Myron Scholes emerged as pioneers in shaping the instruments that are now widely used. In 1973, detailed in their seminal work [11], they derived the Black-Scholes formula, an analytical tool used to price European call and put options, based on a set of key assumptions that underlie its mathematical foundation. Foremost among these assumptions is the proposition that the underlying asset's price conforms to a Geometric Brownian Motion (GBM). This is a stochastic process that amalgamates

deterministic growth and stochastic fluctuations. The dynamics of this process are captured by the equation:

$$\frac{dS_t}{S_t} = \mu dt + \sigma dW(t), \tag{3.1}$$

where μ is the constant drift rate, σ is the constant volatility, $W(t)_{\{t\geq 0\}}$ denotes a standard Brownian motion, also referred to as the Wiener process. This model assumes that percentage changes in the stock price in a very short period of time are normally distributed [12], so that

$$\frac{dS}{S} \sim \mathcal{N}(\mu dt, \sigma^2 dt) \tag{3.2}$$

where $N(m, v)$ denotes a normal distribution with mean m and variance v. Consequently, defining a future time T, it holds

$$ln\frac{S(T)}{S(0)} \sim N\left[\left(\mu - \frac{\sigma^2}{2}\right)T, \sigma^2 T\right] \tag{3.3}$$

and

$$lnS(T) \sim N\left[lnS(0) + \left(\mu - \frac{\sigma^2}{2}\right)T, \sigma^2 T\right], \tag{3.4}$$

where $S(T)$ is the stock price at time T, $S(0)$ is the initial stock price, $lnS(T)$ is normally distributed with mean $lnS(0) + (\mu - \sigma^2/2)T$ and standard deviation $\sigma\sqrt{T}$ so that $S(T)$ follows a lognormal distribution. In a *risk-neutral* world, the drift μ has to be replaced with r, i.e. the risk-free interest rate, compounded continuously. It means that the expected stock price grows at the same rate as the risk-free

investment. For any time t, the dynamics of the underlying's price becomes

$$S(t) = S(0)exp\left\{\left(r - \frac{\sigma^2}{2}\right)t + \sigma\sqrt{t}Z\right\}. \tag{3.5}$$

with Z denoting a standard normal variable.

3.2 The Lévy Processes

The Lévy processes, named after the French mathematician Paul Lévy, are widely used in mathematical finance as they describe financial markets in a more accurate way than models based on Brownian motion. A Levy process has independent and stationary increments and can be viewed as the continuous-time analog of a random walk. The Brownian Motion and the Poisson process are processes belonging to this class.

Definition 1 (Lèvy Process) *A* cadlag *stochastic process* $(X_t)_{t\geq 0}$ *defined on a probability space* $(\Omega, \mathcal{F}, \mathbb{P})$ *with values in* $\mathbb{R}^d$ *such that* $X_0 = 0$ *is said to be a* Lévy Process *if it satisfies the following properties:*

1. *Independent increments: for* $0 \leq s \leq t$, $X_t - X_s$ *is independent of* $\{X_u : u \leq s\}$
2. *Stationary increments: for* $0 \leq s \leq t$, $X_t - X_s$ *is equal in distribution to* X_{t-s}
3. *Stochastic continuity:* $\forall \varepsilon > 0$, $\lim_{h\to 0} P(|X_{t+h} - X_t| \geq \varepsilon) = 0$

The cadlag property means that the paths of X are $\mathbb{P}$-almost surely right continuous with left limits. The third condition serves to exclude processes that have jumps at fixed (non-random) times, guaranteeing that the probability of observing a jump at a given time is zero and emphasizing the random nature of discontinuities in the

process.

A Lévy Process is infinitely divisible, which implies that the process can be decomposed into an arbitrary number of independent and identically distributed (i.i.d.) increments over any time interval. In other words, the process can be viewed as the sum of an arbitrary number of independent and identically distributed random variables.

Definition 2 (Infinite divisibility) *A probability distribution F on $\mathbb{R}^d$ is said to be infinitely divisible if for any integer $n \geq 2$, there exist n i.i.d. random variables $Y_1, \ldots, Y_n$ such that $Y_1 + \ldots + Y_n$ has distribution F.*

The infinitely divisible distributions can be fully characterized by the characteristic exponent Ψ through an expression known as the Lévy–Khintchine formula. Moreover, it is known that any Lévy process $(X_t)_{t\geq 0}$ has the following property

$$\mathbf{E}\left(e^{i\theta X_t}\right) = e^{-t\Psi(\theta)}, \tag{3.6}$$

where $\Psi(\theta)$ is the characteristic exponent of X_t. Thus, the so called Lévy-Khintchine representation allows to derive the characteristic function for Lévy Processes, that is

$$\mathbf{E}\left(e^{i\theta X_t}\right) = e^{-t\Psi(\theta)}, \quad \theta \in \mathbb{R}^d \tag{3.7}$$

where

$$\Psi(\theta) = ia\theta + \frac{1}{2}\sigma^2\theta + \int_{\mathbb{R}}(1 - e^{i\theta x} + i\theta x \mathbf{1}_{|x|<1})\Pi(dx) \tag{3.8}$$

and $a \in \mathbb{R}$, $\sigma^2 \geq 0$ and Π is a measure concentrated on $\mathbb{R} \setminus \{0\}$, such that

$$\int_{\mathbb{R}} \inf\{1, x^2\}\Pi(dx) = \int_{\mathbb{R}} (1 \wedge x^2)\Pi(dx) < \infty. \tag{3.9}$$

The triplet $[a, \sigma^2, \Pi]$ is called the Lévy triplet where $a \in \mathbb{R}$ is the *drift term*, $\sigma^2 \in \mathbb{R}$ is the *diffusion coefficient* and the measure $\Pi(dx)$ is the *Lévy measure* of x. In particular, if $\Pi(dx)$ is of the form $\nu(x)dx$, i.e. it is a density that, in comparison to a probability density, must not necessarily be integrable and must have zero mass in the origin. Heuristically, the function $\nu(x)$ determines how frequently jumps of size X occur in the Lévy process. By analyzing the characteristic triplet $[a, \sigma^2, \Pi]$ of a Lévy process, we can gain insights into the properties of its typical sample paths.

Defining the total variation of a function $f : [a, b] \to \mathbb{R}^d$ as

$$TV(f) = sup \sum_{i=1}^{n} |f(t_i) - f(t_{i-1})|, \tag{3.10}$$

f is of finite variation if $TV(f)$ is finite for each partition of each finite sets and it can be written as the difference of two increasing functions.
If a Lévy process has trajectories which are almost surely functions of finite variation, it belongs to the class of finite variation Lévy Processes.

Definition 3 (Finite variation Lévy processes) *A Lévy process is of finite variation if and only if its characteristic triplet* $[a, \sigma^2, \Pi]$ *satisfies:*

$$\sigma = 0 \quad and \quad \int_{|x| \leq 1} |x|\Pi(dx) < \infty. \tag{3.11}$$

Another important class of the Lévy processes is the one of subordinators. A subordinator is a non-decreasing Lévy process with almost surely positive increments. It is used to apply time changes to another Lévy process, effectively acting as a clock that determines the time at which jumps occur. The following statement provides equivalent definitions of a subordinator Lévy process.

Proposition 1 *Let $(X_t)_{t\geq 0}$ be a Lévy process on $\mathbb{R}$. The following conditions are equivalent:*

1. *$X_t \geq 0$ almost surely for some $t > 0$.*
2. *$X_t \geq 0$ almost surely for every $t > 0$.*
3. *Sample paths of (X_t) are almost surely non-decreasing: $t \geq s \Rightarrow X_t \geq X_s$ almost surely.*
4. *The characteristic triplet of (X_t) satisfies $\sigma = 0$, $\Pi((-\infty, 0]) = 0$, $\int_{-\infty}^{0}(x \wedge 1)\nu(dx) < \infty$, and $a' \geq 0$. In other words, (X_t) has no diffusion component, only positive jumps of finite variation, and positive drift.*

The simplest and well-known Lévy processes include the Wiener Process, the Poisson process, but also any linear combination of indipendent Lévy processes, such as the compound Poisson process. Being an extensive class of stochastic processes, Levy processes encompass several subclasses, each distinguished by distinct characteristics and properties. Among these subclasses two prominent processes known as Jump-diffusion models will be analyzed in this work: the *Merton model* [13] and the *Kou model* [14]. Another significant subclass arises

from Brownian subordination, resulting in processes such as the *Variance Gamma process*, which will be addressed below.

3.2.1 Jump Diffusion Processes

Jump-diffusion processes are stochastic models that combine a continuous diffusion process with a jump process. They are widely used in finance thanks to their ability to capture both the continuous movements of prices and sudden jumps due to unexpected events. The general form of a jump-diffusion process is expressed by the following equation:

$$X(t) = \gamma t + \sigma W(t) + \sum_{i=1}^{N(t)} Y_i \tag{3.12}$$

where $W(t)_{\{t\geq 0\}}$, is the Brownian motion that stands for the diffusion part, $N(t)_{\{t\geq 0\}}$ is the Poisson process that counts the number of jumps of X, and Y_i denotes the size of each jump. What can vary across different models is the distribution of jump sizes, $\nu_0(x)$.

Merton Model

The first jump diffusion process was proposed by Robert C. Merton in 1976, [13] as an extension of the Black-Scholes model. Thus, the model assumes that the stock price follows a geometric Brownian motion with a constant volatility, while the jump component follows a log-normal distribution, which means that the logarithm of the jump size is normally distributed with mean m and standard deviation δ. This allows for the incorporation of both positive and negative jumps in the stock

price.

The dynamics of the asset price S_t in the Merton Model can be described as follows:

$$\frac{dS_t}{S_t} = \mu dt + \sigma dW(t) + d\left(\sum_{i=1}^{N(t)} (Y_i - 1)\right), \tag{3.13}$$

where μ is the constant drift, σ is the constant volatility, $W(t)_{\{t\geq 0\}}$ is a standard Brownian motion, λ is the jump intensity, $N(t)_{\{t\geq 0\}}$ is a Poisson process with rate λ, and $Y_i\{i \geq 1\} - 1$ represents the size of the i-th jump. The Y_i is lognormally distributed, so that $logY_i \sim N(m, \delta^2)$, then for any fixed n,

$$\prod_{i=1}^{N(t)} Y_i \sim log\mathcal{N}(mn, \delta^2 n). \tag{3.14}$$

In the context of the Merton Jump Diffusion Model, the return process $X(t) = \log\left(\frac{S(t)}{S(0)}\right)$ is defined by the equation:

$$X(t) = \mu t + \sigma W(t) + \sum_{i=1}^{N(t)} Y_i, \qquad X(0) = 0. \tag{3.15}$$

After solving the differential equation 3.13, the stochastic dynamics under the probability measure $\mathbb{P}$ of the asset price follows

$$\mathbb{P}: \qquad S(t) = S(0) \exp\{\mu t + \sigma W(t)\} \prod_{i=1}^{N(t)} Y_i. \tag{3.16}$$

To guarantee completeness of such a model, a *risk-neutral* measure has to be found, i.e. a measure $\mathbb{Q} \sim \mathbb{P}$ such that the discounted price $\mathsf{E}[e^{-rt}S(t)] = S(0)$ is a martingale. This lead to a change in the drift of the Wiener process but leaves

unchanged the other ingredients. Thus, under the risk neutral measure $\mathbb{Q}^M$:

$$\mathbb{Q}^M : \qquad S(t) = S(0) \exp\left\{\mu^M t + \sigma W^M(t)\right\} \prod_{i=1}^{N(t)} Y_i, \tag{3.17}$$

where

$$\mu^M = r - \frac{\sigma^2}{2} - \lambda \mathsf{E}\left[e^{Y_i} - 1\right] = r - \frac{\sigma^2}{2} - \lambda \left[\exp(m + \frac{\delta^2}{2}) - 1\right]. \tag{3.18}$$

Kou Model

The Kou Model, also known as the Double Exponential Jump Diffusion Model, was developed by S.G. Kou in 2002 [14]. This model aimed to address two empirical phenomena observed in stock market data: the asymmetric leptokurtic behavior of returns and the volatility smile. It has been demonstrated that a normal distribution for logarithmic returns of asset prices oversimplifies the real-world behavior and fails to capture crucial features such as fat tails, skewness, volatility clustering, and non-constant interest rates. The model developed by Kou is an exponential Lévy with finite jump intensity where the size of the jumps follow a two-sided exponential distribution.

The dynamics of the asset price S_t is as follows

$$\frac{dS_t}{S_t} = \mu dt + \sigma dW(t) + d\left(\sum_{i=1}^{N(t)} (V_i - 1)\right), \tag{3.19}$$

where $W(t)_{\{t\geq 0\}}$ is a standard Brownian motion, $N(t)_{\{t\geq 0\}}$ is a Poisson process with rate λ, $V_{i\{i\geq 1\}}$ is a sequence of independent and identically distributed nonnegative random variables such that $Y = ln(V)$ has an asymmetric double exponential

distribution with density

$$f_Y(y) = p\eta_1 e^{-\eta_1 y} \cdot \mathbf{1}_{\{y \geq 0\}} + q\eta_2 e^{\eta_2 y} \cdot \mathbf{1}_{\{y<0\}}, \tag{3.20}$$

$$\eta_1 > 0, \eta_2 > 0.$$

Here, $p \geq 0$ and $q \geq 0$, with $p+q = 1$ denote the probabilities associated of upward and downward jumps, respectively.

Each random variable Y can be decomposed by:

$$ln(V) = Y = \begin{cases} \xi^+ & \text{with probability } p \\ -\xi^- & \text{with probability } q \end{cases} \tag{3.21}$$

where $\xi^+ \sim \text{Exp}(\eta_1)$ and $\xi^- \sim \text{Exp}(\eta_2)$. Consequently, the return process $X(t) = log(S(t)/S(0))$ is given by

$$X(t) = \mu t + \sigma W(t) + \sum_{i=1}^{N(t)} Y_i, \qquad X(0) = 0. \tag{3.22}$$

The stochastic dynamics of the asset price under the probability measure $\mathbb{P}$ is obtained by solving the differential equation 3.19, such that:

$$\mathbb{P}: \qquad S(t) = S(0) \exp\{\mu t + \sigma W(t)\} \prod_{i=1}^{N(t)} V_i. \tag{3.23}$$

Since $\mathsf{E}[Y] = p\left(\frac{1}{\eta_1}\right) - q\left(\frac{1}{\eta_2}\right)$, and $Var[Y] = pq\left(\frac{1}{\eta_1} + \frac{1}{\eta_2}\right)^2 + \left(\frac{p}{\eta_1^2} + \frac{p}{\eta_2^2}\right)$, the expected

value of $V_{i\{i\geq 1\}}$ can be computed:

$$\mathsf{E}\left[V\right] = \frac{q\eta_2}{(\eta_2+1)} + \frac{p\eta_1}{(\eta_1-1)}, \quad \eta_1 > 1, \eta_2 > 0, \tag{3.24}$$

where the requirement $\eta_1 > 1$ ensures that $\mathsf{E}\left[V\right] < \infty$, which means that the average upward jumps cannot exceed 100%.

To find the risk-neutral adjustment to the dynamics, recall that it must be guaranteed $\mathsf{E}[e^{-rt}S(t)] = S(0)$ to be a martingale. Introducing $\mathbb{Q}^K$ as the proper *risk-neutral* measure, the stochastic dynamics of the asset price becomes

$$\mathbb{Q}^K: \qquad S(t) = S(0)\exp\left\{\mu^K t + \sigma W^K(t)\right\}\prod_{i=1}^{N(t)} V_i, \tag{3.25}$$

where

$$\mu^K = r - \frac{\sigma^2}{2} - \lambda(\mathsf{E}\left[V\right]-1) = r - \frac{\sigma^2}{2} - \lambda\left[\frac{q\eta_2}{(\eta_2+1)} + \frac{p\eta_1}{(\eta_1-1)} - 1\right]. \tag{3.26}$$

3.2.2 The Variance Gamma Process

The Variance Gamma process was initially introduced by Madan and Seneta, who illustrated a first symmetric version in 1990 [15]. After that, the general, non-symmetric version of the model was described in a 1998 paper by Madan, Carr, and Chang [16]. It was devised as an extension of the Black and Scholes model to better model the distribution of asset price returns, having observed from real-world data that the latter is often asymmetric with heavy tails, skewness, and excess kurtosis. The Variance Gamma is a subordinated Levy processes, as it is obtained by applying a random time change to a Brownian motion with constant drift and volatility. The subordinator, which is always non-negative and non-decreasing, is the stochastic gamma process, and it represents the waiting times between jumps of the VG processes.

According to this, the first representation of the VG can be defined. Let

$$b(t; \theta, \sigma) = \theta t + \sigma W(t) \tag{3.27}$$

be a Brownian motion with constant drift rate θ and volatility σ where $W(t)$ is a Wiener process. Let $\gamma(t; \mu, \nu)$ be a gamma process with mean μ and variance ν, based on the gamma function $\Gamma(\cdot)$ and given by independent gamma increments over non-overlapping intervals. The Variance Gamma process $X(t; \sigma, \nu, \theta)$ is thus obtained as a Brownian motion with stochastic time, given by a gamma process with unit mean rate $\gamma(t; 1, \nu)$:

$$(t; \sigma, \nu, \theta) = b(\gamma(t; 1, \nu); \theta, \sigma), \tag{3.28}$$

where $\gamma(t;1,\nu)$ is a gamma with unit mean rate and variance rate ν, while σ and θ, are parameters inherited from the Brownian motion. The parameter σ controls the volatility, ν and θ respectively control the skewness and the kurtosis of the distribution. Denoting the final process as $X(t)$ and the unit mean rate gamma as $\gamma(t)$, the Variance Gamma process is defined as

$$X(t) = \theta\gamma(t) + \sigma W_{\gamma(t)}(t). \tag{3.29}$$

The characteristic function $\phi_{X(t)}(u) = \mathsf{E}\left[\exp\left(iuX(t)\right)\right]$ can be obtained by conditioning on the gamma time change and then employing the characteristic function of the gamma process itself [17]. Given the chf of the gamma distribution:

$$\phi_{\gamma(t)}(u) = \mathsf{E}\left[\exp\left(iu\gamma(t;1,\nu)\right)\right] = (1 - iu\nu)^{-\frac{t}{\nu}} \tag{3.30}$$

and the chf of the normal distribution

$$\phi_{b(t)}(u) = \mathsf{E}\left[\exp\left(iub(t;\theta,\sigma)\right)\right] = \exp\left(iu\theta - \frac{\sigma^2 u^2}{2}\right), \tag{3.31}$$

it follows:

$$\begin{aligned}\phi_{X(t)}(u) &= \mathsf{E}\left[\mathsf{E}\left[\exp(iuX(t) \,|\, \gamma(t;1,\nu)\right]\right] = \\ &= \mathsf{E}\left[\mathsf{E}\left[\exp(iu\, b(\gamma(t;1,\nu);\theta,\sigma) \,|\, \gamma(t;1,\nu)\right]\right]\end{aligned}$$

where $b(\gamma(t;1,\nu);\theta,\sigma)$ is normally distributed with mean $\theta\gamma(t;1,\nu)$ and variance $\sigma^2\gamma(t;1,\nu)$.

Thus,

$$\mathsf{E}\left[\exp(iu\, b(\gamma(t;1,\nu);\theta,\sigma)\,|\,\gamma(t;1,\nu)\right] = \exp\left(iu\theta\,\gamma(t;1,\nu) - \frac{\sigma^2u^2}{2}\,\gamma(t;1,\nu)\right) \tag{3.32}$$

and therefore

$$\begin{aligned}\phi_{X(t)}(u) &= \mathsf{E}\left[\exp(iu\theta - \left(\frac{\sigma^2u^2}{2}\right)\gamma(t;1,\nu)\right] = \\ &= \mathsf{E}\left[\exp(iu\left(\theta - \frac{\sigma^2u^2}{2iu}\right)\gamma(t;1,\nu)\right] = \\ &= \left(1 - iu\left(\theta - \frac{\sigma^2u^2}{2iu}\right)\nu\right)^{-\frac{t}{\nu}} = \\ &= \left(1 - i\theta\nu u - \frac{\sigma^2\nu}{2}u^2\right)^{-\frac{t}{\nu}}.\end{aligned} \tag{3.33}$$

Another common representation of the VG process is derived considering that it is Lévy a process of finite variation, thus it can be seen as the difference of two independent increasing gamma processes:

$$(t;\sigma,\nu,\theta) = \gamma(t;\mu_+,\nu_+) - \gamma(t;\mu_-,\nu_-), \tag{3.34}$$

whose parameters are given by

$$\mu_\pm = \frac{1}{2}\sqrt{\theta^2 + \frac{2\sigma^2}{\nu}} \pm \frac{\theta}{2}, \tag{3.35}$$

$$\nu_\pm = \mu_\pm^2\nu. \tag{3.36}$$

Being a Lévy process, it can be uniquely characterized by the Lévy-Khintchine

formula, which gives the characteristic exponent

$$\Psi(\theta) = -\frac{1}{\nu} ln \left\{ 1 - i\theta\nu u + \frac{\sigma^2 \nu u^2}{2} \right\} \tag{3.37}$$

and the Lévy measure

$$\Pi(x) = \frac{1}{\nu|x|} \exp\left(\frac{\theta}{\sigma^2} x - \frac{1}{\sigma} \sqrt{\frac{2}{\nu} + \frac{\theta^2}{\sigma^2}|x|} \right). \tag{3.38}$$

Being the drift rate and the diffusion component both zero the VG process is a pure jump process with infinite activity and finite variation. Such processes, according to Carr, Geman, Madan, and Yor (2002) [18] are more useful for fitting market prices since they can capture both small and large fluctuations. A process with infinite activity has a high frequency of small jumps, while a process with finite variation has a limited number of large jumps. In conclusion, under the probability measure $\mathbb{P}$, the stochastic asset price dynamics can be expressed as

$$S(t) = S(0) \exp\left[\mu t + X(t; \sigma, \nu, \theta)\right], \tag{3.39}$$

where $X(t; \sigma, \nu, \theta)$ is the Variance Gamma process and μ is the constant drift. A *risk-neutral* probability measure $\mathbb{Q}^{VG}$ can be found such that the discounted asset price is a martingale, i.e. it satisfies $\mathsf{E}[e^{-rt} S(t)] = S(0)$. It follows [19]

$$\mathbb{Q}^{VG}: \qquad S(t) = S(0) \exp\left[\mu^{VG} t + X^{VG}(t; \sigma, \nu, \theta)\right], \tag{3.40}$$

where

$$\mu^{VG} = r + \omega = r + \frac{1}{\nu} \ln(1 - \theta\nu - \frac{\sigma^2 \nu}{2}). \tag{3.41}$$

Chapter 4

Monte-Carlo simulation of price dynamics

Monte Carlo simulations are a powerful computational technique widely used in finance to model and analyze the behavior of financial assets over time. In the realm of financial price paths, Monte Carlo simulations involve generating a large number of possible future scenarios, simulating a multitude of paths that represent potential outcomes. By doing so, analysts and investors are capable of exploring diverse market conditions and assessing associated risks and opportunities. This technique aids in pricing derivatives, estimating probabilities of different market scenarios.

For each Monte Carlo simulation, the number of paths N, the time window T, and the number of time steps *days* will be defined. The simulation process follows these steps:

1. *Initialization* Set the initial conditions, such as the starting asset price and

the parameters of each model.

2. *Time Discretization:* Divide the time window T into discrete time steps *days*.

3. *Simulation Loop:* For each simulation path, iterate through the time steps, generating random numbers to represent the stochastic components, applying the model equations, and calculating the asset price at the next time step.

4. *Aggregation:* Collect the results from all simulation paths in a matrix of dimensions $(days, N)$, with each path occupying one column.

The following sections present the algorithm for simulating each of the four models, which are the subject of this work. The simulation code for all of the models can be found in Appendix A

4.1 The Black-Scholes model

Given the equation 3.5 of the dynamics of the asset prices, the BS models requires the simulation of the GBM.

Algorithm 1 Simulation of Geometric Brownian Motion

Input: GBM parameter σ; time spacing $\Delta t_1, ..., \Delta t_N$ s.t. $\sum_{i=1}^{N} \Delta t_i = T$
Initialization: Set $S_0 = S_0^*$, $i = 1$
while $i \leq N$ **do**
  Generate $Z_i \sim \mathcal{N}(0,1)$ from a standard normal distribution
  Compute $S_{t_i} = S_{t_{i-1}} \exp\left((r - \frac{1}{2}\sigma^2)\Delta t_i + \sigma\sqrt{\Delta t_i} Z_i\right)$
  $i \leftarrow i + 1$
end while
return S

With the initial stock price $S0 = 100$ and the parameters $r = 0.03$, $\sigma = 0.175$, the

Figure 4.1 shows five simulated paths under the Black-Scholes model over $T = 1$ year, i.e. 252 financial days.

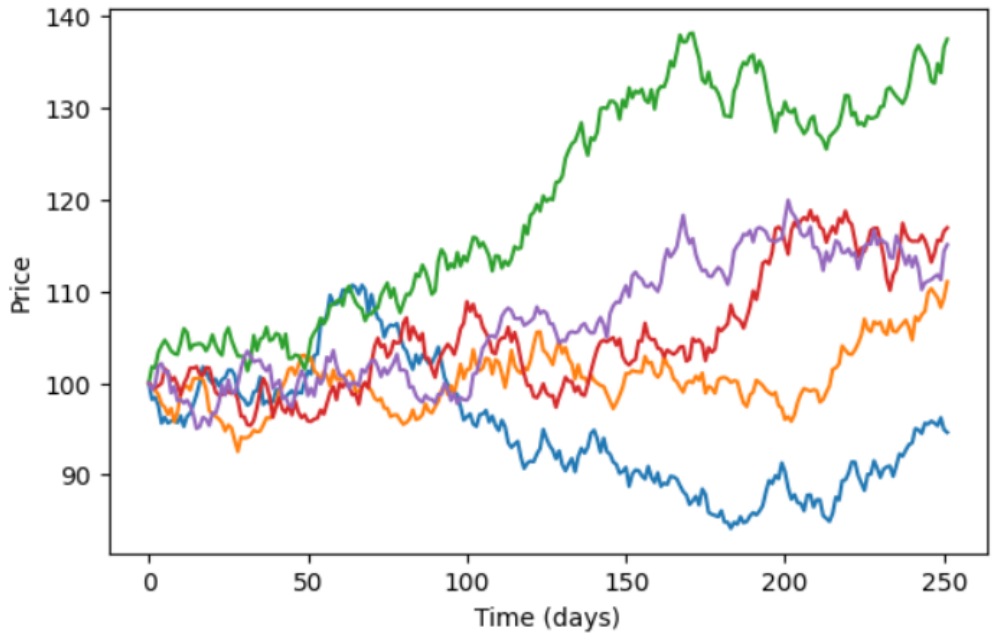


Figure 4.1: Five simulated paths of the BS model

4.2 The Merton Jump Diffusion model

The Merton jump diffusion dynamics of price, as detailed in 3.13 can be simulated as follows.

