Politecnico di Torino

Corso di Laurea Magistrale in Ingegneria Matematica

Basket Option Pricing for Processes with Jumps Using Sparse Grids and Fourier Transforms

A.A 2014/2015

relatori:

Prof. Luigi Preziosi,

Prof. Raul Tempone,

Fabian Crocce,

Yu-Ho Häppölä

elaborato di laurea di:

Alessandro Iania

matricola 190590

Ringraziamenti

Sto concludendo un percorso che probabilmente non avrei mai portato a termine con le mie sole forze.

Devo ringraziare il gruppo di KAUST per avermi permesso di scrivere questa tesi nell'ambito di un'esperienza stimolante, vissuta in un'ambiente internazionale. Grazie dunque a Raul e a Fabian, ma anche ai ragazzi con cui ho tanto condiviso in questi 7 mesi: Marco, Fuad e tutti gli altri.

La scrittura della tesi è solo la fine di un percorso di crescita molto più ampio, e sarei cresciuto molto meno senza il contributo costante dei miei Amici. Gianluca Brero, che mi manca ogni giorno. Matteo Rabagliati, a cui mi accorgo spesso di aver finito per assomigliare nei miei atteggiamenti: lo porto con me. Andrea Valenti, così differente da come sono io, ma a cui voglio un gran bene. Ferdinando, che ringrazio per essersi fatto scegliere, un giorno di parecchi anni fa, in una classe di liceo. E poi i compagni di corso, Veronica, Giri, Erica, Ludo, Simo, Marco, Chiara.

Ai miei genitori, che mi supportano e mi sopportano da ormai 26 anni.

Contents

Introduction

The Common Clock Variance Gamma (CCVG) model, introduced by Madan and Seneta in the 1990 [15], permits to overcome some of the shortcomings that affect the typical assumption of log-normal distribution for stock prices made in the Black-Scholes model [2]. CCVG model has been therefore considered in the literature to model a set of correlated assets' prices. E. Luciano and W. Schoutens [13] calibrated this model in the particular case in which Σ is a diagonal matrix.

We depart from the fundamental theorem of option pricing, which states that the price of a derivative is the expected income under a particular measure, the so-called risk-neutral measure. Assuming a risk-neutral measure given, both algorithms compute this expected income by computing the corresponding multidimensional integral using sparse grids. The use of sparse grids allows us to overcome the curse of dimensionality to a certain extent. As the sparse grid integration requires a special class of integrand functions, we manipulate our d-dimensional integral using the law of total expectation to split it into two parts. Once we obtain a problem formulation that can be solved using sparse grids, we present two numerical algorithms to compute our multidimensional integrals. The first algorithm uses the Fast Fourier Transform (FFT) to evaluate the $(d-1)$-dimensional joint probability density function. The second algorithm exploits the fact that the CCVG's pdf conditioned to the random time change (called the clock) has a normal distribution. We describe the two algorithms and we report the numerical results obtained running them to compute integrals in different dimensions. We conclude the thesis commenting the numerical results and reporting suggestions for further works.

In this thesis we present two algorithms based on sparse grids to compute basket options' price under the CCVG model. We introduce the fundamental concepts needed to fully understand the dissertation, but the reader is still supposed to be competent in basic probability and measure theory.

A Matlab implementation of the two methods are included respectively in the Appendix A and in the Appendix B.

Chapter 1

Option Pricing

1.1 Definitions

A **derivative** can be defined as a financial instrument whose value derives from the values of other, more basic, underlying variables. Often the variables underlying derivatives are the prices of traded assets like **stocks**. Stocks are shares in a company which provide partial ownership in the company, proportional with the investment in the company. They can be bought or sold at any time time t at a **spot price** S_t. All possible future prices $S_t(t)$, together with probabilistic information on the likelihood of a particular price history, constitute the **price process** $S = \{S_t : t \geq 0\}$ of the asset[7].
A **stock option** is a derivative whose value is dependent on the price of a stock. Its value at **expiration time** or **time to maturity** T is determined by the price process of the option up to time T. The **holder** of the option has the right - and not the obligation - to do something. There are several types of options. A **call option** gives the holder of the option the right to buy an asset by a certain expiration date for a certain **strike price** or **exercise price.** A **put option** is an analogous contract, but the holder has the right to sell the asset. Call and put options can be either European or American. An **American option** can be exercised at any time until the expiration date, whereas **European options** can be exercised only on the expiration date

itself. To every option contract there are two sides. On one side there is the investor who has taken the long position (i.e., he bought the option). On the other side there is the investor who has taken a short position (i.e., he has sold or *written* the option). The writer of an option receives cash up front, but has potential liabilities later. His profit or loss is the reverse of that for the purchaser of the option.
The return on an investment is called **payoff**. If K is the strike price and S_T is the final price of the underlying asset, the payoff at time T of a long position in an European call option is

$$\mathcal{P}(S_T) = \max\left(S_T - K, 0\right) = (K - S_T)^+$$

The payoff of the analogous put option is (Figure 1.1):

$$\mathcal{P}(S_T) = \max\left(K - S_T, 0\right) = (S_T - K)^+$$

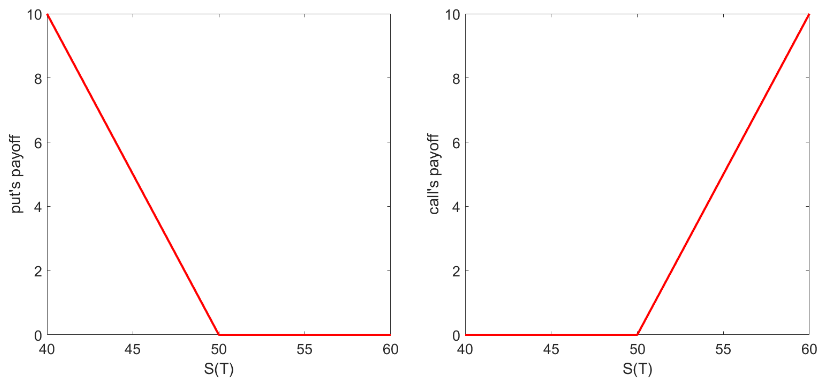

Figure 1.1: payoff of a put (left) and of a call (right) option with strike $K = 50$.