Given the initial stock price $S0 = 100$ and the parameters $r = 0.03$, $\sigma = 0.175$, $\lambda = 0.5$, $m = 0.05$, $\delta = 0.15$ the Figure 4.2 shows five simulated paths under the Merton Jump Diffusion model over $T = 1$ year, i.e. 252 financial days.

Algorithm 2 Simulation of Merton Jump Diffusion model

Input: MJD parameters $\sigma, \lambda, m, \delta$; time spacing $\Delta t_1, ..., \Delta t_N$ s.t. $\sum_{i=1}^{N} \Delta t_i = T$
Initialization: Set $SS_0 = S_0^*$, $i = 1$, $\mu_{RF} = exp(m + \delta^2/2) - 1$
while $i \leq N$ **do**
  Generate $Z_i \sim \mathcal{N}(0,1)$ from a standard normal distribution
  Generate $N_i \sim Pois(\lambda \cdot \Delta t_i)$ from a Poisson distribution
  Generate $J_i \sim \mathcal{N}(m, \delta^2)$ from a normal distribution
  Compute $S_{t_i} = S_{t_{i-1}} \exp\left\{(r - \frac{1}{2}\sigma^2 - \mu_{RF})\Delta t_i + \sigma\sqrt{\Delta t_i} Z_i + J_i \cdot N_i\right\}$
  $i \leftarrow i + 1$
end while
return S

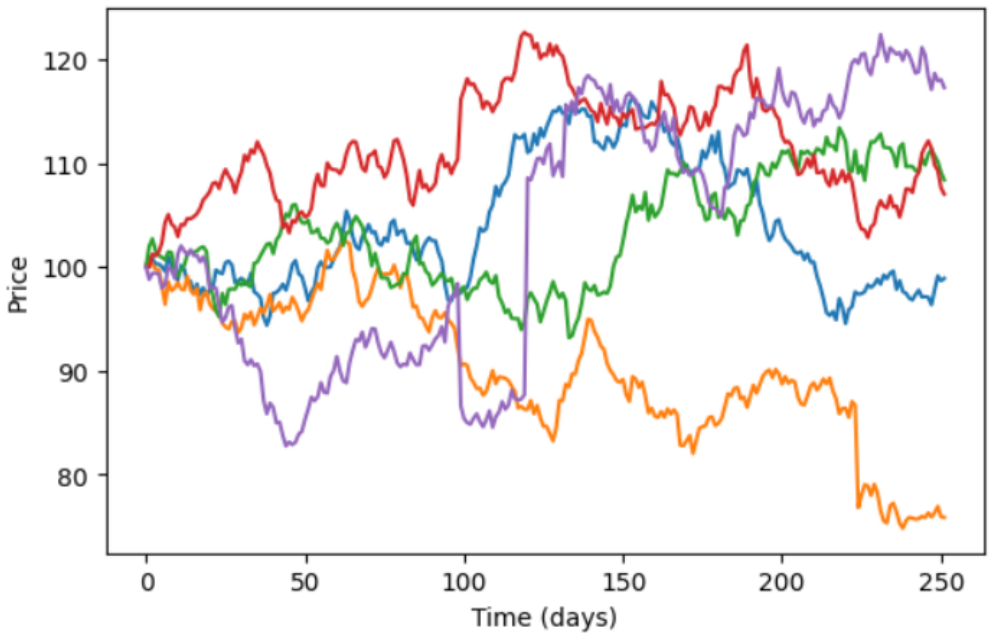


Figure 4.2: Five simulated paths of the MJD model

4.3 The Kou Jump Diffusion model

Algorithm 3 demonstrates the procedure for simulating the Double Exponential Jump Diffusion process, for which the corresponding equation is given by 3.19.

Algorithm 3 Simulation of Kou Jump Diffusion Model

Input: KJD parameters $\sigma, \lambda, p, \eta_1, \eta_2$; time spacing $\Delta t_1, \ldots, \Delta t_N$ s.t. $\sum_{i=1}^{N} \Delta t_i = T$
Initialization: Set $S_0 = S_0^*$, $i = 1$, $\zeta = \frac{(p-1)\cdot\eta_2}{\eta_2+1} + \frac{p\cdot\eta_1}{\eta_1-1} - 1$
while $i \leq N$ **do**
  Generate $N_i \sim \text{Poisson}(\lambda \cdot \Delta t_i)$
  Generate $Z_i \sim \mathcal{N}(0,1)$ from a standard normal distribution
  Compute jump size $J = \begin{cases} 1/\eta_1 & \text{with probability } p \\ -1/\eta_2 & \text{with probability } 1-p \end{cases}$
  Compute $S_{t_i} = S_{t_{i-1}} \exp\left((r - \frac{1}{2}\sigma^2 - \lambda\zeta)\Delta t_i + \sigma\sqrt{\Delta t_i} Z_i + J_i \cdot N_i\right)$
  $i \leftarrow i + 1$
end while
return S

Starting from a stock price $S0 = 100$, Figure 4.3 illustrates paths of the Kou Jump Diffusion model with parameters $r = 0.03$, $\sigma = 0.153$, $\lambda = 1$, $p = 0.6$, $\eta_1 = 8$. $\eta_2 = 5$.

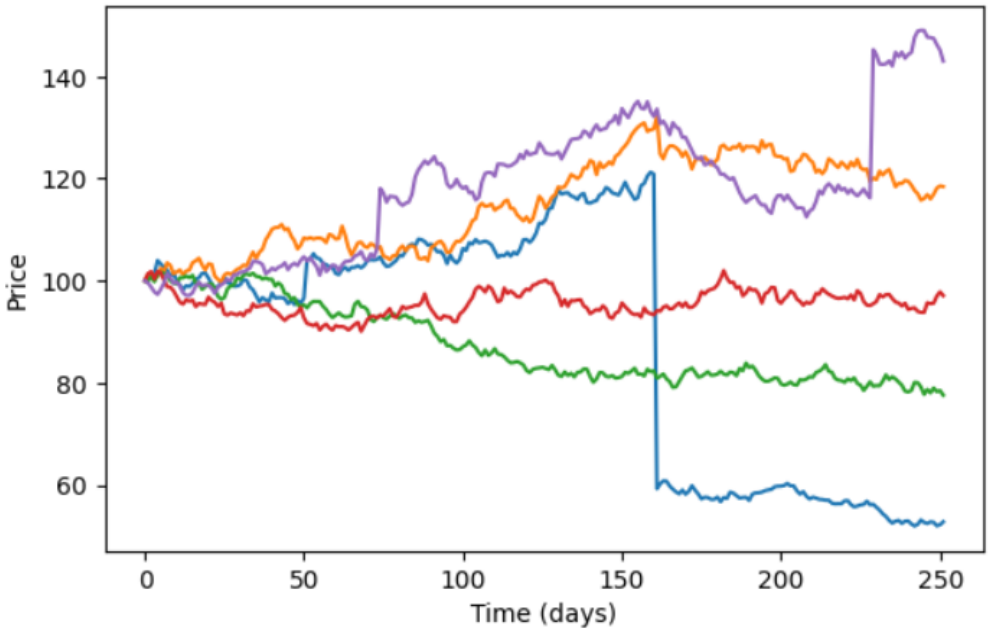


Figure 4.3: Five simulated paths of the KJD model

4.4 The Variance Gamma

As highlighted by Fu in 2007 [19], there exist three primary approaches for simulating VG processes. Among these, two are considered "exact," leading to the accurate derivation of the distribution, while the third involves approximating the VG process through a compound Poisson process. The initial two methods are elaborated upon in Section 3.2.2, while the third approach falls beyond the scope of this study.
The first method, known as the Time-changed (subordinated) Brownian Motion, directly incorporates the three parameters inherent to the VG process definition: the volatility of BM denoted as σ, the constant drift rate of the BM indicated as θ, and the variance rate of the gamma process denoted as ν. Simulation of this approach can be executed utilizing the subsequent algorithm.

Algorithm 4 Simulation of VG as Time changed Brownian Motion

Input: VG parameters σ, θ, ν; time spacing $\Delta t_1, ..., \Delta t_N$ s.t. $\sum_{i=1}^{N} \Delta t_i = T$
Initialization: Set $S_0 = S_0^*$, $i = 1$, $\omega = \frac{1}{\nu} \cdot \left(1 - \theta\nu - \frac{\nu\sigma^2}{2}\right)$
while $i \leq N$ **do**
  Generate $\Delta G_i \sim \Gamma(\Delta t_i/\nu, \nu)$ from a Gamma distribution
  Generate $\Delta Z_i \sim \mathcal{N}(0, 1)$ from a standard normal distribution
  Compute $S_{t_i} = S_{t_{i-1}} \exp\left((r + \omega)\Delta t_i + \theta\Delta G_i + \sigma\sqrt{\Delta G_i} Z_i\right)$
  $i \leftarrow i + 1$
end while
return S

After setting $S0 = 100$, $T = 1$, $days = 252$, and then $\sigma = 0.2$, $theta = -0.12$, $\nu = 0.05$ the following sample can be observed.

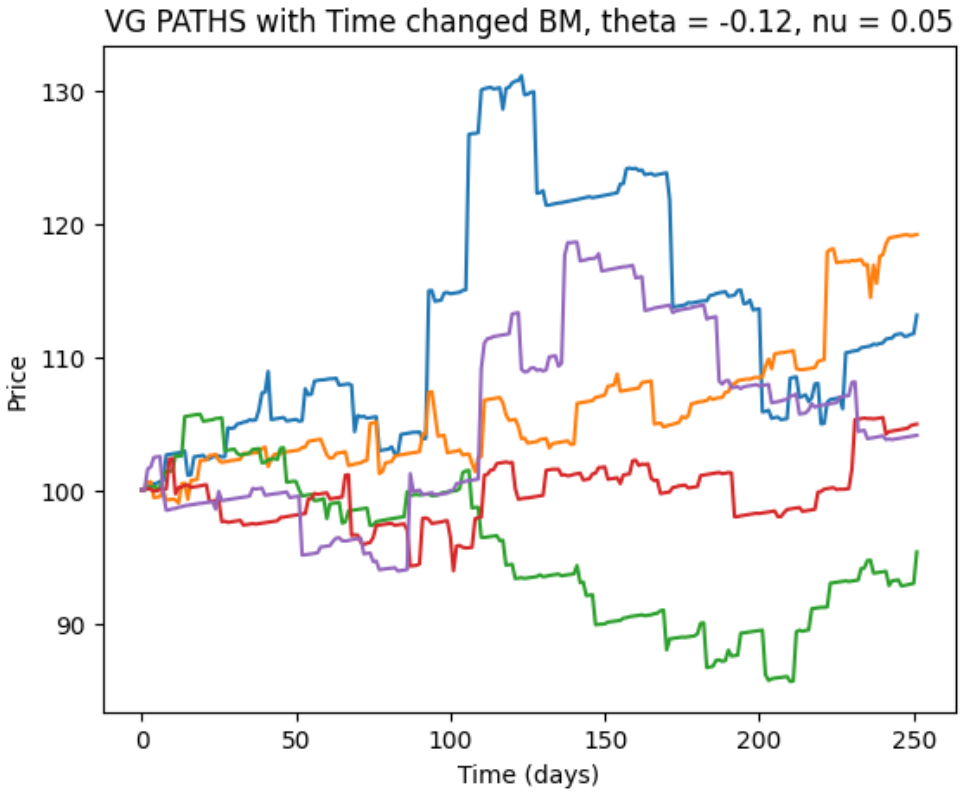


Figure 4.4: Five simulated paths of the VG model with method 1

The second approach involves representing the VG process as a difference between two gamma processes. This method utilizes equations 3.35 and 3.36 to determine the parameters of the two gamma processes. Specifically, one of the gamma processes captures the "positive" shifts in the asset price, corresponding to upward jumps. As such, it carries the mean rate μ_+ and variance rate ν_+. Conversely, the second gamma process characterizes the "negative" price shifts, representing downward jumps in the asset price, with mean rate parameter μ_- and variance rate parameter ν_-.

Algorithm 5 Simulation of VG as Difference of Gammas

Input: VG parameters σ, θ, ν; time spacing $\Delta t_1, ..., \Delta t_N$ s.t. $\sum_{i=1}^{N} \Delta t_i = T$
Initialization: Set $S_0 = S_0^*$, $i = 1$, μ_+, μ_-, $\omega = \frac{1}{\nu}\left(1 - \theta\nu - \frac{\nu\sigma^2}{2}\right)$
while $i \leq N$ **do**
Generate $\Delta\gamma_i^- \sim \Gamma(\Delta t_i/\nu, \nu\mu_-)$, $\Delta\gamma_i^+ \sim \Gamma(\Delta t_i/\nu, \nu\mu_+)$ from a Gamma distribution
Compute $S_{t_i} = S_{t_{i-1}} \exp\left((r+\omega)\Delta t_i + \Delta\gamma_i^+ - \Delta\gamma_i^-\right)$
$i \leftarrow i + 1$
end while
return S

By simulating this process with an initial value of $S_0 = 100$, and utilizing parameters $\sigma = 0.2$, $\theta = -0.12$, and $\nu = 0.05$, we can derive the necessary parameters: $\mu_+ = 0.575$, $\nu_+ = 0.017$ as well as $\mu_- = 0.695$, $\nu_- = 0.024$. The outcomes are depicted in Figure 4.5.

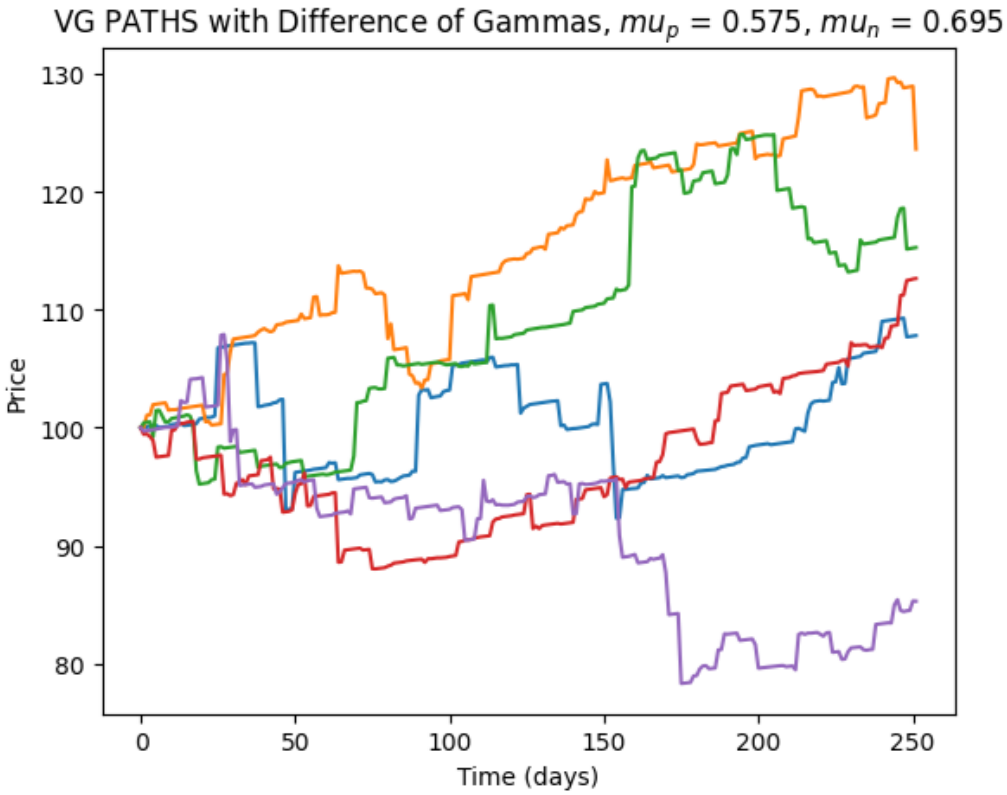


Figure 4.5: Five simulated paths of the VG model with method 2

Chapter 5

Option Pricing

Option pricing is a fundamental concept in the world of finance that plays a pivotal role in managing risk, making investment decisions, and ensuring the efficient functioning of financial markets. Options are financial derivatives that provide investors with the *right*, but not the *obligation*, to buy or sell an underlying asset at a predetermined price within a specified time frame. They are powerful tools that allow individuals and institutions to hedge against adverse price movements, speculate on market trends, and enhance portfolio diversification. Options are actively traded in both organized exchanges and the Over-The-Counter (OTC) market. Organized exchanges, such as the Chicago Board Options Exchange (CBOE) in the United States or Euronext in Europe, provide standardized contracts with readily available bid and ask prices.

In this chapter, we will explore the intricacies of option pricing, with a particular emphasis on European Plain Vanilla options and Exotic options, specifically *One Touch Knock Out Daily Cliquets* (OTKO). Our aim is to provide a comprehensive

understanding of these financial instruments by combining theoretical insights with numerical solutions. We will delve into the structures, payoffs, and the methodologies used to determine their fair values.

To determine the fair value of an option, we begin with the stochastic model that models the evolution of the stock price. Each stochastic model has its own parameters, unknown a priori. So, to simulate the N possible realizations of such stochastic processes, on which to conduct the required analysis, we go through the process of parameter calibration. It consists in finding accurate estimates of these parameters by comparing, for a given set of options, the market and theoretical prices of these options, i.e., calculating the difference between them. So one iteratively changes the parameter estimates, obtaining those that minimize this difference.

5.1 European plain vanilla options

Plain vanilla options represent the bedrock of option trading. They possess a straightforward structure, allowing their holders to buy (call) or sell (put) an underlying asset at a predetermined price on a specified expiration date. The *European* feature means they can only be exercised at the option's expiration date, in opposition with *American* options that allows their exercise at any time before or on the expiration date. In this section, we explore the core concepts and dynamics of these options and we delve into the numerical solutions for European Plain Vanilla options, discussing closed-form solutions and the utilization of Monte Carlo simulations to estimate their fair prices.

Call options

A call option is a financial contract that gives the holder the right, but not the obligation, to buy an underlying asset at a predetermined price, called the *exercise price* or *strike price*, within a specified time frame. The last day of exercise is known as *expiration date* or *maturity date*. As detailed in [12], there are two sides to every option contract: a long position, the one of the investor who has bought the option, and a short position, the one of the investor who has sold or written the option. If T is the maturity date, and we denote S_T as the underlying final price and K as the strike price, the payoff formula from a long position in a call option at expiration is straightforward:

$$\max(0, S_T - K), \tag{5.1}$$

while the payoff from a short position in a call option is

$$-\max(0, S_T - K) = \min(0, K - S_T). \tag{5.2}$$

In this formula, if the asset price is higher than the strike price, the option holder profits and the writer looses; otherwise, the option expires worthless.

Put options

A put option is a financial contract that provides the holder with the right, but not the obligation, to sell an underlying asset at the *exercise price*, within the *expiration date*. Similarly as above, the long position is taken from who has bought the option, whereas the seller takes the short position. The payoff formula for a

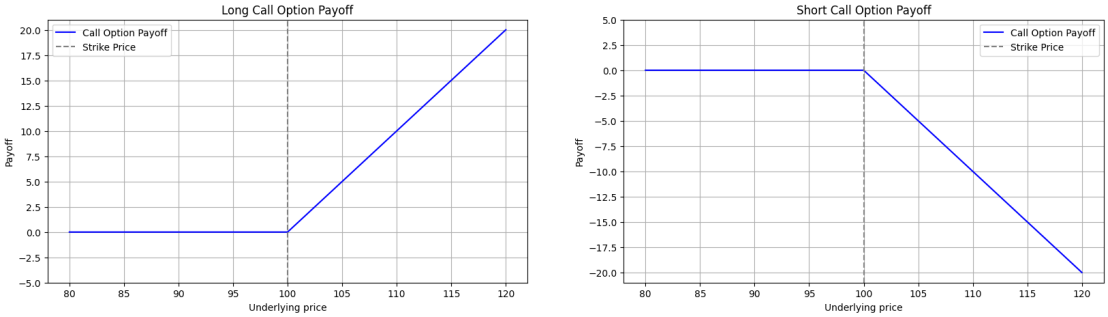
long put option at expiration is:

$$\max(0, K - S_T), \tag{5.3}$$

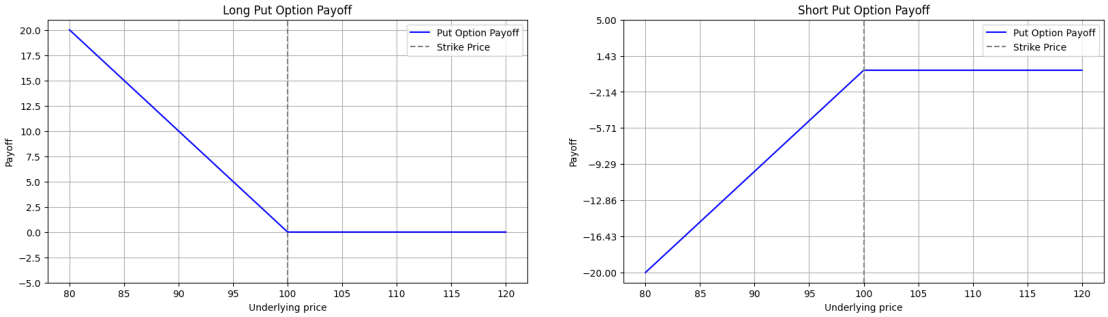
while the payoff formula for a short put option at expiration is:

$$-\max(0, K - S_T) = \min(0, S_T - K). \tag{5.4}$$

In this formula, if the asset price is lower than the strike price, the option holder profits and the seller looses; otherwise, the option expires worthless.



(a) Call option with strike price K = 100



(b) Put option with strike price K = 100

Figure 5.1: Payoff of plain vanilla options

5.1.1 Option pricing methods

Closed-form solutions for pricing European plain vanilla options rely on a general approach that involves computing the expectation of the present value of the vanilla payoffs. In a risk-neutral world, operating under a risk-neutral measure ($\mathbb{Q}$), the current value of a *call* option (C) can be determined by calculating the expected value of the option itself when it is discounted to the present time:

$$C(S, r, T) = e^{-rT}\mathsf{E}^{\mathbb{Q}}\left[\max(0, S_T - K)\right]. \tag{5.5}$$

Similarly, the price of a *put* option (P) at time $t = 0$ can be obtained from its payoff.

$$P(S, r, T) = e^{-rT}\mathsf{E}^{\mathbb{Q}}\left[\max(0, K - S_T)\right], \tag{5.6}$$

where r stands for the *risk-free* interest rate, S_T is the underlying price at time T and K is the strike price. However, it is important to underline that calls and puts prices are related one another by the put-call parity relationship.

In this section, we will explore two distinct methods for determining plain vanilla option prices. We will begin by investigating closed-form solutions for all the four models considered in this work and then proceed to examine Monte Carlo pricing.

Put-Call Parity

Put-call parity establishes a relationship between the prices of European call (C) and put (P) options with the same strike price (K) and expiration date (T), as outlined in [12]. This relationship can be expressed as:

$$C - P = S - \frac{K}{(1+r)^T} \tag{5.7}$$

We will explore this concept further as we delve into closed-form solutions for various option pricing models.

5.1.2 Closed-form solutions

The development of closed-form solutions for pricing European plain vanilla options marked a pivotal moment in the history of finance. In the early 1970s, economists Fischer Black, Myron Scholes, and Robert Merton introduced the groundbreaking Black-Scholes-Merton (BSM) model, revolutionizing the field of option pricing. This model laid the foundation for understanding and valuing financial derivatives, providing a rigorous framework for calculating option prices. The success of the BSM model acted as a milestone in the development of closed-form solutions for various other derivative instruments.

The Black-Scholes-Merton formula

The key assumptions of the Black-Scholes model include:

- Constant volatility of the underlying asset
- Constant risk-free interest rate
- Log-normal distribution of asset returns

The closed-formula for the price of a European call option in the Black-Scholes model is given by:

$$C(S_0, K, T, r, \sigma) = S_0 N(d_1) - K e^{-rT} N(d_2), \tag{5.8}$$

where:

- C is the call option price
- S_0 is the current price of the underlying asset.
- K is the strike price of the option.
- T is the time to expiration.
- r is the risk-free interest rate.
- σ is the volatility of the underlying asset.
- $N()$ is the cumulative standard normal distribution function.
- d_1 and d_2 are defined as:

$$d_1 = \frac{\ln(S_0/K) + (r + (\sigma^2)/2)T}{\sigma\sqrt{T}},$$

$$d_2 = d_1 - \sigma\sqrt{T}.$$

In a similar fashion, the price of a European put option within the Black-Scholes-Merton model can be obtained using the following

$$P(S_0, K, T, r, \sigma) = Ke^{-rT}N(-d_2) - S_0N(-d_1). \tag{5.9}$$

Here, P represents the put option price, and all other symbols maintain the same meanings as in the call option formula. [12]

The Merton Jump Diffusion model

In MJD model, option prices are calculated using closed-formulas that account for the influence of jumps on the underlying asset. Unlike the Black-Scholes-Merton (BSM) model, where volatility is the primary focus, the MJD model places significant importance on two key parameters: the mean jump size m and the jump frequency λ, as discussed in detail in Chapter 3.

To determine the price of a European call or put option with a strike price K, time to maturity T, and the current price of the underlying asset $S(0)$, the general formula, firstly described in [13], can be employed:

$$V_{\text{MJD}}(0) = \sum_{k=0}^{\infty} \frac{\exp(-m\lambda T)(m\lambda T)^k}{k!} V_{\text{BS}}(S, K, T, r, \sigma_k), \tag{5.10}$$

where r is the risk-free interest rate, σ is the volatility, m is the mean jump size, λ is the jump frequency and

- $V_{\text{MJD}}(0)$ can be substituted by C if it indicates the price or a European Call option, or P if it stands for the price of a European Put;
- $V_{\text{BS}}(S, K, T, r, \sigma)$ represents the price of the same option using the Black-Scholes formula.

Kou Jump Diffusion model

The KJD model is a double exponential jump diffusion model. Similar to the Merton Jump Diffusion model, it incorporates jumps described by parameters such as the jump frequency λ, the upward jump probability p, and the mean jump sizes $1/\eta_1$ and $1/\eta_2$, as explained in Chapter 3.

The explicit formulas for European call and put options in the KJD model were derived by Kou in [14]. The price of a European call option with a strike price K, time to maturity T, and the current price of the underlying asset $S(0)$ is given by:

$$\begin{aligned} C_{\mathrm{KJD}}(0) = S(0)\Upsilon\left(r + \frac{1}{2}\sigma^2 - \lambda\zeta, \sigma, \tilde{\lambda}, \tilde{p}, \tilde{\eta_1}, \tilde{\eta_2}; \log(K/S(0)), T\right) \\ - Ke^{-rT}\Upsilon\left(r - \frac{1}{2}\sigma^2 - \lambda\zeta, \sigma, \lambda, p, \eta_1, \eta_2; \log(K/S(0)), T\right). \end{aligned} \tag{5.11}$$

Here,

$$\begin{aligned} \tilde{p} =& \frac{p}{1+\zeta}\frac{\eta_1}{\eta_1 - 1}, \qquad \tilde{\eta_1} = \eta_1 - 1, \qquad \tilde{\eta_2} = \eta_2 + 1, \\ & \tilde{\lambda} = \lambda(\zeta + 1), \qquad \zeta = \frac{p\eta_1}{\eta_1 - 1} + \frac{q\eta 2}{\eta_2 + 1} - 1. \end{aligned}$$

The price of the corresponding put option, $P_{\mathrm{KJD}}(0)$, can be obtained using put-call parity:

$$P_{\mathrm{KJD}}(0) - C_{\mathrm{KJD}}(0) = Ke^{-rT} - S(0). \tag{5.12}$$

The Υ function used in these equations is defined as:

$$\Upsilon(\mu, \sigma, \lambda, p, \eta_1, \eta_2; a, T) = P\{Z(T) \geq a\}$$

Here, $Z(t)$ is the stochastic process that describes the Kou Jump Diffusion model, as given in Equation 3.22. An explicit formula for computing Υ can be found in

[14], Theorem B.1 in Appendix B.

Variance Gamma process

The Variance Gamma (VG) model, which is a pure jump process, is characterized by the key parameters mean rate θ and variance rate ν, in addition to the volatility σ, as elaborated in Chapter 3, Section 3.2.2.

The explicit formulas for European call and put options within the VG model were initially introduced by Madan et al. in [16], marking a significant advancement in the theoretical framework of the VG process, originally theorized by Madan and Seneta in [15]. These formulas have since played a pivotal role in option pricing and have contributed to the broader understanding of VG-based financial derivatives. The price of a European call option with a strike price K, time to maturity T, and the current price of the underlying asset $S(0)$ can be determined using these formulas.

$$\begin{aligned} C_{\mathrm{VG}}(0) = S(0)\psi &\left(d\sqrt{\frac{1-c_1}{\nu}}, (\alpha+s)\sqrt{\frac{\nu}{1-c_1}}, \frac{T}{\nu} \right) \\ - Ke^{-rT}\psi &\left(d\sqrt{\frac{1-c_2}{\nu}}, (\alpha s)\sqrt{\frac{\nu}{1-c_2}}, \frac{T}{\nu} \right), \end{aligned} \tag{5.13}$$

where:

$$\zeta = -\frac{\theta}{\sigma^2}, \qquad s = \frac{\sigma}{\sqrt{1+\left(\frac{\theta}{\sigma}\right)^2 \frac{\nu}{2}}}, \qquad \alpha = \zeta s$$

$$d = \frac{1}{s}\left[log\left(\frac{S(0)}{K}\right) + rT + \frac{T}{\nu} log\left(\frac{1-c_1}{1-c_2}\right)\right],$$

$$c_1 = \frac{\nu(\alpha+s)^2}{2}, \qquad c_2 = \frac{\nu\alpha^2}{2}$$

and the function $\Psi(a, b, \gamma)$ is defined in terms of the modified Bessel function of the second kind, $K_v(z)$, and the degenerate hypergeometric function of two variables, $\Phi(\alpha, \beta, \gamma; x, y)$. Further explanations of these formulas can be found in [16], Appendix.

Furthermore, the corresponding put option price can be obtained using the put-call parity relationship:

$$P_{\mathrm{VG}}(0) - C_{\mathrm{VG}}(0) = Ke^{-rT} - S(0). \tag{5.14}$$

In conclusion, these closed-form solutions have been extensively studied and are detailed in the literature, making them fundamental tools for pricing financial derivatives. The Python code developed for computing the prices of European put and call options in our work will be presented in Appendix A.

5.1.3 Monte Carlo pricing

In the realm of option pricing, Monte Carlo methods play a pivotal role due to the inherent need to compute expected values when valuing derivatives. The ability to simulate the trajectories of stochastic processes that depict the evolution of asset prices provides an indispensable tool for estimating their potential outcomes. As expressed in Equations 5.5 and 5.6, the fair value of European vanilla options can be derived directly from the expected payoff, which is then discounted to the present time using the risk-free interest rate. This simulation-based approach not only offers flexibility in handling complex financial instruments but also accommodates the consideration of various market scenarios. The algorithm for determining the fair value of vanilla options is presented below and serves as a general guideline,

without distinguishing between different models. What sets apart these processes is the computation of $S_i(T)$, which follows the steps outlined in Algorithms 1, 2, 3, 4 and 5.

Algorithm 6 Monte Carlo option pricing

Input: Option type (call/put); Strike price K; Time to maturity T; Number of paths N

Initialization: Set $S_0 = S_0^*$, $i = 1$, $n = 0$, $\hat{V} = 0$

while $i \leq N$ **do**
  Generate $S_i(T)$ according to the chosen process
  $i = i + 1$
end while

if option type == 'call' **then**
  Set $\bar{P}_n = \frac{1}{N} \sum_{i=1}^{N} ((S_i(T) - K, 0)^+)$ ▷ Average payoff
else if option type == 'put' **then**
  Set $\bar{P}_n = \frac{1}{N} \sum_{i=1}^{N} ((K - S_i(T), 0)^+)$ ▷ Average payoff
end if

return $\hat{V}_n = e^{-rT} \bar{P}_n$ ▷ Option price

Denoting V as the price of the European vanilla option (either call or put, depending on the payoff), for any $n \geq 1$, the estimator $\hat{V}_n$ is unbiased, in the sense that its expectation is the target quantity:

$$\mathsf{E}[\hat{V}_n] = V \equiv \mathsf{E}[e^{-rT}(h(S(T))],$$

where $h(S(T))$ represents the payoff function 5.1 or 5.3 depending on the option price. The estimator is strongly consistent, meaning that as $n \to \infty$, $\hat{V}_n \to V$ with probability 1 [20].

5.2 Exotic options: One Touch Knock Out Daily Cliquet

Now, our attention turns to One-Touch Knock-Out (OTKO) Daily Cliquet Options, a subset of exotic options distinguished by unique features setting them apart from European and American options. Our goal is to find optimal models parameters calibrating them on their market prices of these options to identify optimal parameters. This choice stems from the fact that OTKO Daily Cliquet options represent one of the limited instruments within the derivatives market capable of accurately pricing significant downward movements in the underlying asset, whether it's a stock or an index. In this section, we will explain their distinctive structure and payoffs. Next, we will analyze the approximate pricing formulas and the implementation of Monte Carlo pricing for these derivatives. Given their purpose of hedging against rapid downside movements, it is essential to account for sudden jumps in the pricing models. Consequently, the Black-Scholes model, which relies on geometric Brownian motion, is inadequate for these options and assigns them a price of 0, regardless of the volatility parameter chosen. Therefore, we will focus exclusively on the three Lévy processes: Merton, Kou, and Variance Gamma.

Features

This derivative can be seen as a type of short-expiration, typically daily, put option strip. Consequently, it incorporates daily reset points. In the case of a shortfall event occurring during a trading day, the option becomes *in-the-money*, resulting in a payout. If no shortfall event occurs, the option resets and remains active. Its

key characteristics are best explained by the following features:

- *Barriers.* There is an upper barrier K_1 and a lower barrier K_2 expressed as percentages, with $K_1 > K_2$. These are the payoff strikes.

- *One-Touch feature.* For the option to produce a payout, the price of the underlying asset must touch or exceed K_1 or K_2 at least once during the life of the option. If the *touch* level is never reached, the option expires worthless.

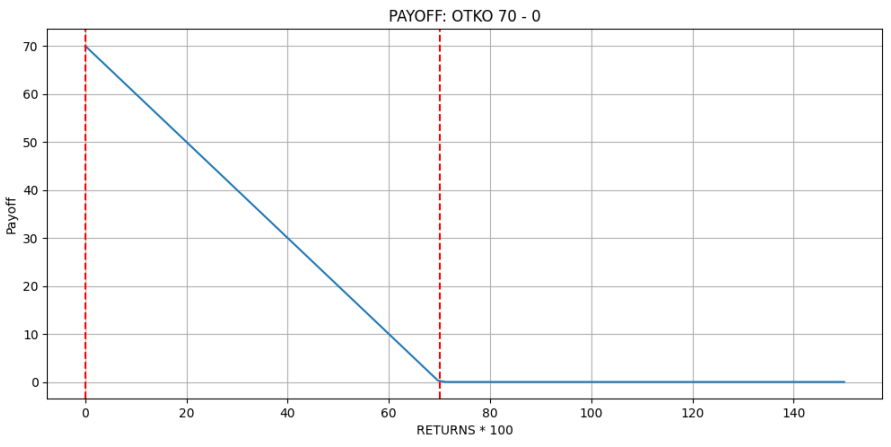
- *Knock-Out feature.* A shortfall event on a specified day t is defined as a negative daily return of the underlying stock price. Specifically, with the daily return denoted as $R_t = \frac{S_t}{S_{t-1}}$, the option *knocks out* if $R_t < K_1$ or, worse, $R_t < K_2$. At that juncture, it triggers a positive payout as determined by its payoff structure and expires.

- *Cliquet feature.* The strike price for the next reset period i s determined based on the spot price of the underlying asset at the end of the previous reset period. In detail, with K_1 and K_2 representing percentages, the strike prices for day $t+1$ are calculated as follows: $K_{1_p} = S_t K_1$ and $K_{2_p} = S_t K_2$.

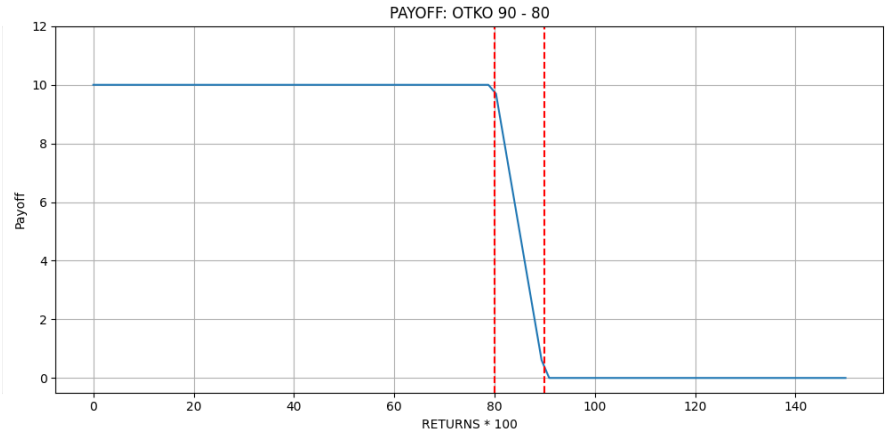
- *Payoff.* It can be expressed according to the following equation

$$\text{payoff} = \min\left\{(K_1 - K_2), \max\left\{0, K_1 - R_t\right\}\right\}, \tag{5.15}$$

  where K_1 and K_2 are the *knock-out* barriers and $R_t = \frac{S_t}{S_{t-1}}$ is the $t-th$ return of the stock price.



(a) OTKO option with barriers $K_1 = 70$, $K_2 = 0$



(b) OTKO option with barriers $K_1 = 90$, $K_2 = 80$

Figure 5.2: Payoff of One-Touch Knock-Out Daily Cliquets Options

5.2.1 Option pricing methods

As in the case of European vanilla options, the general price formula for One-Touch Knock-Out Daily Cliquet Options (P_{otko}) can be determined by discounting the expected value of its payoff to the present time. This expectation is calculated

under a risk-neutral measure $\mathbb{Q}$. Consequently, P_{otko} is expressed as follows:

$$P_{\text{otko}} = e^{-rT}\mathsf{E}^{\mathbb{Q}}\left[\min\left\{(K_1 - K_2), \max\left\{0, K_1 - R_t\right\}\right\}\right], \tag{5.16}$$

where r is the risk-free interest rate, T is the option's time to maturity, and $\mathsf{E}^{\mathbb{Q}}$ indicates that we are operating within a risk-neutral framework.

In this section, we will explore two methods for determining the fair value of the OTKO Daily Cliquets price P_{otko}. To this purpose, will derive the approximate pricing formulas tailored to our Lévy processes and we will present the Monte Carlo pricing technique.

5.2.2 Approximated pricing formula

Pricing formulas for a type of closely related options, known as *Gap Options* or *Crash Notes*, were meticulously examined by Tankov in 2008 [21]. In this work, the author introduced a method for calculating the exact price of a gap option, which involves a computationally expensive integral of the characteristic function of the chosen model. Recognizing the high computational cost, Tankov also provided an accurate approximate pricing formula suitable for Lévy processes. In this section we will see how this approximate formula can be applied to our objectives, namely the pricing of One-Touch Knock-Out Daily Cliquets options, although there may be slight differences between the two.

What the Otko Daily Cliquets Options and the Gap Options described in [21], have in common, is their reliance on assessing the daily performance of the underlying stock (close to close). Both options require the occurrence of a gap event or the reaching of a specific level to trigger an outcome. Following the occurrence of

the gap event, characterized by a downside movement exceeding a predetermined barrier, both options payout and terminate.

The distinctions between these two exotic options primarily stem from their structural differences. Specifically, Crash Notes rely on the singular strike level K as the trigger for the gap event. In contrast, One-Touch Knock-Out (Otko) Daily Cliquets feature two distinct barriers, which serve as strikes. Due to this structural variation, the final payoff slightly differs, accounting for the presence of the additional lower strike. It's worth noting that while the upper strike must necessarily be $K_1 > 0$, there are some Otko Daily Cliquets options for which the lower strike is set to $K_2 = 0$. This scenario is illustrated in Figure 5.2a. In such cases, there is no discernible difference between the two options at hand.

The approximate pricing formula, as presented in [21], is derived under the assumption that $S_t = S_0 e^{X_t}$, where X is a Lévy process. If $\nu(dx)$ represents its Lévy density, as discussed in Section 3.2, then the price of such an option at time $t = 0$ can be expressed as:

$$P(0) = \int_{-\infty}^{\beta} f(e^x)\nu(dx)\frac{1 - e^{-rT - T\int_{-\infty}^{\beta}\nu(dx)}}{r + \int_{-\infty}^{\beta}\nu(dx)}, \tag{5.17}$$

where r denotes the risk-free interest rate, T is the time to maturity, $\beta := \log K < 0$, and $f(x)$ represents the payoff of the gap options. It can be noticed that this formula is universally applicable, in fact it is only necessary to substitute the correct payoff expression into $f(e^x)$.

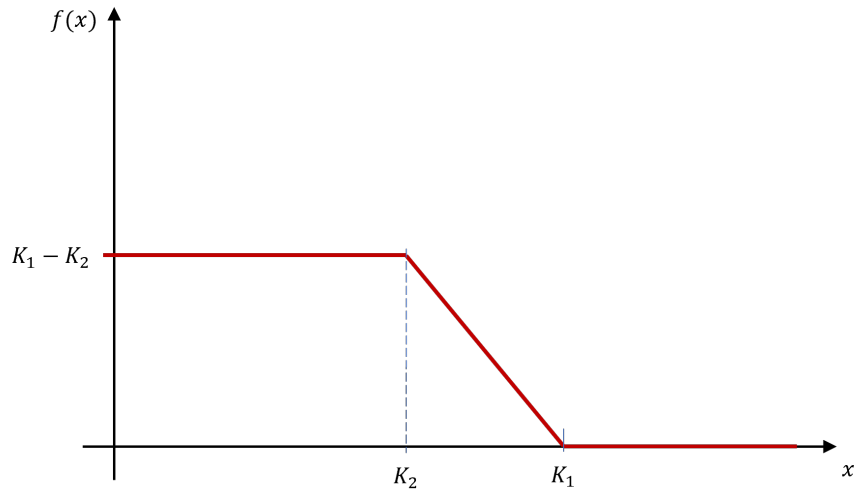


Figure 5.3: Otko Daily Cliquets payoff

The payoff of the One-Touch Knock-Out (Otko) Daily Cliquets Options, illustrated in Figure 5.3, can be expressed analytically as follows:

$$f(x|K_1, K_2) = (K_1 - K_2)\mathbb{1}_{\{0,K_2\}}(x) + (K_1 - x)\mathbb{1}_{\{K_2,K_1\}}(x). \tag{5.18}$$

Consequently, we can derive the expression for the approximate pricing formula applicable to all Lévy processes by substituting it into Equation 5.17. This results in the following expression:

$$\int_{-\infty}^{\beta} f(e^x)\nu(dx) = \int_{-\infty}^{\alpha} (K_1 - K_2)\nu(dx) + \int_{\alpha}^{\beta} (K_1 - e^x)\nu(dx), \tag{5.19}$$

Here, $\alpha := \log K_2 < 0$ and $\beta := \log K_1 < 0$. The specific price P_{otko}, depending on the chosen model, will then rely on the Lévy measure of that model, which should be incorporated into Equation 5.17.

Merton Jump Diffusion model

The Lévy density for the MJD model can be described by the following equation [22, p. 109]:

$$\nu_{MJD}(x) = \frac{\lambda}{\delta\sqrt{2\pi}} \exp\left\{-\frac{(x-m)^2}{2\delta^2}\right\}, \tag{5.20}$$

where λ represents the jump intensity, while m and δ denote the mean and standard deviation of the jump size, respectively.

The integrals in Equation 5.17 can be computed explicitly as follows:

$$\int_{-\infty}^{\beta} \nu(dx) = \lambda\Phi(K_1), \tag{5.21}$$

$$\int_{-\infty}^{\beta} f(e^x)\nu(dx) = \lambda\left[K_1\Phi(\beta) - K_2\Phi(\alpha) - e^{m+\frac{\delta^2}{2}}\left(\Phi(\beta-\delta^2) - \Phi(\alpha-\delta^2)\right)\right], \tag{5.22}$$

where Φ represents the cumulative distribution function of a Gaussian random variable with mean m and standard deviation δ, so that $\Phi(x) = \Phi\left(\frac{x-m}{\delta}\right)$.

Combining these two integrals in Equation 5.17 will yield a good approximation of fair price for the One-Touch Knock-Out Daily Cliquet options according to the Merton model.

Kou Double Exponential Jump Diffusion model

The Double Exponential Jump Diffusion process is characterized by a Lévy density as described in [22, p.109]:

$$\nu_{KJD}(dx) = \lambda p\eta_1 e^{-\eta_1 x}\mathbb{1}_{x>0} + \lambda(1-p)e^{-\eta_2|x|}\mathbb{1}_{x<0} \tag{5.23}$$

Here, λ represents the total jump intensity, p denotes the probability of an upward jump occurring, and $1/\eta_1$ and $1/\eta_2$ signify the means of the upward and downward jump sizes, respectively.

When substituting this Lévy density into the expression 5.17, we derive the following:

$$\int_{-\infty}^{\beta} \nu(dx) = \lambda(1-p)e^{\beta\eta_2} \tag{5.24}$$

$$\int_{-\infty}^{\beta} f(e^x)\nu(dx) = \frac{\lambda(1-p)}{1+\eta^2}\left(K_1^{1+\eta_2} - K_2^{1+\eta_2}\right) \tag{5.25}$$

In these equations, $\beta = \log K_1$. Consequently, we can easily determine the approximate price of an Otko Daily Cliquet Option using the Kou model.

Variance Gamma process

As described in Section 3, Eq. 3.38 is one of the three available forms of the Lévy measure for the Variance Gamma process. Another representation, more suitable for the purposes of this chapter, can be found in [23]. It is expressed in terms of the VG parameters $\mu_\pm$ (3.35) and $\nu_\pm$ (3.36) and is given by:

$$\nu_{VG}(dx) = \begin{cases} \frac{\mu_n^2 \exp\left(-\frac{\mu_n}{\nu_n}|x|\right)}{\nu_n|x|}\,dx & \text{for } x < 0 \\ \frac{\mu_p^2 \exp\left(-\frac{\mu_p}{\nu_p}x\right)}{\nu_p x}\,dx & \text{for } x > 0 \end{cases} \tag{5.26}$$

The Lévy density for the Variance Gamma (VG) process can be expressed in a more compact and useful form as follows:

$$\nu_{VG}(dx) = \frac{C}{|x|}\exp(Gx)\mathbb{1}_{x<0} + \frac{C}{x}\exp(-Mx)\mathbb{1}_{x>0}, \tag{5.27}$$

where:

$$
\begin{aligned}
C &= \frac{1}{\nu}, \\
G &= \left(\sqrt{\frac{1}{4}\theta^2\nu^2 + \frac{1}{2}\sigma^2\nu} - \frac{1}{2}\theta\nu \right)^{-1}, \\
M &= \left(\sqrt{\frac{1}{4}\theta^2\nu^2 + \frac{1}{2}\sigma^2\nu} + \frac{1}{2}\theta\nu \right)^{-1}.
\end{aligned}
$$

Then, we can solve the integrals in 5.17 to find the price of the Otko Daily Cliquet Options according to the VG model:

$$
\int_{-\infty}^{\beta} \nu(dx) = -\frac{C}{G} \operatorname{expi}(\beta G), \tag{5.28}
$$

$$
\begin{aligned}
\int_{-\infty}^{\beta} f(e^x)\nu(dx) = \frac{C}{G} &\left(K_2 \operatorname{expi}(\alpha G) - K_1 \operatorname{expi}(\beta G) \right) \\
&+ \frac{C}{G+1} \left(\operatorname{expi}(\beta(G+1)) - \operatorname{expi}(\alpha(G+1)) \right),
\end{aligned} \tag{5.29}
$$

where $\alpha = \log K_2$, $\beta = \log K_1$, and $\operatorname{expi}(x) = \int_{-\infty}^{x} \frac{e^t}{t} dt$ is the exponential integral.

5.2.3 Monte Carlo pricing

In Section 5.1.3, we discussed how Monte Carlo pricing methods are a powerful tool suitable for options with known payoffs. To determine the price of One-Touch Knock-Out Daily Cliquet Options, we simulate multiple trajectories of various processes, calculate their expected values, and discount them to the present value using Equation 5.16. It's essential to note that each process generates its unique

set of trajectories. Thus, the selection of the desired process and the execution of simulations, following the procedures outlined in Algorithms 2, 3, and 4 or 5, will yield N paths denoted as $S_i(t)$ with $i = 1, \ldots, N$. The following algorithm shows how the returns generated on each path are compared with the two strike barriers to assess whether or not the option will be exercised. At the end of the two loops, the payoffs of all paths are obtained, the expected value is calculated, and the estimated price of the option is found.

Algorithm 7 Otko Option Pricing

Input: Paths of stock prices $S_i(t)$, upper barrier $K1$, lower barrier $K2$, time to maturity T, num of generated paths N

Initialization: Set all payoffs $\pi = 0$

while $i \leq N$ **do**

  for each t from 0 to T **do**

    Compute daily return $R_i(t) = S_i(t)/S_i(t-1)$;

    if $R_t > K1$ **then** ▷ rolls to the next trading day

      $\pi_i \leftarrow 0$

    else if $K2 < R_t \leq K1$ **then** ▷ payouts and terminates

      $\pi_i \leftarrow K1 - R_t$

      $i = i + 1$

      return π_i

    else if $R_t \leq K2$ **then** ▷ payouts and terminates

      $\pi_i \leftarrow K1 - K2$

      $i = i + 1$

      return π_i

    end if

  end for

  $i = i + 1$

  return π_i

end while

Set $\bar{\Pi}_n = \frac{1}{N} \sum_{i=1}^{N} \pi_i$ ▷ average payoff

return $\hat{P}_{\text{otko}} = e^{-rT} \bar{\Pi}_n$ ▷ option price

Chapter 6

Model calibration

In the context of pricing margin loans using the Lévy processes presented in Chapter 3 the accurate determination of model parameters is crucial: the choice of these parameters deeply influences the trajectory of the paths generated in Monte Carlo simulations, which in turn are critical in pricing financial instruments such as margin loans. To determine the correct parameters, a calibration process is used. In financial markets, a pool of options with known prices serves as a reference point for this calibration exercise. These market prices act, in fact, as reference points. The goal is to replicate observed option prices using our stochastic models, as outlined in Chapter 5. This entails exploring the model's parameter space and identifying values that minimize the difference between market prices and model-generated prices. Consequently, parameter calibration is framed as an optimization problem, specifically, a *least-squares* problem, wherein an objective function is minimized by minimizing the squared differences between market and model prices.

The Calibration Approach

Our calibration procedure began with S&P500 options due to their popularity and liquidity. European vanilla options served as our initial benchmark. However, the results of our S&P500 options calibration are not presented here, as they are beyond the scope of this study. Our primary interest lies in analyzing individual company movements, as options on their stock prices provide valuable insights for parameterizing our simulations. The initial phase using S&P500 options helped us understand the correlation between the movement of asset values in the stock market and options prices. Thus, we have shifted our attention to three separate individual stocks: AAPL, TSLA and NVDA. European vanilla option prices of these securities were taken from a financial data provider [24]. In addition, we incorporated One-Touch Knock-Out Daily Cliquets Options into our calibration dataset, which, due to their over-the-counter nature, have lower market liquidity. Price data for these options were provided by *Intesa Sanpaolo S.p.A.*
The observed market prices date back to January 20, 2016, with expiration on January 20 of the following year, thus with a time-to-maturity $T = 1$.

There were two key reasons for this selection of assets.
First, these companies, especially at that moment in history, could be considered as candidates for margin loan contracts. Although they do not perfectly embody the ideal profile for such instruments, given their large market capitalization and market positioning at the time, they provided a solid basis for our comprehensive analysis.
Second, the options associated with these securities are abundant, which facilitates

in-depth and precise analysis, thanks in part to the ready availability of historical data. Detailed reports of these analysis will be explained later in this chapter.

6.1 Implied volatility

In addition to Lévy processes, Chapter 3 introduced the *Geometric Brownian Motion* (GBM), the stochastic process behind the Black-Scholes model. GBM is characterized by a single unobservable parameter known as volatility, which is the standard deviation of daily log-returns in a stock's price. In the context of options pricing, the BS model employs implied volatility. This *forward looking* metric estimates the future volatility of financial assets, such as stocks or indexes, based on option prices associated with those assets, as outlined in [12]. Implied volatility, denoted as σ, can be regarded as the level of volatility that, when incorporated into the Black-Scholes formula, yields the observed market price (P) of an option:

$$BS(S_0, K, T, r, \sigma) = P.$$

In this equation, discussed in Section 5.1.2, the variables S_0 (the current underlying price), K (the strike price), T (the time to maturity), and r (the risk-free interest rate) are all directly observable within the market. The implied volatility (σ), instead, can be determined by reversing the Black-Scholes formula, using numerical methods such as the Bisection method or the Netwon-Raphson method.

The Newton-Raphson method

The Newton-Raphson method is an iterative numerical technique for finding the roots of a real-valued function. The method starts with an initial guess x_0 and

refines it step-by-step to get closer to the actual root. At each step, it linearly approximates the function and solves for the root using the formula:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

Where:

- x_{n+1} is the next approximation
- x_n is the current approximation.
- $f(x_n)$ is the function value at x_n.
- $f'(x_n)$ is the derivative (first derivative) of the function at x_n.

This process iterates until the desired level of accuracy is reached or until $f(x_n)$ approaches zero sufficiently, converging to the root and iteratively improving the approximation.

When it comes to option pricing, this method is reliable and allows us to determine the correct value of σ corresponding to the given parameters. It is important to note that the Black-Scholes model assumes a constant σ for all strike prices K and times to maturity T, but in reality, σ varies with both. The relationship between σ and K is known as the volatility smile, while its dependence on T is called the term structure. Additionally, the relationship involving both $\sigma(K, T)$ is referred to as the volatility surface.

In our study, we have considered the time-to-maturity to be fixed, i.e., $T = 1$. We utilized the *fsolve* method from the Python library *scipy.optimize* to find the volatility smile of European vanilla options on our stocks. This method implements an improved Newton-Raphson method.

Figure 6.1 displays the volatility smiles for both call and put options on AAPL stock, traded on January 20, 2016, and expiring on January 20, 2017. These options exhibit a moneyness ratio, i.e., the ratio K/S_0, within the range $[0.75, 1.35]$.

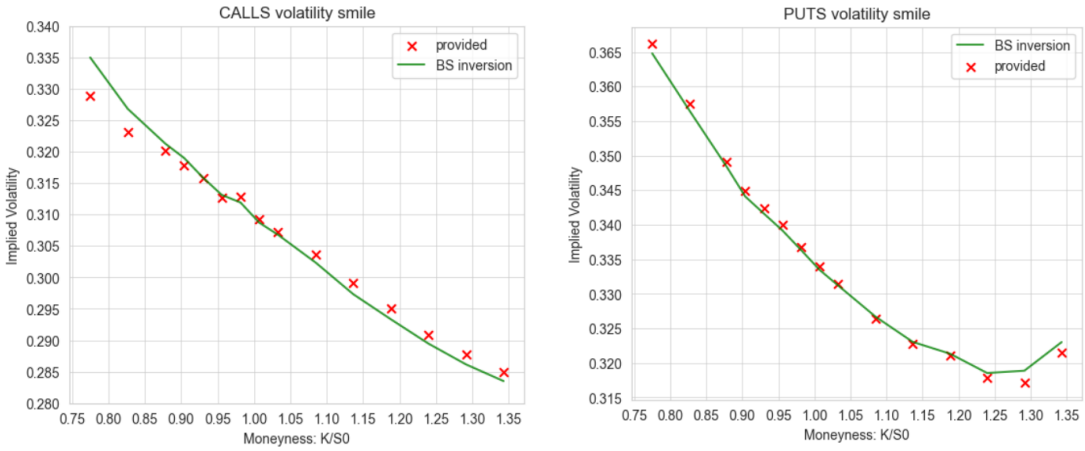


Figure 6.1: Observed and calibrated implied volatilities of 1 year options on the AAPL stock, as a function of moneyness K/S0.

6.2 Models Calibration on Option Prices

The full calibration of all parameters for the Lévy processes involves several steps for each stock (AAPL, NVDA, TSLA) and each stochastic model (MJD, KJD, and VG):

1. Parameter calibration based on European Vanilla Options
2. Parameter calibration based on Otko Daily Cliquets Options
3. Parameter calibration based on a mixed set of all options

We will start with an initial calibration solely based on European vanilla options. This serves as a testing phase to validate the effectiveness of our formulas and provides us with a benchmark for subsequent calibrations. It is expected that due to the unique characteristics of one-touch knock-out options, the parameters calibrated for these options may appear more aggressive. In other words, these parameters may reflect a higher level of volatility in the underlying price movements, considering the specific dynamics and features of Otko options. We will closely monitor whether these expectations hold during the calibration process.

Weighted calibration

As extensively shown in [25], if we define Θ the set of parameters, the goal is to find the optimal parameters Θ^* that minimize the following objective function:

$$\sum_{i=1}^{N} w_i \Big(P_i - f(K_i|\Theta) \Big)^2$$

where w_i are weights, defined as

$$w_i = \frac{1}{\text{spread}_i^2} = \frac{1}{|P_i^{\text{bid}} - P_i^{\text{ask}}|^2}.$$

Here, P_i are the market prices and f is the pricing function, which depends on the set of strike prices K_i (where $i = 1, \ldots, N$). The specific form of f varies for each Lévy process, as elaborated in Section 5. To address this optimization problem, the `least_squares` method from the `scipy.optimize` library has been employed, which is well-suited for tackling the Nonlinear Least Squares (NLS) problem.

Nonlinear least squares optimization problem

The NLS problem aims to find the model parameters that minimize the sum of squared differences between model predictions and observed data. It is used when fitting complex models to data, especially when the relationship between variables is nonlinear. The `least_squares` method, used with the `trf` algorithm, aims to solve constrained minimization problem, allowing for setting boundaries condition and a starting point. The `trf` algorithm, also known as *Trust-Region-Reflective*, iteratively adjusts model parameters while adhering to a predefined *trust region* representing the acceptable range of parameter changes. During each iteration, it assesses the cost function to determine whether parameter adjustments enhance the fit. If improvements are observed, the adjustments are accepted. Otherwise, the algorithm either reduces the step size or changes direction. This process allows the algorithm to navigate the parameter space efficiently, converging towards a solution. Convergence occurs when one of the termination conditions is met. [26]

6.2.1 Calibration on European Vanilla Options

In this context, options with a K/S_0 ratio between 0.75 and 1.35 were chosen. This range offers computational convenience, reducing numerical and stability issues. Furthermore, it is considered to be representative of typical market behavior. Options with moneyness close to 1 (i.e., options close to the current price of the underlying asset) tend to exhibit higher liquidity and more active trading.

Algorithm The pseudo-code 8 demonstrates how the parameters of the Merton Jump-Diffusion model are calibrated. It is required to minimize the cost function, i.e. the objective function which measures the sum of squared differences between the market prices of put options and the prices predicted by the model.
The algorithm takes an initial guess denoted as x_0. and an array of bounds as inputs. The initial guess set helps the convergence to the optimal solution while bounds to set boundaries for the parameter space. The outcome is an array containing the best estimation for the model parameters (σ, λ, m, v). Here's a breakdown of the steps:

1. The cost function updates iteratively the estimation of the four model parameters.

2. Using the current parameter estimates, the cost function computes the theoretical prices for put options.

3. It calculates the differences between the market option prices and the model prices, weighting them by the bid-ask spread of each option.

4. The sum of squared differences quantifies the overall error.

5. The optimization algorithm finds the parameter set that minimizes this error.

6. An array containing the best estimates for each parameter is returned.

The same procedure is performed with call option prices, using the specific Merton formula for them.

Algorithm 8 Parameter Calibration for Put Options

Input: Initial guess x_0, bounds $[lb, ub]$, strike prices $strikes$, market prices mkt_prices, weights w

Initialization: $x \leftarrow x_0$

function COST_FUNCTION($x, strikes, mkt\_prices$)

$sigma, lambd, meanJ, stdJ \leftarrow x$

$M \leftarrow$ Merton_pricer($S0, T, r, q, sigma, lambd, meanJ, stdJ$)

$th\_prices \leftarrow M.closed\_formula\_put(strikes)$

$sq\_err \leftarrow \sum w \cdot (th\_prices - mkt\_prices)^2$

return sq_err

end function

Optimization:
$x \leftarrow$ least_squares($cost\_function, x_0, args = (strikes, mkt\_prices), bounds = [lb, ub]$)

return $x \leftarrow x[:4]$

It's worth noting that this calibration process remains consistent when applying other models such as the Kou Jump-Diffusion (KJD) or Variance Gamma (VG) models. The primary differences lie in the choice of the model pricer class and the specific formula used to compute option prices. All the formulas described in Section 5 have been implemented for this purpose. Some results of the calibration process are shown in Figure 6.2 and 6.3.

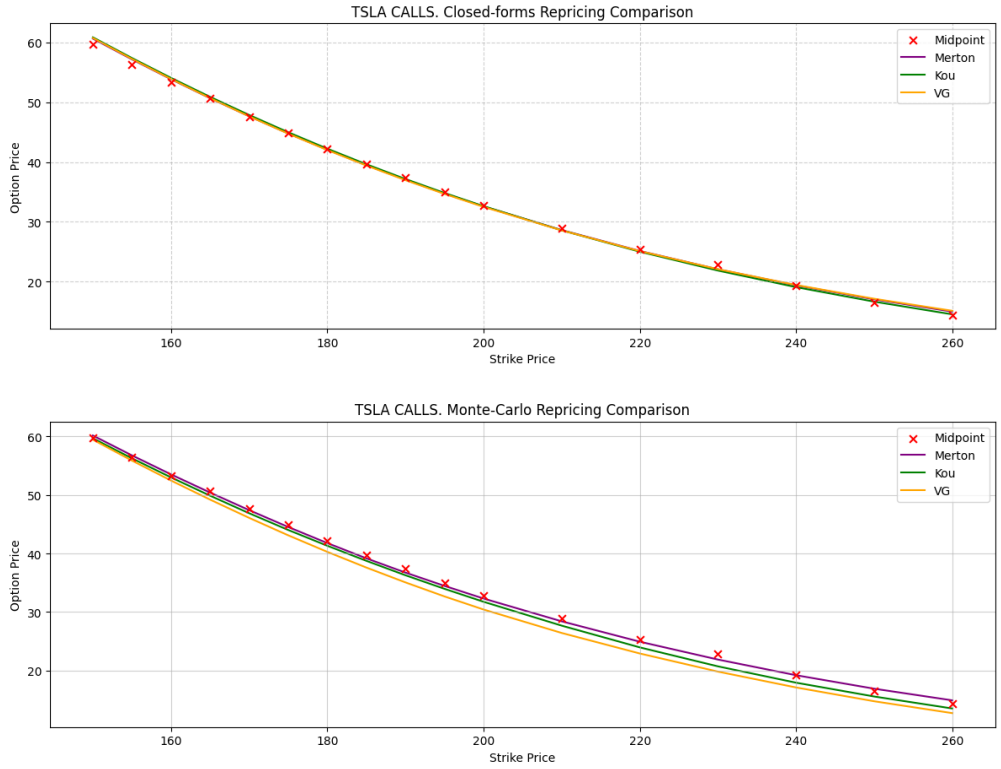


Figure 6.2: TSLA call options repriced using closed formulas and Monte-Carlo pricing with the calibrated parameters

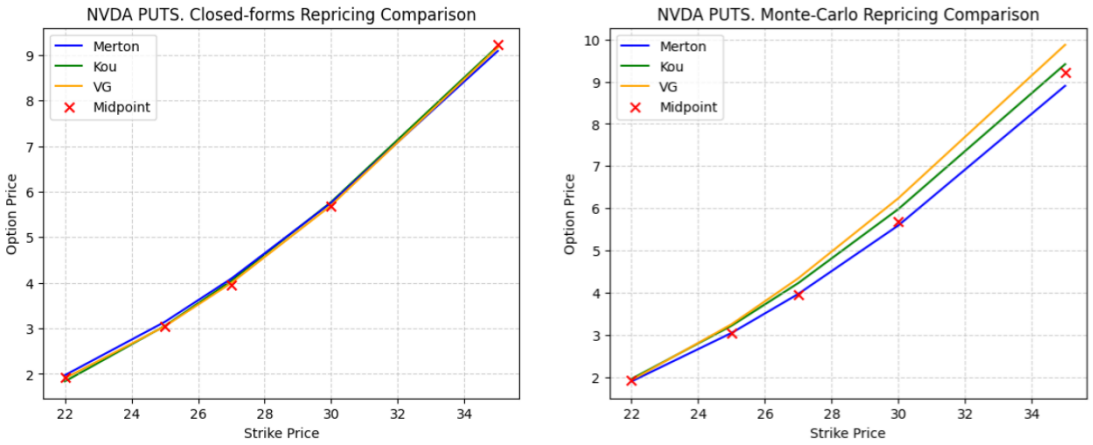


Figure 6.3: NVDA put options repriced using closed formulas and Monte-Carlo pricing with the calibrated parameters

6.2.2 Calibration on Otko Daily Cliquet Options

In this section, we will discuss the calibration of One-Touch Daily Cliquets (OTKO) options using approximate formulas, as previously explored in Chapter 5. We have at our disposal five Otko contracts, all with a one-year maturity. It is important to note that these five contracts differ based on the pairs of strikes $[K1/100, K2/100]$, representing upper and lower barriers, where $K1$ and $K2$ are percentages. Specifically, we will consider a set of options composed of the following strike pairs: $[0.70, 0]$, $[0.75, 0]$, $[0.80, 0.70]$, $[0.85, 0.75]$, and $[0.90, 0.80]$.

Algorithm Similarly to what Algorithm 8 demonstrates, the following 10 illustrates the calibration method for the Variance Gamma process parameters, focusing exclusively on One-Touch Knock-Out Daily Cliquets Options. The steps involved in this calibration process closely resemble those in the previous pseudo-code with the primary distinction being the utilization of a specific VG function designed to

approximate option prices, as elaborated upon in Section 5.2.2. The process of calibrating parameters for the Merton Jump-Diffusion and Kou Jump-Diffusion models remains consistent, requiring only the use of the correct model and the application of the corresponding pricing formula.

Algorithm 9 Parameter Calibration for Otko Options

Input: Initial guess x_0, bounds $[lb, ub]$, strike prices $strikes$, market prices mkt_prices, weights w

Initialization: $x \leftarrow x_0$

function COST_FUNCTION($x, strikes, mkt\_prices$)

$sigma, theta, nu \leftarrow x$

$VG \leftarrow$ VG_pricer($S0, T, r, q, sigma, lambd, nu$)

$th\_prices \leftarrow VG.closed\_formula\_otko(strikes)$

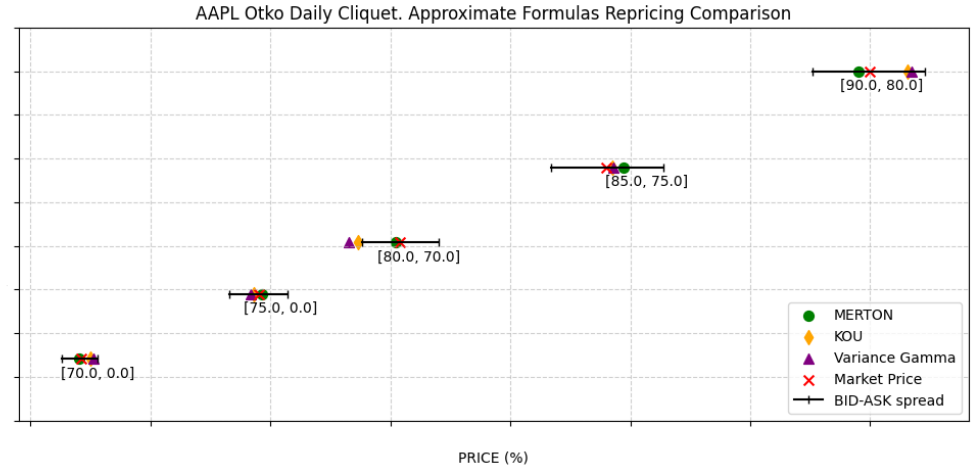
$sq\_err \leftarrow \sum w \cdot (th\_prices - mkt\_prices)^2$

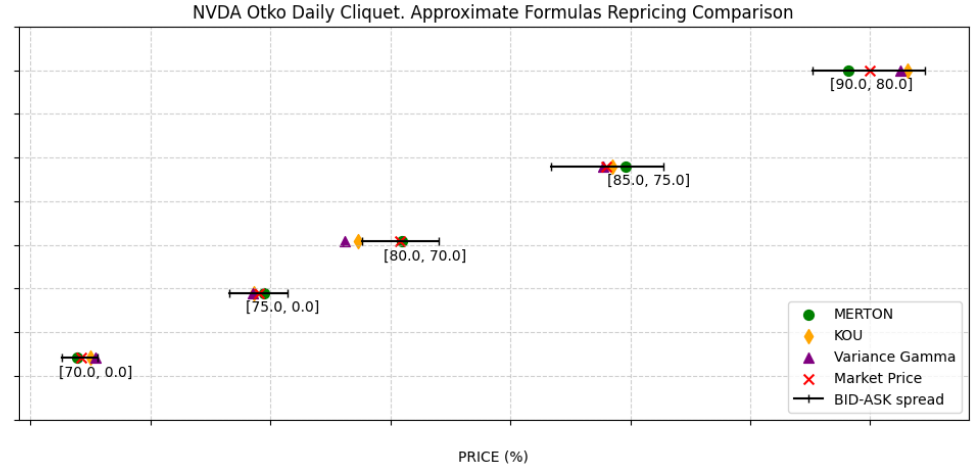
return sq_err

end function

Optimization:
$x \leftarrow$ least_squares($cost\_function, x_0, args = (strikes, mkt\_prices), bounds = [lb, ub]$)

return $x \leftarrow x[:3]$





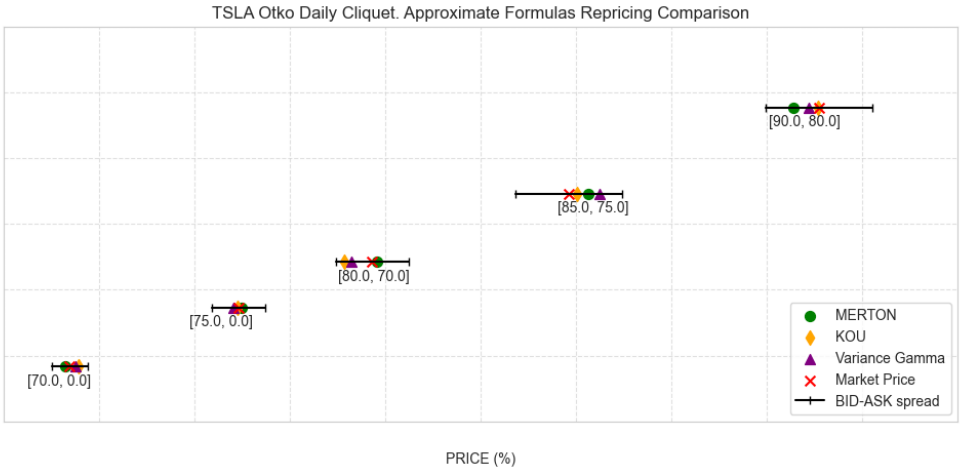
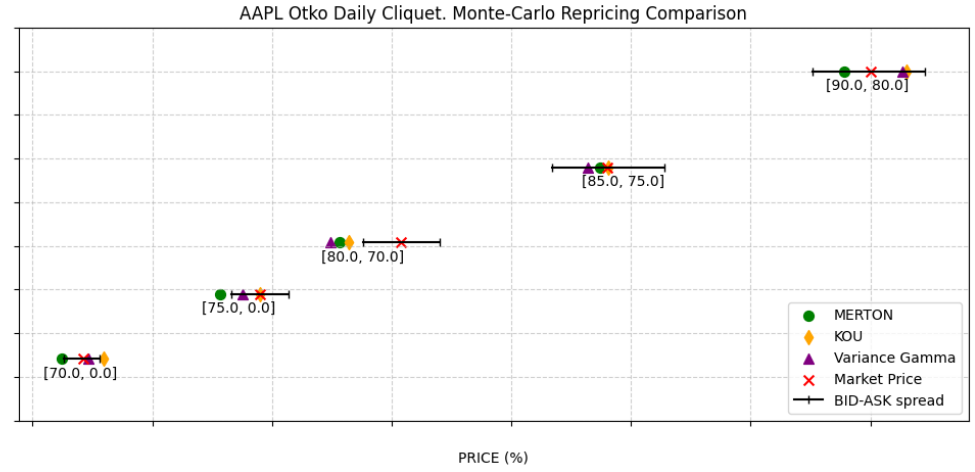
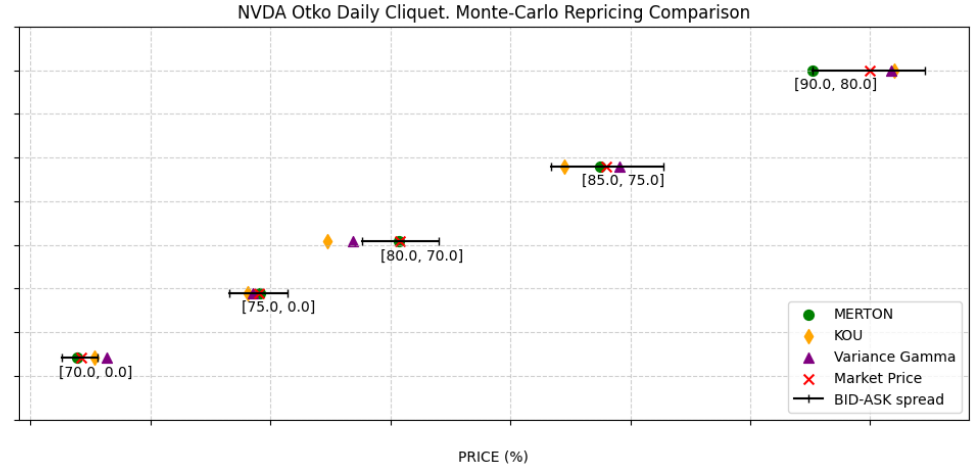


Figure 6.4: Otko Daily cliquets option on all stocks repriced using closed formulas with the calibrated parameters





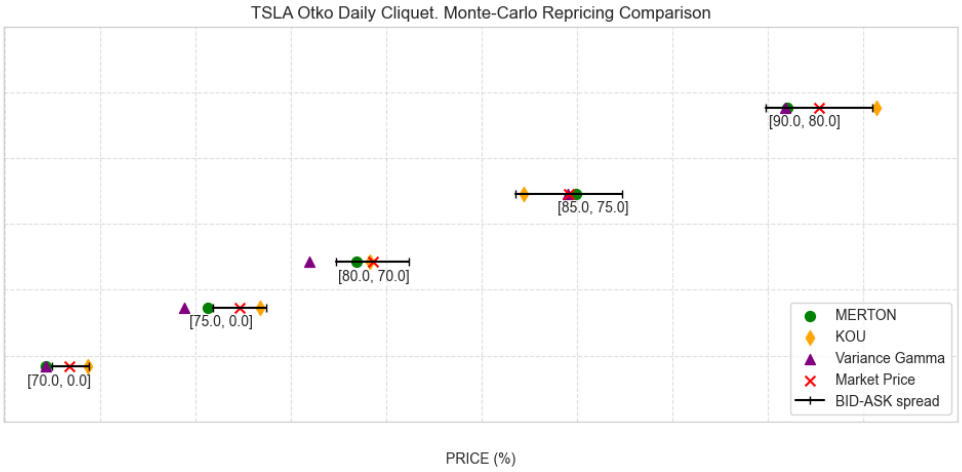


Figure 6.5: Otko Daily cliquets options on all stocks repriced using Monte-Carlo technique with the calibrated parameters

Figures 6.4 and 6.5 contain qualitative plots that illustrate the results of this calibration process applied to all the models for each stock. In these plots, the black segment represents the market bid-ask spread, while the midpoint is indicated with a red marker labeled *Market price*.

It is evident that the best calibration results are achieved for the TSLA stock, where all models reprice the options with nearly zero error, whether utilizing closed-form solutions or Monte Carlo pricing. For AAPL and NVDA, the only notable concern arises with the Kou model, which experiences minor difficulties in pricing the first two options, those with $K2 = 0$. Nonetheless, there remains a high degree of consistency between approximate formulas and Monte Carlo technique.

6.2.3 Calibration on a Mixed Set of Options

At this stage, we conducted a Mixed calibration, incorporating a total of 5 Call Options, 5 Put Options, and 5 One-Touch Knock-Out Daily Cliquet Options, all sharing the same start and expiry date. For each stock and each model under consideration, the calibration process entailed the following steps:

1. Starting with an initial guess for the model's parameters.
2. Defining bounds within the parameter space, as done in previous cases.
3. Optimizing the cost function, which, in this instance, differs from previous calibrations because it takes into account all three types of options and, consequently, the sum of squared errors for each.

Algorithm 10 outlines the steps and the content of the cost function. We have instantiated the `<model_name>_pricer` class, where parameter estimates are updated progressively. Subsequently, we determine theoretical prices and aggregate

the squares of all errors. We specify that we will showcase the code for searching the best-fitting parameters of the Kou model. However, as in previous cases, Merton and Variance Gamma models are interchangeable.

Algorithm 10 Parameter Calibration for Otko Options

Input: Initial guess x_0, bounds $[lb, ub]$, call strikes $strikes$, put strikes $p\_strikes$, otko strikes $o\_strikes$, call market prices c_mkt_prices, put market prices p_mkt_prices, otko market prices o_mkt_prices

Initialization: $x \leftarrow x_0$

function COST_FUNCTION($x, strikes, mkt\_prices$)

$\sigma, \lambda, p, \eta_1, \eta_2 \leftarrow x$

$K \leftarrow \text{Kou\_pricer}(S0, T, r, q, \sigma, \lambda, p, \eta_1, \eta_2)$

$c\_th\_prices \leftarrow K.closed\_formula\_call(c\_strikes)$

$sq\_err \leftarrow \sum(c\_th\_prices - c\_mkt\_prices)^2$

$p\_th\_prices \leftarrow K.closed\_formula\_put(p\_strikes)$

$sq\_err2 \leftarrow \sum(p\_th\_prices - p\_mkt\_prices)^2$

$o\_th\_prices \leftarrow K.closed\_formula\_otko(o\_strikes)$

$sq\_err3 \leftarrow \sum(o\_th\_prices - o\_mkt\_prices)^2$

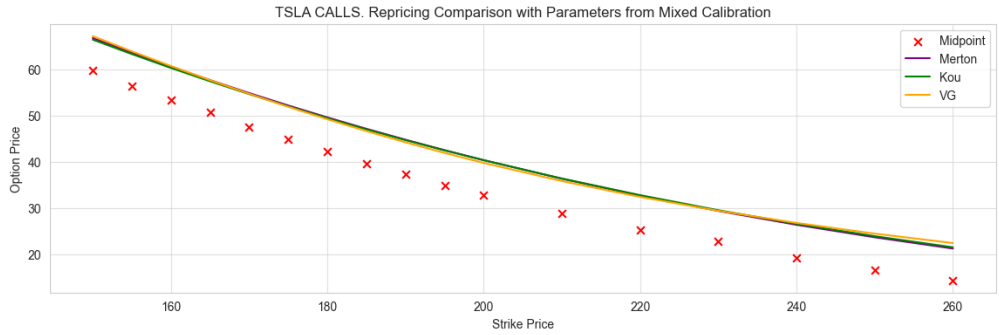
return $sq\_err1 + sq\_err2 + sq\_err3$

end function

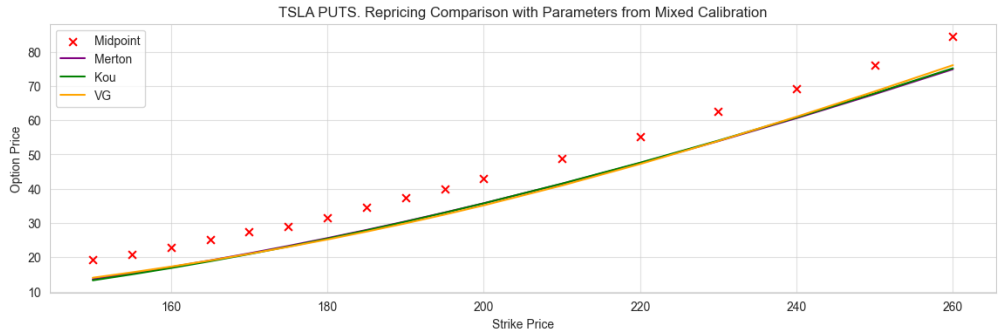
Optimization:
$x \leftarrow \text{least\_squares}(cost\_function, x_0, args, bounds = [lb, ub])$

return $x \leftarrow x[:5]$

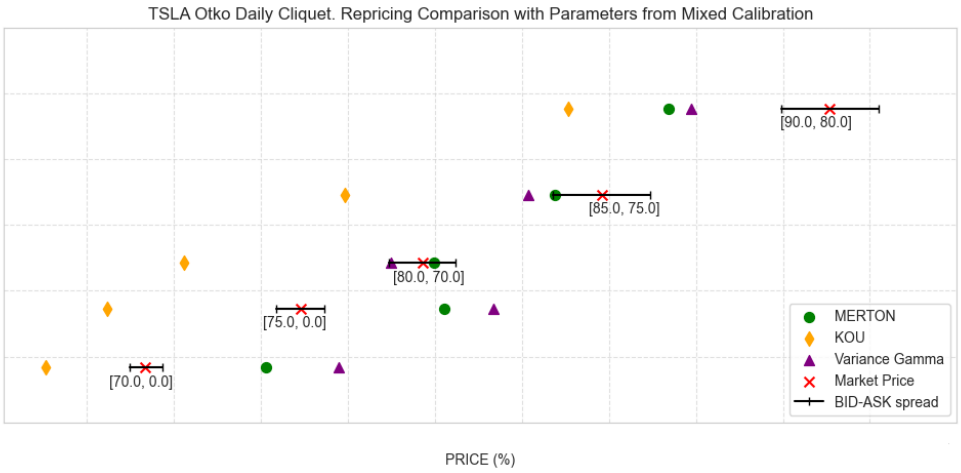
It is important to emphasize that, in this third case, a weighted calibration approach is not feasible. The reason lies in the significantly different characteristics of the options under consideration, particularly in terms of bid-ask spreads. The bid-ask spread serves as an indicator of option liquidity, reflecting the volume of market trading activity. Notably, European vanilla options exhibit bid-ask spreads that are substantially narrower than those of Otko Daily Cliquets options. Attempting a weighted calibration would entail assigning a disproportionately high weight to the vanillas and an exceedingly low weight to the OTKO options. This would essentially revert to the calibration approach illustrated in Section 6.2.1.



(a) Call Options for TSLA stock repriced by all models calibrated on mixed set



(b) Put Options for TSLA stock repriced by all models calibrated on mixed set



(c) Otko Options for TSLA stock repriced by all models calibrated on mixed set

Figure 6.6: All options repriced using closed-forms solution with the parameters from the mixed calibration

Some results of this last calibration can be observed in Fig. 6.6. The reported results were obtained by running the three models with parameters resulting from the mixed calibration. To validate these findings and appreciate the differences in the calibration processes, the initial options were repriced, grouped by option type. We present the results of this repricing performed with TSLA options. It is worth noting that the results obtained with other stocks present the same behaviour, i.e. the identified patterns remain consistent. Specifically:

1. Fig. 6.6a Repricing call options using non-calibrated parameters exclusively for call options generates higher prices at the same strike.

2. Fig. 6.6b Repricing put options using parameters from the Mixed calibration results in lower prices than the actual market ones at the same strike price.

3. Fig. 6.6c Repricing Otko options with parameters from the mixed calibration yields results of less straightforward interpretation. The Kou model performs consistently, providing lower prices for all OTKO contracts. Meanwhile, the Merton model and Variance Gamma process exhibit a similar pattern for OTKO contracts with both strike prices different from 0; they price these contracts lower than the originals. However, they tend to overvalue the first two contracts with $K2 = 0$.

6.3 Results Analysis

This section focuses on the collection and analysis of calibration results. Due to the extensive number of stocks analyzed, we'll focus on the most significant findings. Notably, these patterns often replicate across different Lévy processes.

In Figure 6.7, we present the calibrated parameter sets for the three models. A notable observation is the variation in parameter values obtained from calibration on different sets of options.

Specifically, when considering the implied volatility (σ), which is common to all examined Lévy processes, a consistent trend emerges. Calibration on call options consistently results in a significantly lower σ compared to calibration on put options across all analyzed stocks. However, the interpretation becomes more intricate when examining calibration results for OTKO options and the mixed set. For instance, when analyzing TSLA stock, the σ derived from the mixed calibration is notably higher than that from the OTKO calibration. In contrast, AAPL consistently exhibits a lower σ from the mixed calibration compared to the OTKO one.

Another parameter of interest is λ, representing the frequency of jumps per year in both Merton and Kou Jump Diffusion models. Calibrations on OTKO options consistently yield higher λ values across all analyzed stocks, which aligns with the nature of OTKO options, known for pricing significant downward movements in stock prices.

Additionally, when examining parameters related to the Variance Gamma process, specifically μ_+ and μ_-, representing the mean of upward and downward jumps, respectively, an interesting pattern emerges. Calibration on OTKO options consistently results in higher values for μ_- compared to μ_+, indicating that, on average,

downward jumps are larger than upward jumps.

Similar observations hold when analyzing jump parameters for the MJD and KJD models. In both cases, parameters such as the mean (m) of jumps in MJD and η_1 and η_2 in KJD demonstrate that, on average, there are bigger negative movements than positive ones.

We can also conduct a general comparison among the parameters obtained for each stock across all processes. It becomes apparent that the parameters calibrated for TSLA options tend to be higher than the other two models in most cases. This trend is particularly noticeable when considering volatility (σ), which consistently yields the highest values. The reason for this trend is not clear nor readily derivable, as the calibration of option parameters is influenced by multiple factors, including historical volatility, the availability of options with different characteristics, and specific stock market dynamics.

As discussed earlier, the diversity of the companies considered, can lead to differences in investor expectations and, consequently, parameter estimations when it relies on option prices. However, the exact reasons for these differences can vary depending on the specific circumstances of each company and the market context, rendering a concise explanation complex.

Overall, it should be noted that the Double Exponential Jump Diffusion model represents an improvement over the Gaussian Jump Diffusion model, and this distinction becomes evident in the pricing results. Furthermore, the Variance Gamma model, despite relying on three parameters, demonstrates its ability to capture the nuances of option pricing, making it an optimal choice for pricing derivatives. These differences will be explored in greater detail in Chapter 7.

Figures 6.8, 6.9, and 6.10, later in this section, provide a more in-depth analysis,

illustrating the Monte-Carlo simulations (Algorithms 2, 3, 4, 5) with all the sets of parameters calibrated, for a total of $N = 5000$ paths in 252 days.

All sets of parameters calibrated

OPTION TYPE	STOCK	Merton Jump Diffusion				Kou Jump Diffusion				
		σ	λ	m	δ	σ	λ	p	η1	η2
CALLS	AAPL	25.48%	0.240	-0.122	0.197	24.41%	0.655	0.373	8.255	9.937
	NVDA	25.67%	1.011	-0.078	0.233	31.26%	0.738	0.412	8.151	9.926
	TSLA	30.62%	0.997	0.054	0.232	35.47%	0.924	0.569	7.842	10.034
PUTS	AAPL	32.07%	0.577	-0.011	0.289	34.14%	0.770	0.387	5.428	9.927
	NVDA	30.52%	1.216	-0.083	0.298	40.80%	0.806	0.264	5.968	8.966
	TSLA	39.96%	1.211	-0.085	0.446	51.72%	1.261	0.427	8.336	7.906
OTKO	AAPL	32.54%	1.912	-0.056	0.203	34.01%	3.400	0.134	11.283	9.073
	NVDA	33.12%	1.349	-0.122	0.183	30.19%	3.469	0.151	10.300	9.073
	TSLA	33.52%	2.025	-0.097	0.181	31.12%	4.718	0.256	11.234	9.463
MIXED	AAPL	15.82%	1.490	-0.024	0.254	19.00%	2.052	0.331	5.511	5.835
	NVDA	31.40%	1.327	-0.127	0.179	27.80%	3.019	0.448	7.335	8.602
	TSLA	41.25%	1.135	-0.147	0.214	44.77%	2.527	0.198	10.559	10.734

(a) MJD and KJD parameters calibrated on option prices

OPTION TYPE	STOCK	Variance Gamma1			Variance Gamma2			
		σ	θ	*V*	μ+	μ-	*V*+	*V*-
CALLS	AAPL	27.81%	0.134	0.181	0.534	0.401	0.052	0.029
	NVDA	34.92%	0.158	0.235	0.594	0.437	0.083	0.045
	TSLA	39.17%	-0.051	0.234	0.547	0.599	0.070	0.084
PUTS	AAPL	38.89%	0.009	0.178	0.656	0.647	0.077	0.075
	NVDA	44.76%	0.116	0.189	0.788	0.672	0.117	0.085
	TSLA	61.45%	0.159	0.357	0.811	0.652	0.235	0.152
OTKO	AAPL	37.32%	-0.118	0.252	0.439	0.584	0.049	0.087
	NVDA	37.90%	-0.085	0.271	0.470	0.588	0.056	0.087
	TSLA	45.59%	-0.198	0.134	0.789	0.987	0.083	0.130
MIXED	AAPL	33.77%	-0.048	0.435	0.339	0.387	0.050	0.065
	NVDA	39.77%	-0.013	0.318	0.493	0.506	0.077	0.081
	TSLA	51.10%	0.164	0.342	0.705	0.541	0.170	0.100

(b) VG1 (Time changed Brownian motion) and VG2 (Difference of Gammas) parameters calibrated on option prices

Figure 6.7: Overview of all the models parameters found by calibration on the different sets of options

Monte-Carlo simulations

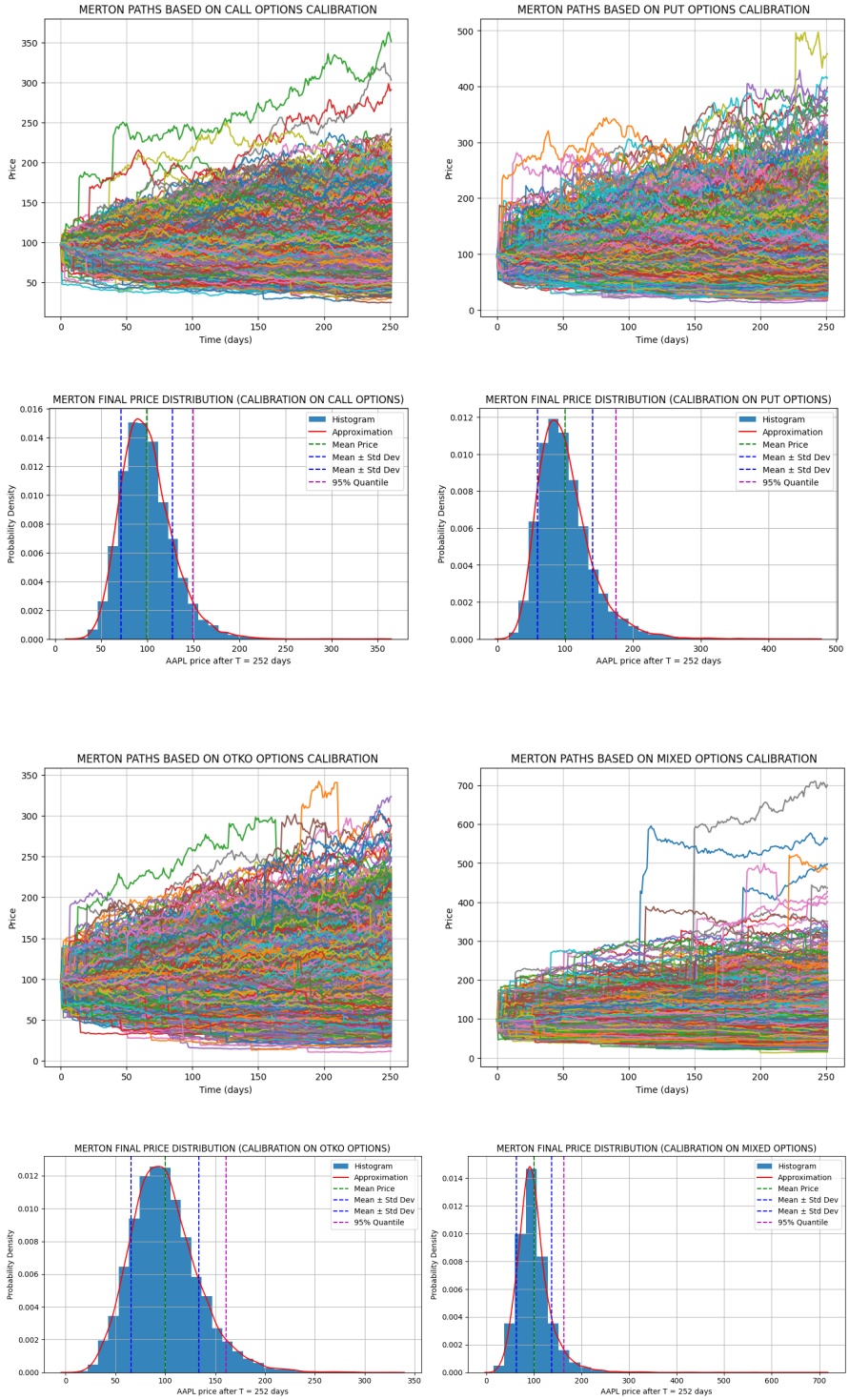


Figure 6.8: Monte-Carlo simulation of paths generated with the Merton Jump Diffusion model with the four set of parameters calibrated for AAPL stock

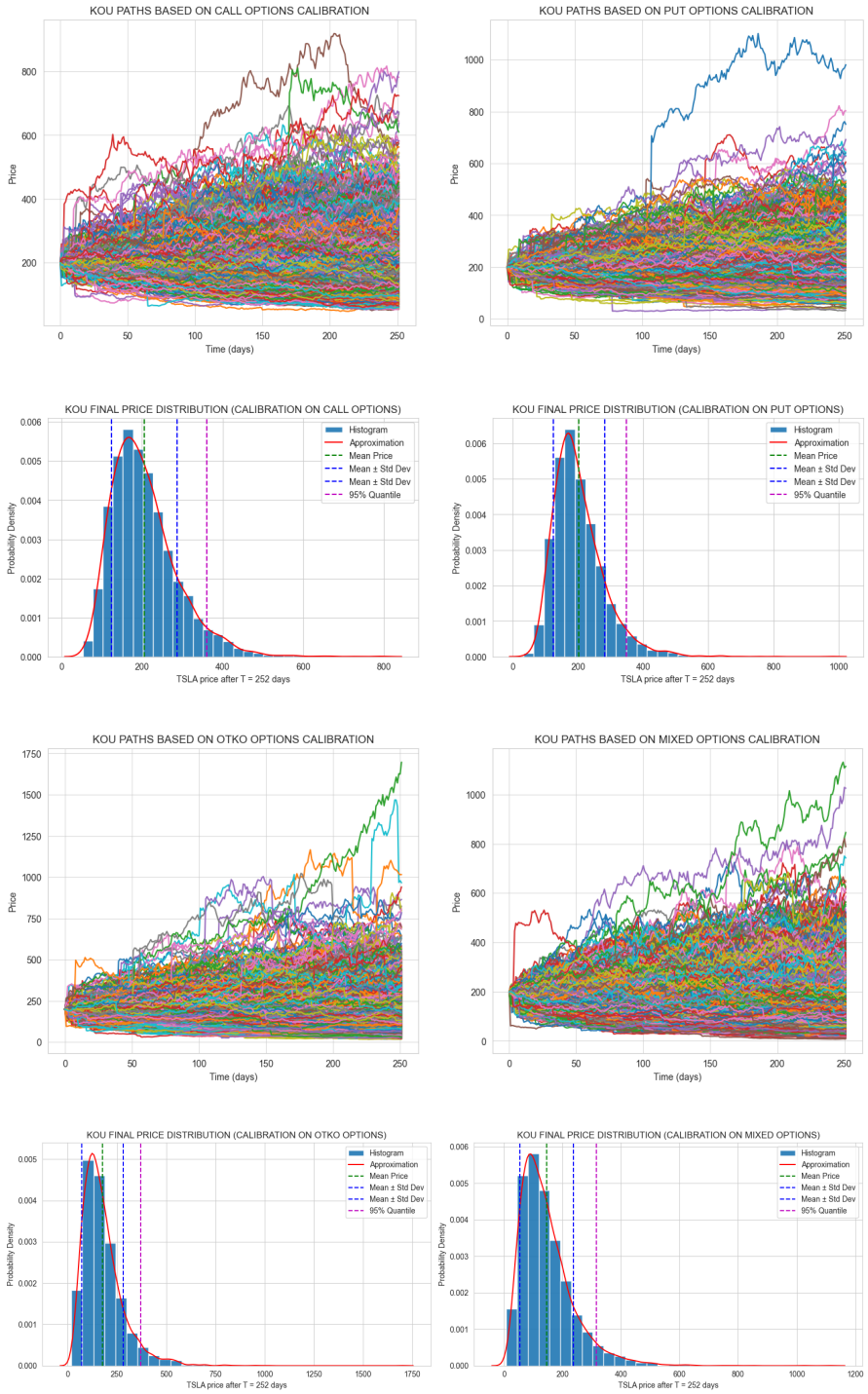


Figure 6.9: Monte-Carlo simulation of paths generated with the Kou Jump Diffusion model with the four set of parameters calibrated for TSLA stock

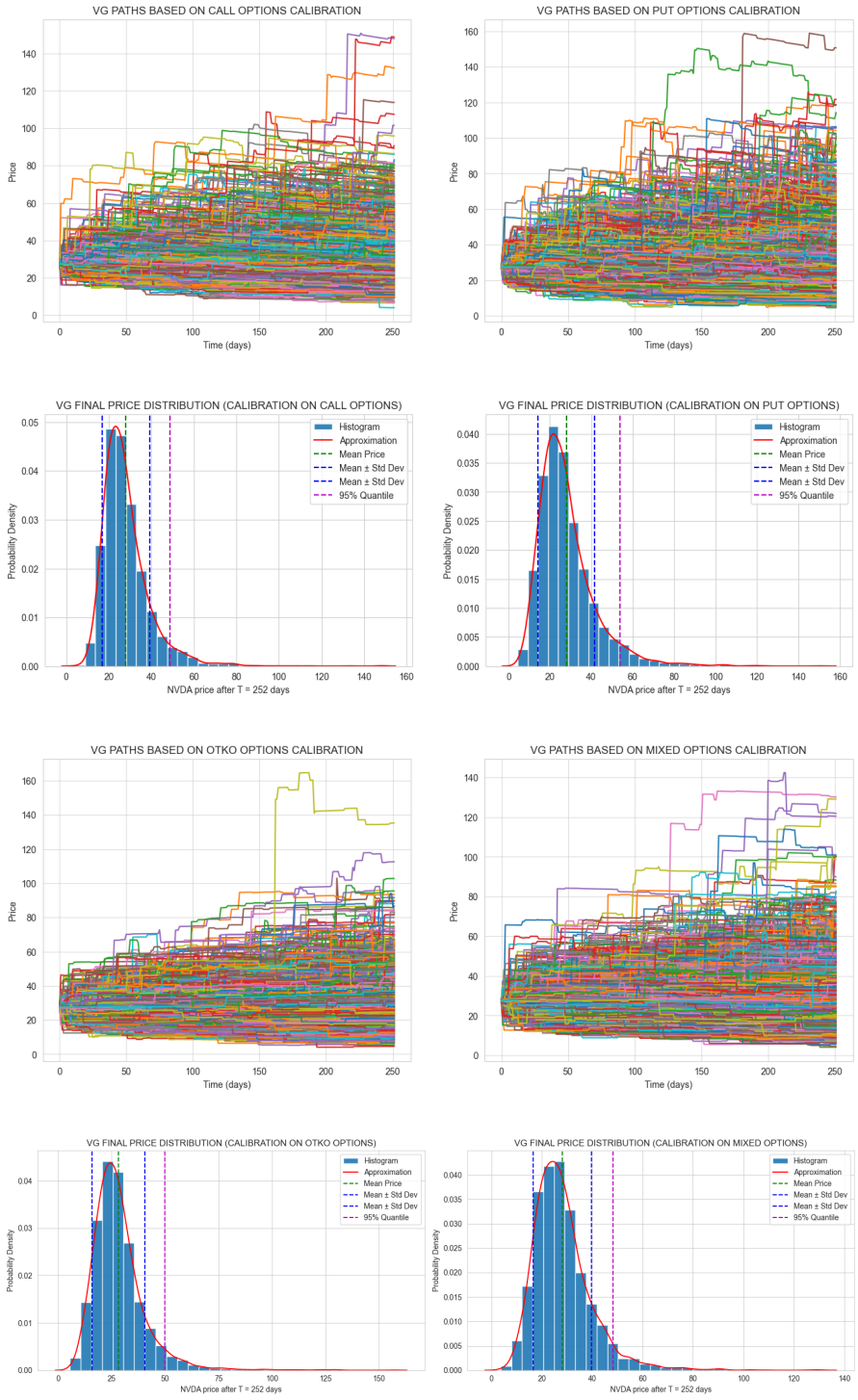


Figure 6.10: Monte-Carlo simulation of paths generated with the Variance Gamma model with the four set of parameters calibrated for NVDA stock

Let us now highlight the key observations arising from the Monte Carlo simulations depicted in the preceding figures.

1. The distributions of the final price from simulations (i.e., the price at $t = 252$) can provide valuable insights in the analysis of Monte Carlo simulated paths. Distributions characterized by a pronounced peak and a narrow standard deviation display a substantial portion of paths that quickly approach zero. In addition to this feature, there are also exceptionally high extreme values (i.e., observing the 95-th quantile), indicating paths that attain very high prices. (see 6.8).

2. The presence of extreme values on the right side is more pronounced in simulations conducted with parameters calibrated for the OTKO options and the mixed set. This further confirms our expectations, as previously discussed at the beginning of this Chapter, Section 6.2. In general, the inclusion of OTKO options during calibration results in parameter values that exhibit a higher degree of risk-taking, leading to stochastic paths that either sharply surge or precipitously decline.

3. In conclusion, all three jump models analyzed effectively capture significant market fluctuations. Both the two Jump Diffusion models and the Pure Jump model provide a solid foundation for pricing margin loans. Their differences in this regard will be further examined in the subsequent chapter.

To conclude our analysis, we present a summary of the computational cost associated with each calibration in terms of time.

OPTION TYPE	STOCK	Computational time (sec)		
		MJD	KJD	VG
CALLS	AAPL	81.89	4719.26	222.53
	NVDA	58.06	5356.43	118.57
	TSLA	90.19	5931.02	403.77
	Average	76.71	5335.57	248.29
PUTS	AAPL	103.45	4618.03	416.81
	NVDA	97.08	4263.89	204.41
	TSLA	115.27	4443.06	335.10
	Average	105.27	4441.66	318.77
OTKO	AAPL	3.21	0.44	0.30
	NVDA	2.81	0.29	0.17
	TSLA	3.39	0.40	0.19
	Average	3.14	0.38	0.22
MIXED	AAPL	153.21	10850.26	1483.18
	NVDA	155.58	10810.28	1050.16
	TSLA	195.45	11708.42	2674.14
	Average	168.08	11122.99	1735.83

Figure 6.11: Summary of time required by each calibration procedure

Figure 6.11 illustrates the time required for each calibration process. Without a doubt, the Kou Jump Diffusion model, when calibrated with vanilla options, stands out as the most time-consuming. This is primarily due to the involvement of a set of recursive functions, detailed in [14] Appendix B, which are used to compute the sum of two double exponential random variables. As a result this cost carries over to the calibration based on the mixed set of options, requiring nearly 3 hours. In contrast, the calibration involving One-Touch Knock-Out Daily Cliquet Options, thanks to its computationally straightforward approximate formula, requires minimal time.

Chapter 7

Margin Loans pricing

In light of the features of margin loans outlined in Section 2.2, we are now prepared to simulate some contracts of this nature on the stocks we introduced in Chapter 6. We have devised three distinct margin loans and have evaluated their outcomes on each of the aforementioned stocks. In this chapter, we delve into the actual pricing of these financial instruments. Specifically, we will simulate Monte Carlo paths using the parameters calibrated in Chapter 6, employing the sets of options Otko Daily Cliquets and the mixed set, which includes both European vanilla and Otko options. For each path, we will assess the payoff of the specific margin loan contract and evaluate its average loss. This loss will enable us to determine the effective financing cost, or in other words, the interest rates. It's worth noting that the results obtained will be highly influenced by the choice of stochastic process. In the next section, we will provide a detailed overview of the characteristics that define a margin loan, as well as the algorithm used to estimate the loss on each contract.

7.1 Contract features and algorithm

The essential features required for a thorough analysis of our financial instrument are as follows:

1. Date and Time Period: The margin loan contract was initiated on January 20, 2016, with a duration spanning 3 years. This date corresponds to the trading date of options, taken into account for the calibration scope.

2. Loan Amount: The loan amount stands at €1,000,000, representing the funds borrowed for the contract.

3. Loan-to-Value Ratio (LTV): The LTV ratio indicates that the loan is secured by collateral worth $1/LTV\%$ of the loan amount.

4. Collateral: The collateral amount can be determined using the equation $Collateral = Loan/LTV$.

5. Margin Call Threshold: The margin call threshold is the critical level that triggers the reintegration process. When the value of the collateral falls below $(1 - \text{Margin Call Threshold}) \times Loan$, a margin call is initiated. The borrower is required to add a specific number of shares to restore the LTV ratio to its original value.

6. Closing Price: The closing price of the stock on the contract's start date, denoted as $S(0)$. This value is crucial for calculating the initial number of shares required as collateral at the beginning of the financing arrangement.

7. Initial Number of Shares as Collateral: This number can be computed directly based on the stock price at $t = 0$ using the formula: $\#\text{of shares} =$

$Collateral/S(0)$.

8. Max Collateral Shares Available: This figure represents the upper limit of shares that can be utilized as collateral throughout the loan period. It is important to note that this number has illustrative purposes, since the company may not have an unlimited quantity of shares available as collateral for the lender. Therefore, a maximum limit was set at twice the initial value for exercise purposes.

Algorithm 11 outlines the procedure for evaluating Monte Carlo simulation paths for a specific given based on a specific Lévy process. The overall process iterates through each path and assesses, day by day, whether a margin call has occurred, following the rules outlined above. At the conclusion of each path, the total loss for that realization is calculated. This total loss will be 0 if the final price S_T multiplied by the total number of shares requested remains above the loan amount. Otherwise, the absolute value of the loss is computed.

Algorithm 11 Margin Loan Price Calculation

Input: Paths of stock prices $S_i(t)$, loan amount F, loan-to-value ltv, maximum allowable shares max_shares, margin call trigger $trigger$, # of simulation paths N, loan period T;

Initialization: Collateral: $C = F/ltv$, # of shares required currently: $shares = C/S[0]$, time-steps: $days = 252 \cdot T$;

for each i from 0 to N **do**

  for each day from 0 to $days - 1$ **do**

    $price \leftarrow S_i[day]$ ▷ Get current price

    if $shares < max\_shares$ **then** ▷ Reintegration allowed

      if $(price \cdot shares_i) \leq ((1 \text{ - } trigger) \cdot F)$ **then**

        $shares\_needed = C/price$ ▷ Add shares for reintegration

        if $shares\_needed \geq max\_shares$ **then**

          Set $shares_i$ to max_shares ▷ Max availability reached

        else

          Set $shares_i \leftarrow shares\_needed$

        end if

      else ▷ No big downward jumps encountered

        Continue.

      end if

    end if

  end for

  $S[days]$ the price at the end of the simulation path.

  Calculate $final\_value_i \leftarrow S_i[days] \times shares_i$.

  if $final\_value_i \geq F$ **then**

    Set $loss_i$ to 0.

  else

    Set $loss_i$ to $F - final\_value_i$.

  end if

end for

return $loss$ array

7.1.1 Some examples

To gain insights into the behavior of margin loans, we conducted a series of tests with the same loan amount but varying loan-to-value ratios (LTVs), resulting in different collateral and the number of shares required. The duration of each test remained consistent at three years. We applied each of these tests to the three selected stocks: NVDA, TSLA, and AAPL. In this section, we will present the results concerning total losses for each model, evaluating the outcomes of three LTV ratios for each of our stock analyzed. The Monte Carlo simulations to estimate total losses and calculate the resulting interest rate were conducted by generating 20,000 paths over a period of $T = 3 \cdot \text{days}$, where $days = 252$ represents the number of financial days in a year, as specified in [27]. We chose to adopt a Margin Call Threshold of 0.07, meaning that the threshold yield R_t that triggers the margin call is set at $th = 0.93$. Therefore, collateral replenishment is required when the price of the respective stock experiences a daily return of -7All these considerations apply to the three different LTV ratios (LTV = 0.85, LTV = 0.7, and LTV = 0.75) mentioned earlier. These ratios represent a range from the lowest amount of collateral required to the highest. Comprehensive results will be summarized in Section 7.2, where we will show the margin loans pricing.

Margin Loan Contracts for Stock: AAPL

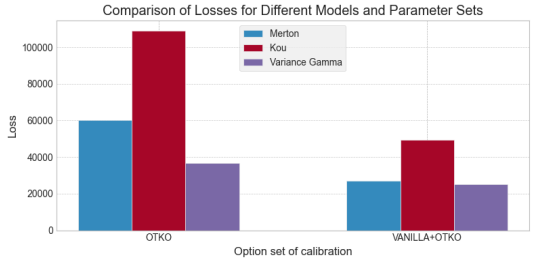
```
> DATE: 2016-01-20
> TIME PERIOD: 3 years

> LOAN: 1000000€
> LOAN-TO-VALUE ratio: 0.85
> COLLATERAL: 1176470.59€
> MARGIN CALL THRESHOLD: -7.0%

> AAPL CLOSING PRICE: 24.17

> INITIAL NUM OF SHARES AS COLLATERAL: 48684.9
> MAX COLLATERAL SHARES AVAILABLE:  97369.8
```

(a) AAPL Contract, LTV = 0.85



(b) AAPL total estimated loss comparison

Figure 7.1: Margin loan contract with LTV ratio of 0.85, to Apple Inc. starting on January 20, 2016. Overview of details and losses.

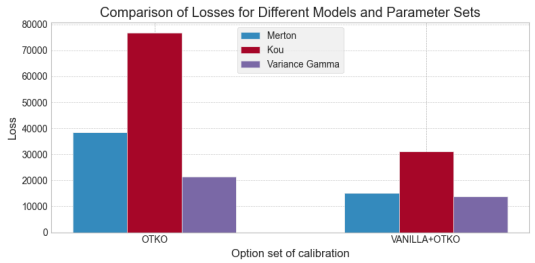
```
> DATE: 2016-01-20
> TIME PERIOD: 3 years

> LOAN: 1000000€
> LOAN-TO-VALUE ratio: 0.7
> COLLATERAL: 1428571.43€
> MARGIN CALL THRESHOLD: -7.0%

> AAPL CLOSING PRICE: 24.17

> INITIAL NUM OF SHARES AS COLLATERAL: 59117.38
> MAX COLLATERAL SHARES AVAILABLE:  118234.76
```

(a) AAPL Contract, LTV = 0.70



(b) AAPL total estimated loss comparison

Figure 7.2: Margin loan contract with LTV ratio of 0.70, to Apple Inc. starting on January 20, 2016. Overview of details and losses.

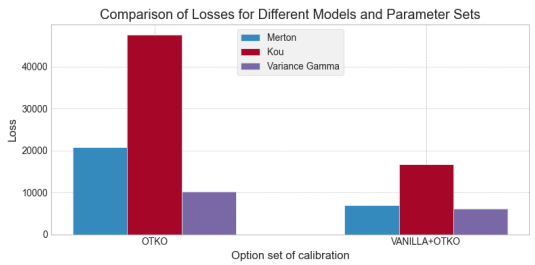
```
> DATE: 2016-01-20
> TIME PERIOD: 3 years

> LOAN: 1000000€
> LOAN-TO-VALUE ratio: 0.55
> COLLATERAL: 1818181.82€
> MARGIN CALL THRESHOLD: -7.0%

> AAPL CLOSING PRICE: 24.17

> INITIAL NUM OF SHARES AS COLLATERAL: 75240.3
> MAX COLLATERAL SHARES AVAILABLE:  150480.6
```

(a) AAPL Contract, LTV = 0.55



(b) AAPL total estimated loss comparison

Figure 7.3: Margin loan contract with LTV ratio of 0.55, to Apple Inc. starting on January 20, 2016. Overview of details and losses.

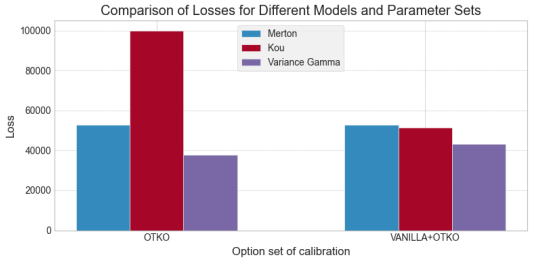
Margin Loan Contracts for Stock: NVDA

```
> DATE: 2016-01-20
> TIME PERIOD: 3 years

> LOAN: 1000000€
> LOAN-TO-VALUE ratio: 0.85
> COLLATERAL: 1176470.59€
> MARGIN CALL THRESHOLD: -7.0%

> NVDA CLOSING PRICE: 6.83

> INITIAL NUM OF SHARES AS COLLATERAL: 172187.43
> MAX COLLATERAL SHARES AVAILABLE:  344374.86
```



(a) NVDA Contract, LTV = 0.85 **(b)** NVDA total estimated loss comparison

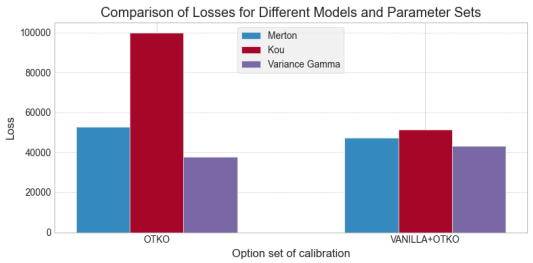
Figure 7.4: Margin loan contract with LTV ratio of 0.85, to NVIDIA Corp. starting on January 20, 2016. Overview of details and losses.

```
> DATE: 2016-01-20
> TIME PERIOD: 3 years

> LOAN: 1000000€
> LOAN-TO-VALUE ratio: 0.55
> COLLATERAL: 1818181.82€
> MARGIN CALL THRESHOLD: -7.0%

> NVDA CLOSING PRICE: 6.83

> INITIAL NUM OF SHARES AS COLLATERAL: 266107.84
> MAX COLLATERAL SHARES AVAILABLE:  532215.68
```



(a) NVDA Contract, LTV = 0.70 **(b)** NVDA total estimated loss comparison

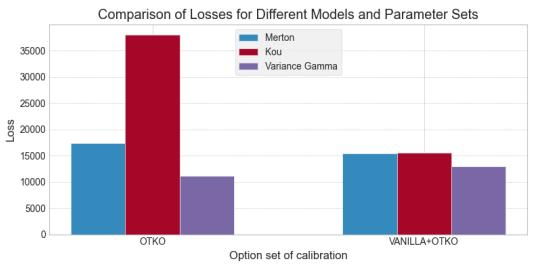
Figure 7.5: Margin loan contract with LTV ratio of 0.70, to NVIDIA Corp. starting on January 20, 2016. Overview of details and losses.

```
> DATE: 2016-01-20
> TIME PERIOD: 3 years

> LOAN: 1000000€
> LOAN-TO-VALUE ratio: 0.55
> COLLATERAL: 1818181.82€
> MARGIN CALL THRESHOLD: -7.0%

> NVDA CLOSING PRICE: 6.83

> INITIAL NUM OF SHARES AS COLLATERAL: 266107.84
> MAX COLLATERAL SHARES AVAILABLE:  532215.68
```



(a) NVDA Contract, LTV = 0.55 **(b)** NVDA total estimated loss comparison

Figure 7.6: Margin loan contract with LTV ratio of 0.55, to NVIDIA Corp. starting on January 20, 2016. Overview of details and losses.

Margin Loan Contracts for Stock: TSLA

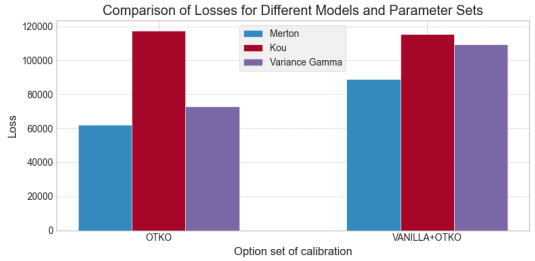
```
> DATE: 2016-01-20
> TIME PERIOD: 3 years

> LOAN: 1000000€
> LOAN-TO-VALUE ratio: 0.85
> COLLATERAL: 1176470.59€
> MARGIN CALL THRESHOLD: -7.0%

> TSLA CLOSING PRICE: 13.65

> INITIAL NUM OF SHARES AS COLLATERAL: 86200.95
> MAX COLLATERAL SHARES AVAILABLE:  172401.9
```

(a) TSLA Contract, LTV = 0.85



(b) TSLA total estimated loss comparison

Figure 7.7: Margin loan contract with LTV ratio of 0.85, to Tesla Inc. starting on January 20, 2016. Overview of details and losses.

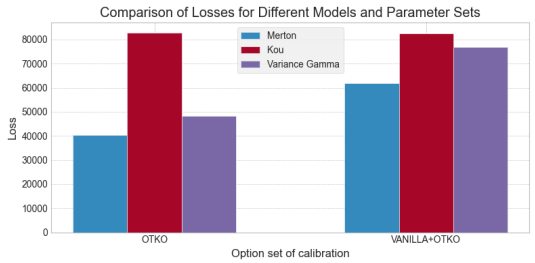
```
> DATE: 2016-01-20
> TIME PERIOD: 3 years

> LOAN: 1000000€
> LOAN-TO-VALUE ratio: 0.7
> COLLATERAL: 1428571.43€
> MARGIN CALL THRESHOLD: -7.0%

> TSLA CLOSING PRICE: 13.65

> INITIAL NUM OF SHARES AS COLLATERAL: 104672.59
> MAX COLLATERAL SHARES AVAILABLE:  209345.18
```

(a) TSLA Contract, LTV = 0.70



(b) TSLA total estimated loss comparison

Figure 7.8: Margin loan contract with LTV ratio of 0.70, to Tesla Inc. starting on January 20, 2016. Overview of details and losses.

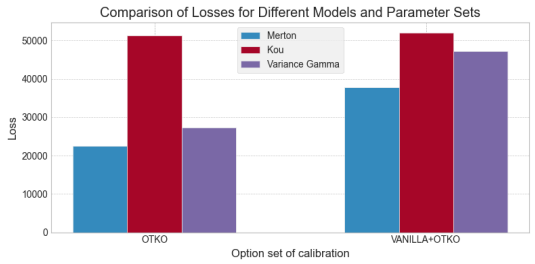
```
> DATE: 2016-01-20
> TIME PERIOD: 3 years

> LOAN: 1000000€
> LOAN-TO-VALUE ratio: 0.55
> COLLATERAL: 1818181.82€
> MARGIN CALL THRESHOLD: -7.0%

> TSLA CLOSING PRICE: 13.65

> INITIAL NUM OF SHARES AS COLLATERAL: 133219.65
> MAX COLLATERAL SHARES AVAILABLE:  266439.3
```

(a) TSLA Contract, LTV = 0.55



(b) TSLA total estimated loss comparison

Figure 7.9: Margin loan contract with LTV ratio of 0.55, to Tesla Inc. starting on January 20, 2016. Overview of details and losses.

Comments

1. The most immediate observation that comes out from the nine contracts simulated, from Fig. 7.1 to Fig. 7.9 is that the Kou model generates higher losses compared to the other two models. These differences are enhanced in all the margin loans proposed, when using simulations run with the Otko calibrated parameters. In the other cases, all the losses generated are roughly comparable.

2. When considering the three tested LTV ratios, a consistent trend emerges: setting the LTV at 0.85 results in the highest losses, while an LTV of 0.7 leads to moderate losses, and an LTV of 0.55 results in the lowest losses. This pattern is not surprising because when the required collateral at the start of the contract is higher, the bank places a greater portion of the loss risk on the initial collateral, thereby protecting itself from the outset.

3. The margin loans tested on AAPL (Fig. 7.1, 7.2, and 7.3) reveal that the KJD model appears to be the most conservative from the bank point of view, resulting in a higher expected loss. Regarding the other two models, MJD evaluates a greater loss when simulated with parameters from the OTKO calibration. However, in the case of the mixed calibration, the two models, MJD and VG perform almost equally everywhere. In any case, the parameters calibrated on the OTKO options consistently yield significantly higher losses.

4. Concerning the NVDA contracts, all the charts shown in Figures 7.4, 7.5, and 7.6 illustrate the same relationship established between the three calibrated Lévy Processes based on the OTKO options: KJD results in almost double the loss compared to MJD, which, in turn, outperforms the VG model. This

trend changes when simulating paths with the parameters calibrated using the mixed set. In this case, the MJD, KJD, and VG models all produce very similar losses overall, altering the final ranking: MJD prevails over KJD in Fig. 7.4 and 7.6.

5. The margin loan contracts offered for TSLA produce results that slightly differ from the other stocks. As shown in Figures 7.7 and 7.9, the losses generated by the KJD model when simulating with the OTKO-calibrated parameters or the mixed set calibrated parameters are approximately the same. However, in Figure 7.8, a different trend is observed, similar to the other cases. The most significant difference is observed in the MJD and VG models. Not only do the simulations using parameters estimated with the mixed set of options produce considerably higher losses than the OTKO ones, but in this case, the VG model consistently generates higher losses than MJD in all the provided charts.

7.2 Pricing results

The formula for calculating the effective interest rate based on the initial loan and losses is as follows:

$$\text{Effective Interest Rate} = \frac{\text{Total Loss}}{\text{Initial Loan}} \times 100, \tag{7.1}$$

which is expressed in percentage. Detailed calculations and results regarding the estimated cost of the loan for each stock, along with each model run using the two calibrated parameter sets (on Otko Daily Cliquets options and on the mixed

set of options), can be found in Figure 7.10. This figure contains three tables, each summarizing the findings for different scenarios. In the second column of the tables, there can be found the contract details considered for each scenario. Parameters of the contract that vary based on the selected LTV are highlighted in gray.

Comments

Given that the actual interest rate is directly related to the estimated total loss, as demonstrated by Equation 7.1, the findings presented in the tables in Figure 7.10 are in complete alignment with those detailed in Section 7.1.1. Notably, the Kou model consistently yields the highest interest rates. This implies that if a bank chooses to adopt this model for pricing margin loans, it is opting for the most conservative approach — one that offers greater repayment, demanding a significantly higher cost of borrowed capital.

Moreover, the relationships between the models, as previously explained in the context of expected losses, remain valid and consistent when assessing interest rates. It's noteworthy that, contract by contract, the Merton and VG models produce highly comparable interest rates, generally exhibiting close alignment. As previously observed, the contract with an LTV of 0.55 allows for relatively lower interest rates across the board, proving to be up to expectations. Indeed, it is a foregone conclusion that the company involved can repay the loan at a lower rate if it can secure more collateral at the principle.

MARGIN LOAN # 1			Merton Jump Diffusion		
			OTKO	MIXED	AVERAGE
		AAPL	6.0183%	2.6898%	4.3541%
LOAN:	1,000,000.00 €	NVDA	5.2950%	4.7327%	5.0138%
LTV:	0.85	TSLA	6.1982%	8.8863%	7.5422%
COLLATERAL:	1,176,470.59 €		Kou Jump Diffusion		
YEARS:	3		OTKO	MIXED	AVERAGE
STOCK VALUE AT t = 0:	24.17 €	AAPL	10.8977%	4.9190%	7.9084%
	6.83 €	NVDA	9.9993%	5.1286%	7.5639%
	13.65 €	TSLA	11.7461%	11.5552%	11.6506%
MAX SHARES:	2x		Variance Gamma		
TRIGGER:	0.07		OTKO	MIXED	AVERAGE
# OF SHARES AT t=0:	48674.83	AAPL	3.6849%	2.5281%	3.1065%
	172250.45	NVDA	3.7605%	4.3182%	4.0394%
	86188.32	TSLA	7.2775%	10.9263%	9.1019%

MARGIN LOAN # 2			Merton Jump Diffusion		
			OTKO	MIXED	AVERAGE
		AAPL	3.8390%	1.5050%	2.6720%
LOAN:	1,000,000.00 €	NVDA	5.2950%	4.7327%	5.0138%
LTV:	0.7	TSLA	4.0473%	6.1826%	5.1150%
COLLATERAL:	1,428,571.43 €		Kou Jump Diffusion		
YEARS:	3		OTKO	MIXED	AVERAGE
STOCK VALUE AT t = 0:	24.17 €	AAPL	7.6798%	3.1028%	5.3913%
	6.83 €	NVDA	9.9993%	5.1286%	7.5639%
	13.65 €	TSLA	8.2804%	8.2471%	9.9966%
MAX SHARES:	2x		Variance Gamma		
TRIGGER:	0.07		OTKO	MIXED	AVERAGE
# OF SHARES AT t=0:	59105.15	AAPL	2.1467%	1.3760%	1.7614%
	209161.26	NVDA	3.7605%	4.3182%	4.0394%
	104657.25	TSLA	4.8121%	7.6928%	6.2525%

MARGIN LOAN # 3			Merton Jump Diffusion		
			OTKO	MIXED	AVERAGE
		AAPL	2.0693%	0.6873%	1.3783%
LOAN:	1,000,000.00 €	NVDA	1.7303%	1.5332%	1.6318%
LTV:	0.55	TSLA	2.2516%	3.7770%	3.0143%
COLLATERAL:	1,818,181.82 €		Kou Jump Diffusion		
YEARS:	3		OTKO	MIXED	AVERAGE
STOCK VALUE AT 0:	24.17 €	AAPL	4.7603%	1.6700%	3.2152%
	6.83 €	NVDA	3.8000%	1.5486%	2.6743%
	13.65 €	TSLA	5.1404%	5.2042%	5.1723%
MAX SHARES:	2x		Variance Gamma		
TRIGGER:	0.07		OTKO	MIXED	AVERAGE
# OF SHARES AT t=0:	75224.73	AAPL	1.0238%	0.6160%	0.8199%
	266205.24	NVDA	1.1051%	1.2883%	1.1967%
	133200.13	TSLA	2.7253%	4.7203%	3.7228%

Figure 7.10: Cost of the three simulated margin loans. Comparison of the interest rates estimated by each stochastic process.

Chapter 8

Conclusions

This thesis embarked on a journey to examine and compare various stochastic models for pricing margin loans, a financial instrument that is sometimes unfamiliar because of the target companies it is aimed at. Our central focus was simulating diverse market scenarios to assess expected losses associated with margin loan issuance. Using Monte Carlo methods, we crafted simulations incorporating factors specific to margin loans, borrowing companies, and broader market trends.

In our pursuit of modeling potential price trajectories of the considered companies, we chose to employ jump models, particularly three Lévy processes. This choice was justified by the unique characteristics of margin loans, which necessitated the consideration of extreme price drop scenarios. Our choice of the Merton and Kou Jump Diffusion models and the Pure Jump Variance Gamma model proved to be effective in accurately modeling extreme events. The calibration process of these model, as detailed in Chapter 6, was a critical step in our research. In this endeavor, we used approximated pricing formula tailored for generic gap options to derive the ones for pricing One-Touch Knock-Out Daily Cliquets options, within all the

three models taken into account in our journey.

Our findings revealed intriguing insights into the calibration process. Solely relying on plain vanilla options resulted in less volatile simulated scenarios with significantly lower values in the tail of the final price distribution. In contrast, utilizing OTKO Daily Cliquets introduced more aggressive jump parameters in terms of intensity and frequency. Therefore, we explored also a mixed calibration approach, incorporating both vanilla and OTKO options, to achieve a balanced representation.

Comparing the results, we found that the parameters derived from Otko options and the mixed set were most suitable for our margin loan pricing work. In the final chapter, we evaluated different margin loan contracts and calculated the applicable interest rates for each financial model. Notably, the Kou model provided the highest estimates of expected losses, while the other two models, although sometimes diverging from Kou's estimates, remained consistent with each other.

In conclusion, this thesis significantly advances the field of margin loan pricing, extending its applicability beyond the specific stocks analyzed. Future research may encompass the analysis of different companies, calibration of models using associated options, and further exploration of gain and loss distributions. Additionally, implementing Value-at-Risk assessments can provide deeper insights into margin loans and their associated risks.

The methodologies and insights generated through this research have the potential to inform and shape risk assessment and management practices in the financial industry. By broadening the scope of analysis and delving into more intricate metrics, future studies can contribute to a more comprehensive understanding of margin loans, ultimately facilitating a more informed approach to their risk management.

Bibliography

[1] Investopedia. *Loan - Definition*. 2023. URL: https://www.investopedia.com/terms/l/loan.asp#toc-why-are-loans-used (cit. on pp. 4, 5).

[2] *Types of Credit: Definitions & How They Impact Credit Score | TIME Stamped*. URL: https://time.com/personal-finance/article/types-of-credit/ (cit. on p. 5).

[3] Wikipedia. *Loans*. 2023. URL: https://en.wikipedia.org/wiki/Loan (cit. on p. 5).

[4] *Loan Structure Definition - Financial Edge*. URL: https://www.fe.training/free-resources/credit/loan-structure/ (cit. on p. 6).

[5] *Simple vs. Compounding Interest: Definitions and Formulas*. URL: https://www.investopedia.com/articles/investing/020614/learn-simple-and-compound-interest.asp (cit. on p. 8).

[6] *LTV (Loan-to-Value) - Overview, Calculating, Collateral*. URL: https://corporatefinanceinstitute.com/resources/commercial-lending/loan-to-value-ratio/ (cit. on p. 9).

[7] *Loan Payment Calculator*. URL: https://www.omnicalculator.com/finance/loan-payment (cit. on p. 10).

[8] White and Case LLP. *NAVs meet margin loans: The rise in single asset financings*. 2018 (cit. on pp. 11, 13).

[9] Blackridge Research. *What Are Special Purpose Vehicle (SPV) and Special Purpose Entity (SPE) Functions?* 2020. URL: https://www.blackridgeresearch.com/blog/what-are-special-purpose-vehicle-spv-and-special-purpose-entity-spe-functions (cit. on p. 12).

[10] *Over-Collateralization (OC): Definition, Benefits, and Examples*. (Accessed on 09/25/2023). URL: https://www.investopedia.com/terms/o/overcollateralization.asp#:~:text=Over%2Dcollateralization%20(OC)%20is,than%20the%20amount%20being%20borrowed. (cit. on p. 13).

[11] Fischer Black and Myron Scholes. «The Pricing of Options and Corporate Liabilities». In: *Journal of Political Economy* 81.3 (1973), pp. 637–654. ISSN: 00223808, 1537534X. URL: http://www.jstor.org/stable/1831029 (visited on 08/21/2023) (cit. on p. 15).

[12] J. Hull. *Options, Futures, and Other Derivatives*. Pearson, 2015. URL: https://books.google.it/books?id=t6CSAgAAQBAJ (cit. on pp. 16, 42, 45, 47, 64).

[13] Robert C. Merton. «Option pricing when underlying stock returns are discontinuous». In: *Journal of Financial Economics* 3.1 (1976), pp. 125–144. ISSN: 0304-405X. DOI: https://doi.org/10.1016/0304-405X(76)90022-2. URL: https://www.sciencedirect.com/science/article/pii/0304405X76900222 (cit. on pp. 20, 21, 47).

[14] S. G. Kou. «A Jump-Diffusion Model for Option Pricing». In: *Management Science* 48.8 (2002), pp. 1086–1101. DOI: 10.1287/mnsc.48.8.1086.166.

URL: https://doi.org/10.1287/mnsc.48.8.1086.166 (cit. on pp. 20, 23, 48, 49, 88).

[15] Dilip B. Madan and Eugene Seneta. «The Variance Gamma (V.G.) Model for Share Market Returns». In: *The Journal of Business* 63.4 (1990), pp. 511–524. ISSN: 00219398, 15375374. URL: http://www.jstor.org/stable/2353303 (visited on 05/05/2023) (cit. on pp. 26, 49).

[16] Dilip B. Madan, Peter P. Carr, and Eric C. Chang. «The Variance Gamma Process and Option Pricing». In: *Review of Finance* 2.1 (Apr. 1998), pp. 79–105. ISSN: 1572-3097. DOI: 10.1023/A:1009703431535. eprint: https://academic.oup.com/rof/article-pdf/2/1/79/26315662/2-1-79.pdf. URL: https://doi.org/10.1023/A:1009703431535 (cit. on pp. 26, 49, 50).

[17] Dilip Madan. «Purely Discontinuous Asset Price Processes». In: (Oct. 2001) (cit. on p. 27).

[18] Peter Carr, Hélyette Geman, Dilip B. Madan, and Marc Yor. «The Fine Structure of Asset Returns: An Empirical Investigation». In: *The Journal of Business* 75.2 (2002), pp. 305–332. ISSN: 00219398, 15375374. URL: http://www.jstor.org/stable/10.1086/338705 (visited on 05/05/2023) (cit. on p. 29).

[19] Michael C. Fu. «Variance-Gamma and Monte Carlo». In: 2007 (cit. on pp. 29, 36).

[20] Paul Glasserman. *Monte Carlo methods in financial engineering*. New York: Springer, 2004. ISBN: 0387004513 9780387004518 1441918221 9781441918222. URL: http://www.amazon.com/Financial-Engineering-Stochastic-

Modelling-Probability/dp/0387004513/ref=pd_sim_b_68?ie=UTF8&refRID=1AN8JXSDGMEV2RPHFC2A (cit. on p. 51).

[21] Peter Tankov. «Pricing and Hedging Gap Risk». In: *SSRN Electronic Journal* (2008). DOI: 10.2139/ssrn.1263352. URL: https://doi.org/10.2139/ssrn.1263352 (cit. on pp. 55, 56).

[22] Peter Tankov. *Financial Modelling with Jump Processes*. Chapman and Hall/CRC, Dec. 2003. DOI: 10.1201/9780203485217. URL: https://doi.org/10.1201%5C%2F9780203485217 (cit. on p. 58).

[23] Filo Fiorani. «Option Pricing under the Variance Gamma Process». In: *SSRN Electronic Journal* (2004). DOI: 10.2139/ssrn.1411741. URL: https://doi.org/10.2139/ssrn.1411741 (cit. on p. 59).

[24] *Shop - OptionsDX*. 2023. URL: https://www.optionsdx.com/shop/ (cit. on p. 63).

[25] *Financial-Models-Numerical-Methods/4.2 Volatility smile and model calibration.ipynb*. URL: https://github.com/cantaro86/Financial-Models-Numerical-Methods/blob/master/4.2%20Volatility%20smile%20and%20model%20calibration.ipynb (cit. on p. 67).

[26] Wikipedia. *Trust region*. URL: https://en.wikipedia.org/wiki/Trust_region (cit. on p. 68).

[27] *Trading day - Wikipedia*. URL: https://en.wikipedia.org/wiki/Trading_day (cit. on p. 93).

Appendix A

Python Code

Listing A.1: Black-Scholes Pricer - Class Definition

```
class BS_Pricer:
    def __init__(self, S0, r, q, sigma, ttm, exercise, K):
        self.S0 = S0  # current price
        self.r = r  # interest rate
        self.sigma = sigma  # diffusion coefficient
        self.ttm = ttm  # maturity in years
        self.q = 0  # dividend yield
        self.exercise = None
        self.K = None  # strike

    def BlackScholesPath(self, days, N):
    """ Paths generation """
        S = np.zeros((days, N))
        S[0] = self.S0
        dt = self.ttm / days
        for t in range(1, days):
```

```
17            Z = np.random.normal(size=N)
18            S[t] = S[t - 1] * np.exp((self.r - 0.5 * self.sigma ** 2)
    * dt + self.sigma * np.sqrt(dt) * Z)
19        return S
20
21    def closed_formula_call(self, K):
22        """ closed formula for call options """
23        self.K = K
24        d1 = (np.log(self.S0 / self.K) + (self.r - self.q + self.
    sigma ** 2 / 2) * self.ttm) / ( self.sigma * np.sqrt(self.ttm))
25        d2 = (np.log(self.S0 / self.K) + (self.r - self.q - self.
    sigma ** 2 / 2) * self.ttm) / (self.sigma * np.sqrt(self.ttm))
26        return self.S0 * np.exp(-self.q * self.ttm) * ss.norm.cdf(d1)
    - (self.K * np.exp(-self.r * self.ttm) * ss.norm.cdf(d2))
27
28    def closed_formula_put(self, K):
29        """ closed formula for put options """
30        self.K = K
31        d1 = (np.log(self.S0 / self.K) + (self.r - self.q + self.
    sigma ** 2 / 2) * self.ttm) / (self.sigma * np.sqrt(self.ttm))
32        d2 = (np.log(self.S0 / self.K) + (self.r - self.q - self.
    sigma ** 2 / 2) * self.ttm) / (self.sigma * np.sqrt(self.ttm))
33        return self.K * np.exp(-self.r * self.ttm) * ss.norm.cdf(-d2)
    - self.S0 * np.exp(-self.q * self.ttm) * ss.norm.cdf(-d1)
34
35    def MonteCarlo_Call(self, K, time, days, N):
36        """ MonteCarlo pricing options"""
37        self.K = K
38        payoffs = []
39        SBlackScholes = self.BlackScholesPath(days, N)
```

```
40          paths_at_t = SBlackScholes[time, :]
41          for i in range(len(paths_at_t)):
42              payoffs = self.payoff_vanilla(paths_at_t[i], 'call')
43          return np.mean(payoffs) * np.exp(-self.r * self.ttm)
44
45      def payoff_vanilla(self, St, type_o):
46          """ european put and call """
47          if type_o == 'call':
48              return self.payoff_call(St)
49          elif type_o == 'put':
50              return self.payoff_put(St)
51          else:
52              raise ValueError('Please select "call" or "put" type.')
53      [...]
```

Listing A.2: Merton Pricer - Class Defintion

```
1  class Merton_pricer():
2      def __init__(self, S0, K, ttm, r, q, sigma, lambd, meanJ, stdJ,
       exercise):
3          self.S0 = S0  # current STOCK price
4          self.K = None  # strike
5          self.ttm = ttm  # maturity in years
6          self.q = 0      #dividend yield
7          self.r = r  # interest rate
8          self.sigma = sigma  # diffusion coefficient
9          self.lambd = lambd  # Num of jumps per year
10         self.meanJ = meanJ  # m: Mean of jump size
11         self.stdJ = stdJ  # v: St. dev. of jump size
12         self.exercise = None
```

```
13      def MertonPath(self, days, N):
14      """ Paths Generation """
15          dt = self.ttm / days
16          size = (days, N)
17          SMerton = np.zeros(size)
18          SMerton[0] = self.S0
19          for t in range(1, days):
20              mean = np.exp(self.meanJ + self.stdJ ** 2 / 2)
21              Z = np.random.normal(size=(N,))  # Brownian motion,
    diffusion component
22              Nj = np.random.poisson(lam=self.lambd * dt, size=(N,))
23              J = np.random.normal(self.meanJ, self.stdJ, size=(N,))
24              jump_component = J * Nj
25              drift_component = (self.r - self.lambd * (mean - 1) - 0.5
     * self.sigma ** 2) * dt  # risk-neutral adjustment
26              diffusion_component = self.sigma * np.sqrt(dt) * Z
27              SMerton[t] = SMerton[t - 1] * np.exp(drift_component +
    diffusion_component + jump_component)
28          return SMerton
29
30      def closed_formula_call(self, K):
31      """ closed formula for call (put) options"""
32          self.K = K
33          V = 0
34          mean = np.exp(self.meanJ + self.stdJ ** 2 / 2)
35          for k in range(40):
36              r_k = self.r - self.lambd * (mean - 1) + (k * np.log(mean
    )) / self.ttm
37              sigma_k = np.sqrt(self.sigma ** 2 + (k * self.stdJ ** 2)
    / self.ttm)
```

```
38            k_fact = factorial(k)
39            V += (np.exp(-mean * self.lambd * self.ttm) * np.power(
    mean * self.lambd * self.ttm, k)) / k_fact *          BS_Pricer.
    BlackScholes(type_o='call', S0=self.S0, K=self.K, ttm=self.ttm, r=
    r_k, q=self.q, sigma=sigma_k)
40        return V
41
42    def closed_formula_otko(self, K1, K2):
43    """ approximated formula for otko options"""
44        tol = 1e-6
45        phi1 = ss.norm.cdf(np.log(K1), self.meanJ, self.stdJ)
46        phi2 = ss.norm.cdf(np.log(K2 + tol), self.meanJ, self.stdJ)
47        phi4 = ss.norm.cdf(np.log(K1) - self.stdJ ** 2, self.meanJ,
    self.stdJ)
48        phi5 = ss.norm.cdf(np.log(K2 + tol) - self.stdJ ** 2, self.
    meanJ, self.stdJ)
49        den = self.r + self.lambd * phi1
50        num = (1 - np.exp(-self.ttm * den))
51        Int = self.lambd * (K1 * phi1 - K2 * phi2 - (np.exp(self.
    meanJ + self.stdJ ** 2 / 2) * (phi4 - phi5)))
52        return Int * num / den * 100
53
54    [...]
```

Listing A.3: Kou Pricer - Class Definition

```
class Kou_pricer():
    def __init__(self, S0, K, ttm, r, sigma, lambd, p, eta1, eta2,
    exercise):
        self.S0 = S0  # current STOCK price
        self.K = None  # strike
        self.T = ttm  # maturity in years
        self.r = r  # interest rate
        self.sigma = sigma  # diffusion coefficient (annual
    volatility)
        self.lambd = lambd  # Num of jumps per year
        self.p = p  # p: probability of upward jumps
        self.q = 1 - self.p  # q: probability of downward jumps
        self.eta1 = eta1  # rate of exponential r.v. (mean: 1/eta1)
        self.eta2 = eta2  # rate of exponential r.v. (mean: 1/eta2)
        self.exercise = None

    def KouPath(self, days, N):
        """ Paths generation """
        dt = self.T / days
        size = (days, N)
        SKou = np.zeros(size)
        SKou[0] = self.S0
        for t in range(1, days):
            # find risk-neutral parameters
            zeta = self.q * self.eta2 / (self.eta2 + 1) + self.p *
    self.eta1 / (self.eta1 - 1) - 1
            # Random numbers generation
            Z = np.random.normal(size=(N,))
```

```
26            Nj = np.random.poisson(lam=self.lambd * dt, size=(N,))
27            # Generate jump sizes J
28            U = np.random.uniform(0, 1, size=(N,))  # Generate
    uniform random variables
29            J = np.zeros_like(U)  # Initialize jump sizes
30            for i in range(N):
31                if U[i] >= self.p:
32                    J[i] = (-1/self.eta1) * np.log((1-U[i]) / self.p)
33                else:
34                    J[i] = (1 / self.eta2) * np.log(U[i] / self.q)
35            # Find components
36            jump_component = J * Nj
37            drift_component = (self.r - 0.5 * self.sigma ** 2 - self.
    lambd*zeta) * dt
38            # drift_component = (self.r - 0.5 * self.sigma ** 2) * dt
39            diffusion_component = self.sigma * np.sqrt(dt) * Z
40            # New prices computation
41            SKou[t] = SKou[t - 1] * np.exp(drift_component +
    diffusion_component + jump_component)
42        return SKou
43
44    def closed_formula_call(self, K):
45    """ closed formula for call options"""
46        self.K = K
47        zeta = self.p * self.eta1 / (self.eta1 - 1) + (self.q * self.
    eta2) / (self.eta2 + 1) - 1
48        lambd2 = self.lambd * (zeta + 1)
49        eta1_2 = self.eta1 - 1
50        eta2_2 = self.eta2 + 1
51        p2 = self.p / (1 + zeta) * self.eta1 / (self.eta1 - 1)
```

```
52          vkjd_1 = self.S0 * self.Yfunction(self.r + 1 / 2 * self.sigma
    ** 2 - self.lambd * zeta, self.sigma, lambd2, p2, eta1_2, eta2_2,
    np.log(self.K / self.S0), self.T)
53          vkjd_2 = self.K * np.exp(-self.r * self.T) * self.Yfunction(
    self.r - 1 / 2 * self.sigma ** 2 - self.lambd * zeta, self.sigma,
    self.lambd, self.p, self.eta1, self.eta2, np.log(self.K / self.S0)
    , self.T)
54          return vkjd_1 - vkjd_2
55
56      # Three term recursion
57      def Hh(self, n, x):
58          if n < -1:
59              return 0
60          elif n == -1:
61              return np.exp(-x ** 2 / 2)
62          elif n == 0:
63              return np.sqrt(2 * np.pi) * scs.norm.cdf(-x)
64          else:
65              return (self.Hh(n - 2, x) - x * self.Hh(n - 1, x)) / n
66
67      # Pfunction from KOU 2002 (Appendix B)
68      def P(self, n, k, eta1, eta2, p):
69          q = 1 - p
70          if k < 1 or n < 1:
71              return 0
72          elif n == k:
73              return p ** n
74          else:
75              sum_p = 0; i = k
76              while i <= n - 1:
```

```
77                  sum_p = sum_p + ssp.binom(n - k - 1, i - k) * ssp.
    binom(n, i) * (eta1 / (eta1 + eta2)) ** (i - k) * (eta2 / (eta1 +
    eta2)) ** (n - i) * p ** i * q ** (n - i); i += 1
78          return sum_p
79
80      # Qfunction from KOU 2002 (Appendix B)
81      def Q(self, n, k, eta1, eta2, p):
82          q = 1 - p
83          if k < 1 or n < 1:
84              return 0
85          elif n == k:
86              return q ** n
87          else:
88              sum_q = 0; i = k
89              while i <= n - 1:
90                  sum_q = sum_q + ssp.binom(n - k - 1, i - k) * ssp.
    binom(n, i) * (eta1 / (eta1 + eta2)) ** (n - i) * (eta2 / (eta1 +
    eta2)) ** (i - k) * p ** (n - i) * q ** i
91                  i += 1
92          return sum_q
93
94      def I(self, n, c, alpha, beta, delta):
95          if beta < 0 and alpha < 0:
96              sum_i = 0
97              i = 0
98              while i <= n:
99                  sum_i = sum_i + (beta / alpha) ** (n - i) * self.Hh(i
    , beta * c - delta); i += 1
```

```
100             return -np.exp(alpha * c) / alpha * sum_i - (beta / alpha
    ) ** (n + 1) * (np.sqrt(2 * np.pi) / beta) * np.exp((alpha * delta
     / beta) + (alpha ** 2 / (2 * beta ** 2))) * scs.norm.cdf(beta * c
     - delta - alpha / beta)
101         elif beta > 0 and alpha != 0:
102             sum_i = 0; i = 0
103             while i <= n:
104                 sum_i = sum_i + (beta / alpha) ** (n - i) * self.Hh(i
    , beta * c - delta); i += 1
105             return -np.exp(alpha * c) / alpha * sum_i + (beta / alpha
    ) ** (n + 1) * (np.sqrt(2 * np.pi) / beta) * np.exp((alpha * delta
     / beta) + (alpha ** 2 / (2 * beta ** 2))) * scs.norm.cdf(-beta *
    c + delta + alpha / beta)
106         else:
107             return 0
108
109     def Pi(self, n, lambd):
110         return np.exp(-lambd * self.T) * (lambd * self.T) ** n / math
    .factorial(n)
111
112     def Yfunction(self, mu, sigma, lambd, p, eta1, eta2, a, T):
113         bound = 10; sump1 = 0; sumq1 = 0
114         for n in range(1, bound + 1):
115             sump1_n = 0
116             sumq1_n = 0
117             for k in range(1, n + 1):
118                 sump2_k = self.P(n, k, eta1, eta2, p) * (sigma * np.
    sqrt(T) * eta1) ** k * self.I(k - 1, a - mu * T, -eta1, -1 / (
    sigma * np.sqrt(T)), -sigma * eta1 * np.sqrt(T))
```

```
119                 sumq2_k = self.Q(n, k, eta1, eta2, p) * (sigma * np.
    sqrt(T) * eta2) ** k * self.I(k - 1, a - mu * T, eta2, 1 / (sigma
    * np.sqrt(T)), -sigma * eta2 * np.sqrt(T))
120                 sump1_n += sump2_k; sumq1_n += sumq2_k
121             sump1 += self.Pi(n, lambd) * sump1_n
122             sumq1 += self.Pi(n, lambd) * sumq1_n
123         Y1 = np.exp((sigma * eta1) ** 2 * T / 2) / (sigma * np.sqrt(2
     * np.pi * T)) * sump1
124         Y2 = np.exp((sigma * eta2) ** 2 * T / 2) / (sigma * np.sqrt(2
     * np.pi * T)) * sumq1
125         Y3 = self.Pi(0, lambd) * scs.norm.cdf(-(a - mu * T) / (sigma
    * np.sqrt(T)))
126         return Y1 + Y2 + Y3
127
128     def closed_formula_otko(self, K1, K2):
129     """ approximate formula for otko options price"""
130         beta = np.log(K1)
131         phi = self.lambd * self.q * np.exp(beta*self.eta2)
132         den = self.r + phi
133         num = (1 - np.exp(-self.T * den))
134         Int = self.lambd*self.q / (1 + self.eta2) * (K1**(1+self.eta2
    ) - K2**(1+self.eta2))
135         return Int * num / den * 100
136     [...]
```

Listing A.4: Variance Gamma Pricer - Class Definiton

```
1 class VG_pricer():
2     def __init__(self, S0, K, ttm, r, q, sigma, theta, nu, exercise):
3         self.S0 = S0  # current STOCK price
```

```
4           self.K = None  # strike
5           self.ttm = ttm  # maturity in years
6           self.r = r  # interest rate
7           self.q = q  # dividend yield
8           self.sigma = sigma  # diffusion coefficient
9           self.theta = theta  # Drift of gamma process
10          self.nu = nu  # variance of gamma process
11          self.rho = 1/self.nu
12          self.exercise = None
13
14          # Parameters of the difference of gammas representation
15          self.mu_p = 0.5 * np.sqrt(self.theta ** 2 + (2 * self.sigma
    ** 2 / self.nu)) + 0.5 * self.theta  # positive jump mean
16          self.mu_n = 0.5 * np.sqrt(self.theta ** 2 + (2 * self.sigma
    ** 2 / self.nu)) - 0.5 * self.theta  # negative jump mean
17          self.nu_p = np.power(self.mu_p, 2) * self.nu  # positive jump
     variance
18          self.nu_n = np.power(self.mu_n, 2) * self.nu  # negative jump
     variance
19
20      def VarianceGammaPath1(self, days, N):
21      """ Paths Generation as time changed Brownian motion """
22          dt = self.ttm / days
23          size = (days, N)
24          SVarGamma = np.zeros(size)
25          SVarGamma[0] = self.S0
26          omega = np.log(1 - self.theta * self.nu - 0.5 * self.nu *
    self.sigma ** 2) / self.nu
27          for t in range(1, days):
28              Z = np.random.normal(0, 1, size=(N,))
```

```
29            deltaG = ss.gamma.rvs(a=dt / self.nu, scale=self.nu, size
    =(N,))
30            VG = self.theta * deltaG + self.sigma * np.sqrt(deltaG) *
     Z
31            SVarGamma[t] = SVarGamma[t - 1] * np.exp((self.r + omega)
     * dt + VG)
32        return SVarGamma
33
34    def VarianceGammaPath2(self, days, N):
35    """ Paths Generation as the difference of two gammas """
36        dt = self.ttm / days
37        size = (days, N)
38        SVarGamma = np.zeros(size)
39        SVarGamma[0] = self.S0
40        omega = np.log(1 - (self.theta * self.nu) - (self.nu * self.
    sigma ** 2)/2) / self.nu
41        for t in range(1, days):
42            Gamma_p = ss.gamma.rvs(a=dt / self.nu, scale=self.mu_p *
    self.nu, size=(N,))
43            Gamma_n = ss.gamma.rvs(a=dt / self.nu, scale=self.mu_n *
    self.nu, size=(N,))
44            VG = (Gamma_p - Gamma_n)
45            SVarGamma[t] = SVarGamma[t - 1] * np.exp((self.r + omega)
    *dt + VG)
46        return SVarGamma
47
48    def omega(self):  # martingale correction
49        return - np.log(1 - self.theta * self.nu - (self.sigma ** 2 *
     self.nu) / 2) / self.nu
50
```

```
51      def closed_formula_call(self, K):
52          """ closed formula for call options """
53          self.K = K; eps = 1e-6
54          def Psy(a, b, g):
55              f = lambda u: ss.norm.cdf(a / np.sqrt(u) + b * np.sqrt(u)
    ) * np.exp((g - 1) * np.log(u)+eps) * np.exp(
56                  -u) / ssp.gamma(g)
57              result = quad(f, 0, np.inf, epsabs=1e-6, epsrel=1e-6)
58              return result[0]
59
60          xi = - self.theta / self.sigma ** 2
61          s = self.sigma / np.sqrt(1 + ((self.theta / self.sigma) ** 2)
     * (self.nu / 2))
62          alpha = xi * s
63          c1 = self.nu / 2 * (alpha + s) ** 2; c2 = self.nu / 2 * alpha
     ** 2
64          d = 1 / s * (np.log((self.S0 / self.K) + eps) + self.r * self
    .ttm + self.ttm / self.nu * np.log((1 - c1)/(1 - c2)+eps ))
65          call = self.S0 * Psy(d * np.sqrt((1 - c1) / self.nu), (alpha
    + s) * np.sqrt(self.nu / (1 - c1)), self.ttm / self.nu) - self.K *
     np.exp(-self.r * self.ttm) * Psy(d * np.sqrt((1 - c2) / self.nu),
     alpha * np.sqrt(self.nu / (1 - c2)), self.ttm / self.nu)
66          return call
67
68      def closed_formula_otko(self, K1, K2):
69      """ approximate pricing formual for otko options """
70          tol = 1e-6; c = 1 / self.nu
71          G = 1 / (np.sqrt(self.theta ** 2 * self.nu ** 2 / 4 + self.
    sigma ** 2 * self.nu / 2) - self.theta * self.nu/2)
72          phi = -c * ssp.expi(G * np.log(K1))
```

```
73          den = self.r + phi
74          num = 1 - np.exp(-self.ttm * den)
75          Int1 = c / G * (K2 * ssp.expi(G * np.log(K2 + tol)) - K1 *
    ssp.expi(G * np.log(K1)))
76          Int2 = c / (G+1) * (ssp.expi((G + 1) * np.log(K1)) - ssp.expi
    ((G + 1) * np.log(K2 + tol)))
77          return (Int1 + Int2) * num / den * 100
78      [...]
```

Listing A.5: Simulation of Monte-Carlo paths for each model

```
1  """ Paths Simulation """
2  [...]
3  from BSpricer import BS_Pricer
4  from MERTONpricer import Merton_pricer
5  from KOUpricer import Kou_pricer
6  from VGpricer import VG_pricer
7
8  symbol = 'AAPL'  # TSLA, NVDA
9  dates = options.get_expiration_dates(symbol)
10
11 T_str = 'June 21, 2024'
12 puts = options.get_puts(symbol, T_str)
13
14 T_datetime = datetime.strptime(T_str, '%B %d, %Y')
15 ttm = (T_datetime - datetime.now()).days / 365.0  # ttm in float mode
16
17 puts['Time-to-maturity'] = ttm
18
19 # historical volatility
```

```
stock_data = si.get_data(symbol, start_date='31/05/2021', end_date='31/05/2023')
stock_data['Returns'] = stock_data['close'] / stock_data['close'].shift()
stock_data['Log Returns'] = np.log(stock_data['Returns'])
volatility = stock_data['Log Returns'].std() * np.sqrt(252)
print(f'\n{symbol} historical volatility: {round(volatility, 3)}')

# Fix simulation parameters
S0 = si.get_live_price(symbol)  # get live price of stock
T = ttm  # Expiry Date in years
days = 252
paths = 1000
K = option[2]    #Strike price
sigma = 0.153    #volatility
r = 0.03     #risk.free interest rate
q = 0        #dividend yield
size = (days, paths)
exercise = 'european'

# Black Scholes model
BS = BS_Pricer(S0, r, q, sigma, T, exercise, K)
SBlackScholes = BS.BlackScholesPath(days, paths)
BS.plotBSPath(SBlackScholes, symbol)  #Plot all paths

# Merton Jump Diffusion model
lamda = 0.5; jump_mean = 0.05; jump_std = 0.15
Merton = Merton_pricer(S0, K, T, r, q, sigma, lamda, jump_mean, jump_std, exercise)
SMerton = Merton.MertonPath(days, paths)
```

```
47 Merton.plotMertonPath(SMerton, symbol)  # Plot all paths
48
49 # Kou Jump Diffusion model
50 lamda = 1; eta1 = 8; eta2 = 5; p = 0.6;
51 KOU = Kou_pricer(S0, K, T, r, sigma, lamda, p, eta1, eta2, exercise)
52 SKou = KOU.KouPath(days, paths)
53 KOU.plotKouPath(SKou, symbol)   # Plot all paths
54
55 # Variance Gamma model
56 sigma = 0.2; theta = -0.12; nu = 0.05
57 VG = VG_pricer(S0, K, T, r, q, sigma, theta, nu, exercise)
58 SVarGamma = VG.VarianceGammaPath1(days, paths)
59 SVarGamma2 = VG.VarianceGammaPath2(days, paths)
60 method = ['Time changed BM', 'Difference of Gammas']
61 # Plot all paths
62 fig1, axes = plt.subplots(nrows=1, ncols=2, figsize=(16,6))
63 VG.plotVGPath(SVarGamma, symbol, method[0], ax=axes[0])
64 VG.plotVGPath(SVarGamma2, symbol, method[1], ax=axes[1])
65 plt.tight_layout()
66 [...]
67 # Model validation
68 daily_avg = pd.DataFrame(columns=['BlackScholes', 'Merton', 'Kou', '
       VarGamma1', 'VarGamma2'])
69 # Day-by-day mean of the 1000 paths, for each model
70 daily_avg['BlackScholes'] = SBlackScholes.mean(axis=1)
71 daily_avg['Merton'] = SMerton.mean(axis=1)
72 daily_avg['Kou'] = SKou.mean(axis=1)
73 daily_avg['VarGamma1'] = SVarGamma.mean(axis=1)
74 daily_avg['VarGamma2'] = SVarGamma2.mean(axis=1)
75 St = S0 * np.exp(r * T)
```

```
daily_growth_factor = np.exp(r/ days)
St = S0 * np.cumprod(np.full(days, daily_growth_factor))

fig1 = go.Figure()
for column in daily_avg.columns:
    fig1.add_trace(go.Scatter(x=spx_data.loc[0:251, 'Date'], y=
    daily_avg[column], name=column))
fig1.add_trace(go.Scatter(x = spx_data.loc[0:251, 'Date'], y = St,
    name='Risk-free Growth'))

fig1.update_layout(
    xaxis_title='Date',
    yaxis_title='Price',
    title = 'Real vs Simulated SPX paths',
    showlegend=True
)
pio.show(fig1)
```

Listing A.6: Find Implied Volatility from Option Market prices

```
""" Implied volatilty found on Call Options"""
[...]
def implied_volatility(price, S, strike, t, rate, q, type_o, method='
    fsolve', disp=True ):

    def obj_fun(vol):
        return BS.BlackScholes(type_o=type_o, S0=S, K=strike, ttm=t,
    r=rate, q=q, sigma=vol) - price

    def vega(vol):
```

```
9           return BS.vega(S, strike, rate, q, vol, t)
10
11      if method =='fsolve':
12          X0 = [0.01, 0.2, 0.35, 7]          #initial guess points for
    imp.vol.
13          for x_0 in X0:
14              x, _, solved, _ = scpo.fsolve(obj_fun, x_0, full_output=
    True, xtol=1e-8)
15              if solved == 1:
16                  return x[0]
17      if disp:
18          return -1
19
20      IV_market = []
21  for i in range(len(call_prices)):
22      IV_market.append(implied_volatility(call_prices[i], S=S0, strike=
    call_strikes[i], t = T, rate=0.027, q = 0.02, type_o='call',
    method='fsolve'))
23
24  print(f'Implied volatilities of market prices (calls):\nS0 = {S0}')
25  for a,b in zip(call_strikes.tail(6), IV_market[-6:]):
26      print(f'K = {a}, IV = {round(b, 4)}')
27
28  # Plot market implied volatilities w.r.t. moneyness K/S0
29  x = (call_strikes/S0)
30  IV_filtered = [iv for iv in IV_market if iv != -1]
31  x_filtered = [log_m for iv, log_m in zip(IV_market, x) if iv != -1]
32
33  fig, ax1 = plt.subplots(figsize=(6,5))
```

```
ax1.scatter(x, calls['C_IV'], marker='x', label='provided', color='red', s=40)
ax1.plot(x, IV_market, label='BS inversion', alpha = 0.8, color = 'green')
ax1.set_xlabel('Moneyness: K/S0'); ax1.set_ylabel('Implied Volatility'); ax1.set_title('CALLS volatility smile')
ax1.legend(); plt.grid(True, linewidth=0.5); plt.show()
```

Listing A.7: Find the payoff of One-Touch Knock-Out Daily Cliquets Options

```
""" Otko Payoff """
def otko_payoff(R, K1, K2):
payoffs = []
for Rt in R:
    if Rt > K1:
        payoffs.append(0)
    elif K2 < Rt <= K1:
        payoffs.append((K1 - Rt))
    elif Rt <= K2:
        payoffs.append((K1–K2))
return payoffs

R = np.linspace(0,150,100)
fig, axes = plt.subplots(nrows=2, ncols=2, figsize=(20,10))
row = 0
col = 0

for index, contract in otko_spx.iterrows():
    if index != 2:  # Exclude the 2nd row (index 1)
        K1 = contract['K1']
```

```
        K2 = contract['K2']
        axes[row, col].plot(R, otko_payoff(R, K1, K2))
        axes[row, col].set_xlabel('RETURNS * 100')
        axes[row, col].set_ylabel('Payoff')
        axes[row, col].set_title(f'PAYOFF: OTKO {K1} - {K2}')
        if index != 0 and index!= 1:
            axes[row, col].set_ylim(-0.5, 12)
        axes[row, col].grid()
        axes[row, col].axvline(x=K1, color='red', linestyle='--')
        axes[row, col].axvline(x=K2, color='red', linestyle='--')
        col += 1
        if col == 2:
            row += 1
            col = 0
plt.tight_layout()
plt.show()
```

Listing A.8: Call Options calibration

```
"""Calibration of Parameters on Call Options"""
df_call = pd.read_csv('../../data/AAPL/OPT16_AAPL_CALLS_75_135.csv')
calls['C_Midpoint'] = abs(calls['C_BID'] + calls['C_ASK']) / 2
calls['C_Spread'] = calls['C_BID'] - calls['C_ASK']
# Set fixed parameters
q = 0           # dividend yield
r = 0.03        # risk-free interest rate
S0 = calls.iloc[0]['UNDERLYING_LAST']
T = 1           # time-to-maturity
call_strikes = calls['STRIKE']    # array of K for call options
exercise = 'european'
```

```
call_prices = calls['C_Midpoint']
call_spreads = calls['C_Spread']
c_weights = 1/ call_spreads**2

#Initialize objects of Pricer classes
BS = BS_Pricer(S0=S0, r=r, q = q, sigma=sigma, ttm=T, exercise=
    exercise, K=None)
Merton = Merton_pricer(S0=S0, K=None, ttm=T, r=r, q = q, sigma=0.15,
    lambd=0.5, meanJ=-0.1, stdJ=0.1, exercise=exercise)
Kou = Kou_pricer(S0=S0, K=None, ttm=T, r=r, sigma=0.15, lambd=0.5, p
    =0.6, eta1=12, eta2=5, exercise=exercise)
VG = VG_pricer(S0, K=None, ttm=T, r=r, q=q, sigma=0.15, theta=-0.2,
    nu=0.3, exercise=exercise)

#Black-Scholes
x0 = 0.5
bounds = [1e-5, 2]
def cost_function(x, strikes, mkt_prices):
    sigma = x
    BS = BS_Pricer(S0=S0, K = None, ttm=T, r=r, q=0, sigma=sigma,
    exercise=exercise)
    sq_err = np.sum(c_weights* (BS.closed_formula_call(strikes) -
    mkt_prices)**2)
    return sq_err

result_p = scpo.least_squares(cost_function, x0, args=(call_strikes,
    call_prices), bounds=bounds, method = 'trf', verbose=1)
calls_bs_params_aapl = result_p.x[0]
[...]
```

Listing A.9: Merton parameters calibration on Put Options

```
"""Find Put Options prices according to the MJD model and estimate
    the best-fitting parameters to market prices"""
[...]
x0 = [0.4, 0.5, -0.05, 0.1]        #[sigma, lambda, m, v]
bounds = ([1e-3, 1e-2, -2, 0], [1, 5, 2, 2])

def cost_function(x, strikes, mkt_prices):
    sigma, lambd, meanJ, stdJ = x
    M = Merton_pricer(S0, None, T, r, q, sigma, lambd, meanJ, stdJ,
    exercise)
    sq_err = np.sum(p_weights*(M.closed_formula_put(strikes) -
    mkt_prices)**2)
    return sq_err

start = time.time()
mert = scpo.least_squares(cost_function, x0, args=(put_strikes,
    put_prices), bounds=bounds, method='trf', verbose=2)
end = time.time()

mert_params_calibrated = [round(p, 4) for p in mert.x[:4]]

#Reprice Options with parameters found for results validation
days = 252
paths = 1000

put_calib_prices = pd.DataFrame({
    'STRIKE': puts['STRIKE'],    # array of K for put options
    'MKT_BID': puts['C_BID'],
```

```
25      'MKT_MID': puts['C_Midpoint'],
26      'MKT_ASK': puts['C_ASK'],
27  })
28  sigma, lambd, meanJ, stdJ = mert_params_calibrated
29  MertonCAL = Merton_pricer(S0, None, T, r, q, sigma, lambd, meanJ,
        stdJ, exercise)
30  SMerton_CAL = MertonCAL.MertonPath(days, paths)
31  avg_payoffs = []
32  for k in put_strikes:
33      payoffs = []        # stores here the payoff for each path, for a
        specific couple K1-K2
34      for St in SMerton_CAL[-1]:
35          payoffs.append(MertonCAL.payoff_put(k, St))
36      avg_payoffs.append(np.mean(payoffs))
37  merton_mc_prices = np.zeros(len(put_calib_prices))
38  merton_cf_prices = np.zeros(len(put_calib_prices))
39  for index in range(len(put_calib_prices)):
40      merton_mc_prices[index] = np.exp(-r*T)* avg_payoffs[index]
41      merton_cf_prices[index] = MertonCAL.closed_formula_put(
        put_strikes[index])
42  put_calib_prices['MERTON MC'] = merton_mc_prices
43  put_calib_prices['MERTON CF'] = merton_cf_prices
44
45  [...]
```

Listing A.10: Variance Gamma Parameters Calibration on OTKO Options

```
1  """Find OTKO Options prices according to the VG model and estimate
       the best-fitting parameters to market prices"""
2  [...]
```

```
x0 = [0.3612, -0.185, 0.76]  #Initial guess: [sigma, theta, nu]
bounds = ([1e-3, -2, 0], [0.8, 2, 4])

def cost_function(x, strikes, mkt_prices):
    sigma, theta, nu = x
    VG = VG_pricer(S0, None, T, r, q, sigma, theta, nu, exercise)
    sq_err = 0
    for k in range(len(strikes)):
        sq_err += weights[k] * (VG.closed_formula_otko(strikes[k][0],
    strikes[k][1]) - mkt_prices[k]) ** 2
    return sq_err

start = time.time()
vg = scpo.least_squares(cost_function, x0, args=(strikes, mkt_prices)
    , method='trf', bounds=bounds, verbose=1)
end = time.time()

vg_params_calibrated = [round(p, 4) for p in vg.x[:3]]
sigma, theta, nu = vg_params_calibrated
VG_CAL = VG_pricer(S0, None, T, r, q, sigma, theta, nu, exercise)
SVarGamma_cal = VG_CAL.VarianceGammaPath1(days, paths)

#Reprice Options with parameters found for results validation
[...]
calibrated_prices = pd.DataFrame({
    'K1': otko_aapl['K1'],
    'K2': otko_aapl['K2'],
    'MKT_BID (%)': otko_aapl['BID(%)'],
    'MKT_MID (%)': otko_aapl['Midpoint(%)'],
    'MKT_ASK (%)': otko_aapl['ASK(%)']
```

```
})
avg_payoffs = []
for K1, K2 in strikes:
    payoffs = []        # stores here the payoff for each path, for a
    specific couple K1-K2
    for path in SVarGamma_cal.T:
        payoffs.append(VG_CAL.payoff_otko(path, K1, K2))
    avg_payoffs.append(np.mean(payoffs))

vg_mc_prices = np.zeros(len(calibrated_prices))
vg_cf_prices = np.zeros(len(calibrated_prices))

for index in range(len(calibrated_prices)):
    vg_mc_prices[index] = np.exp(-r*T)* avg_payoffs[index] *100
    vg_cf_prices[index] = VG_CAL.closed_formula_otko6(strikes[index
    ][0], strikes[index][1])

calibrated_prices['VG MC (%)'] = [round(p,2) for p in vg_mc_prices]
calibrated_prices['VG CF (%)'] = [round(p,2) for p in vg_cf_prices]
```

Listing A.11: Kou Parameters Calibration on Mixed Set of Options

```
"""Find options prices of a mixed set according to the KJD model and
   estimate the best-fitting parameters to market prices"""
[...]
x0=[0.24,1.5,0.5,7.5,5]   #[sigma, lambda, p, eta1, eta2]
bounds = ([0.1, 1e-2, 0, 0, 0], [0.9, 5, 1, 10, 12])

# Define the objective function
```

```
7  def cost_function(x, c_strikes, c_mkt_prices, p_strikes, p_mkt_prices
       , o_strikes, o_mkt_prices):
8      sigm, lamb, p, eta1, eta2 = x
9      KOU = Kou_pricer(S0=S0, K=c_strikes, ttm=T, r=r, sigma=sigm,
       lambd=lamb, p=p, eta1=eta1, eta2=eta2, exercise=exercise)
10     sq_err1 = np.sum((KOU.closed_formula_call(c_strikes) -
       c_mkt_prices) ** 2)
11     sq_err2 = np.sum((KOU.closed_formula_put(p_strikes) -
       p_mkt_prices) ** 2)
12     sq_err = sq_err2 + sq_err1
13     for k in range(len(o_strikes)):
14         sq_err += (KOU.closed_formula_otko(o_strikes[k][0], o_strikes
       [k][1]) - o_mkt_prices[k]) ** 2
15     return sq_err
16
17 start = time.time()
18 kou = scpo.least_squares(cost_function, x0, args=(calls_strikes,
       calls_prices, puts_strikes, puts_prices, otko_strikes,
       otko_mkt_prices), method='trf', bounds=bounds, ftol=1e-7, verbose
       =2)
19 end = time.time()
20 k_params_mixed = [round(p,4) for p in kou.x[:5]]
21 [...]
22 #Reprice Options with estimated parameters for results validation
23 days = 252
24 paths = 5000
25 #CALLS PRICES
26 calls_calib_prices = pd.DataFrame({
27     'STRIKE': calls['STRIKE'],    # array of K for call options
28     'MKT_BID': calls['C_BID'],
```

```
29      'MKT_MID': calls['C_Midpoint'],
30      'MKT_ASK': calls['C_ASK'],
31  })
32  #PUTS PRICES
33  puts_calib_prices = pd.DataFrame({
34      'STRIKE': puts['STRIKE'],     # array of K for call options
35      'MKT_BID': puts['P_BID'],
36      'MKT_MID': puts['P_Midpoint'],
37      'MKT_ASK': puts['P_ASK'],
38  })
39  # OTKO PRICES
40  otko_calib_prices = pd.DataFrame({
41      'K1': otko_aapl['K1'],
42      'K2': otko_aapl['K2'],
43      'MKT_BID (%)': otko_aapl['BID(%)'],
44      'MKT_MID (%)': otko_aapl['Midpoint(%)'],
45      'MKT_ASK (%)': otko_aapl['ASK(%)']
46  })
47  [...]
48  print(f'> MIXED:\t\t\t {k_params_mixed}')
49  Kou4 = Kou_pricer(S0, None, T, r, k_params_mixed[0], k_params_mixed
        [1], k_params_mixed[2], k_params_mixed[3], k_params_mixed[4],
        exercise)
50  SKou4 = Kou4.KouPath(days, paths)
51  avg_payoffs = []
52  for K1, K2 in otko_strikes:
53      payoffs = []          # stores here the payoff for each path, for a
        specific couple K1-K2
54      for path in SKou4.T:
55          payoffs.append(Kou4.payoff_otko(path, K1, K2))
```

```
56      avg_payoffs.append(np.mean(payoffs))
57  kou_mc_prices = np.zeros(len(otko_calib_prices))
58  kou_cf_prices = np.zeros(len(otko_calib_prices))
59  for index in range(len(otko_calib_prices)):
60      kou_mc_prices[index] = np.exp(-r*T)* avg_payoffs[index] * 100
61      kou_cf_prices[index] = Kou4.closed_formula_otko2(otko_strikes[
        index][0], otko_strikes[index][1])
62  otko_calib_prices['KOU MC (%)'] = kou_mc_prices
63  otko_calib_prices['KOU CF (%)'] = kou_cf_prices
64  [...]
```

Listing A.12: Margin Loans Evaluation

```
1  """Simulate Monte-Carlo Paths with estimated parameters and find
       margin loans interest rates"""
2  # MERTON MODEL
3  sigma, lambd, m, v = otko_mert_params_aapl  # OTKO_PARAMS
4  Mert1 = Merton_pricer(last_price, None, T, r, q, sigma, lambd, m, v,
       None)
5  Smert1 = Mert1.MertonPath(days, paths)
6  sigma, lambd, p, eta1, eta2 = mixed_kou_params_aapl # MIXED VANILLA+
       OTKO PARAMS
7  sigma, lambd, m, v = mixed_mert_params_aapl # MIXED  PARAMS
8  Mert2 = Merton_pricer(last_price, None, T, r, q, sigma, lambd, m, v,
       None)
9  Smert2 = Mert2.MertonPath(days, paths)
10
11 # KOU MODEL
12 sigma, lambd, p, eta1, eta2 = otko_kou_params_aapl  # OTKO_PARAMS
```

```
13 Kou1 = Kou_pricer(last_price, None, T, r, sigma, lambd,p, eta1, eta2,
       None)
14 Skou1 = Kou1.KouPath(days, paths)
15 sigma, lambd, p, eta1, eta2 = mixed_kou_params_aapl # MIXED VANILLA+
     OTKO PARAMS
16 Kou2 = Kou_pricer(last_price, None, T, r, sigma, lambd,p, eta1, eta2,
       None)
17 Skou2 = Kou2.KouPath(days, paths)
18
19 #VG MODEL
20 sigm, theta, nu = otko_vg_params_aapl    # OTKO_PARAMS
21 VG1 = VG_pricer(last_price, None, T, r, q, sigm, theta, nu, None)
22 Svg1 = VG1.VarianceGammaPath1(days, paths)
23 sigm, theta, nu = mixed_vg_params_aapl # MIXED VANILLA+OTKO PARAMS
24 VG2 = VG_pricer(last_price, None, T, r, q, sigm, theta, nu, None)
25 Svg2 = VG2.VarianceGammaPath1(days, paths)
26
27 # MARGIN LOANS EVALUATION
28 def margin_loan_price1(Spaths, loan):
29     num_shares_required = [num_of_shares_at_S0] * paths
30     num_margin_calls = np.zeros(paths)
31     flag = ['green'] * paths
32     loss = np.zeros(paths)
33     gain_loss = np.zeros(paths)  # Create an array to store gains/
       losses
34
35     for path in range(paths):
36         for day in range(days - 1):
37             price = Spaths[day][path]
38             if flag[path] == 'green':
```

```
39                if (price * num_shares_required[path]) <= ((1 -
    trigger) * loan):
40                    num_margin_calls[path] += 1
41                    shares_needed = collateral / price
42                    if shares_needed >= max_shares:
43                        num_shares_required[path] = max_shares
44                        flag[path] = 'red'
45                    else:
46                        num_shares_required[path] = round(
    shares_needed, 2)
47                else:
48                    continue
49
50     for index in range(paths):
51         S_T = Spaths[-1, index]
52         final_value = S_T * num_shares_required[index]
53         gain_loss[index] = final_value - loan  # Calculate gains/
    losses, gain is positive, loss is negative
54
55     for index in range(paths):
56         S_T = Spaths[-1, index]
57         final_value = S_T * num_shares_required[index]
58         if final_value >= loan:
59             loss[index] = 0
60         else:
61             loss[index] = loan - final_value
62
63     print('Average margin calls:', round(np.mean(num_margin_calls),
    2))
```

```
64      margin_call_probability = round(np.sum(num_margin_calls) / (days
    * paths), 4)
65      print(f'Probability of a margin call: {margin_call_probability},
    {margin_call_probability * 100}%')
66      print('Average loss:', round(np.mean(loss), 2))
67      print('Average profit & loss:', round(np.mean(gain_loss), 2))
68      print('Average num of shares required:', round(np.mean(
    num_shares_required), 2))
69
70      return loss, gain_loss  # Return both arrays
71
72  """ Few examples of margin loans computation """
73  loan = 1000000          # 1 MLN
74  loan_to_value = 0.7
75  collateral = 1/(loan_to_value / loan)
76  num_of_shares_at_S0 = round(collateral / last_price,2)
77  max_shares = num_of_shares_at_S0 * 2
78  trigger = 0.07
79
80  [loss_merton1, gain_loss_merton1] = margin_loan_price1(Smert1, loan)
81  avg_loss_merton1 = round(np.mean(loss_merton1), 2)
82  [loss_merton2, gain_loss_merton2] = margin_loan_price1(Smert2, loan)
83  avg_loss_merton2 = round(np.mean(loss_merton2), 2)
84
85  [loss_kou1, gain_loss_kou1] = margin_loan_price1(Skou1, loan)
86  avg_loss_kou1 = round(np.mean(loss_kou1), 2)
87  [loss_kou2, gain_loss_kou2] = margin_loan_price1(Skou2, loan)
88  avg_loss_kou2 = round(np.mean(loss_kou2), 2)
89
90  [loss_vg1, gain_loss_vg1] = margin_loan_price1(Svg1, loan)
```

```
avg_loss_vg1 = round(np.mean(loss_vg1), 2)
[loss_vg2, gain_loss_vg2] = margin_loan_price1(Svg2, loan)
avg_loss_vg2 = round(np.mean(loss_vg2), 2)

[...]
#Interest rates computation
price= []
price.append(avg_loss_merton1/loan*100)
price.append(avg_loss_merton2/loan*100)
price.append(avg_loss_kou1/loan*100)
price.append(avg_loss_kou2/loan*100)
price.append(avg_loss_vg1/loan*100)
price.append(avg_loss_vg2/loan*100)

[...]
```