Call options are referred to as *in the money* when the difference between stock price S and strike price K is positive, *at the money* when $S = K$, *out of the money* when $S < K$. There are options on stocks, currencies, stock indices, and futures.

1.2 Properties of Stock Options

The price of a stock option depends on six factors [11]:

- The current stock price, S_0
- The strike price, K
- The time to expiration, T
- The volatility of the stock price, σ
- The risk-free interest rate, r
- The dividends that are expected to be paid

The value of a call option generally increases as the current stock price, the time to expiration, the volatility, and the risk-free interest rate increase, while it decreases as the strike price and expected dividends increase (Figure 1.2).

1.2.1 Multidimensional Options: Basket Call Options

A call options on a set of stock with component weights **c** offers its holder the right to buy the portfolio at a certain strike price $K = e^k$. The payoff at maturity T is therefore

$$\mathcal{P}(\mathbf{S}_T) = \max\{\mathbf{c} \cdot \mathbf{S}_T - K, 0\} = (\mathbf{c} \cdot \mathbf{S}_T - K)^+ \tag{1.1}$$

The volatility of the basket is generally smaller than that of each underlying. Therefore, the basket option premium is less than the sum of the premiums of all individual options on each underlying.

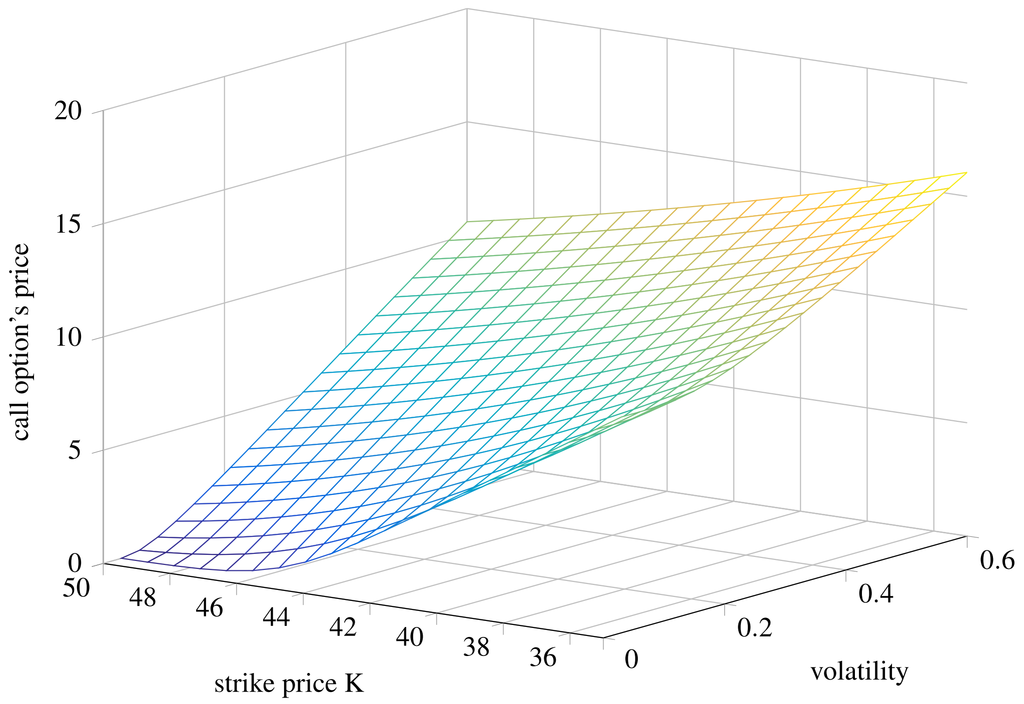

Figure 1.2: price of a call option, evaluated using the B-S formula, depending on the strike price and on the volatility σ

.

1.3 Pricing Background

1.3.1 Stochastic Processes

Let $(\Omega, \mathscr{F}, \mathbb{P})$ be a probability space, and let $\mathcal{T}$ be a set of infinite cardinality. If for each $t \in \mathcal{T}$ there is a random variable $S_t : \Omega \to \mathbb{R}$ defined on $(\Omega, \mathscr{F}, \mathbb{P})$, the function $S : \mathcal{T} \times \Omega \to \mathbb{R} : S(t, \omega) = S_t(\omega)$ is called a **stochastic process** with indexing set $\mathcal{T}$, and is written $S = \{S_t, \mathrm{t} \in \mathcal{T}\}$.
Prices of stocks can be modelled by stochastic processes in continuous time $t \in \mathcal{T} = [0, T]$ where the maturity T is the time horizon. A price model is said to be arbitrage-free if it is not possible buy and sell a security at two different prices in two different markets, making profits without risk.

1.3.2 Pricing

An important result states that a price model is arbitrage free if and only if there exists an equivalent risk-neutral measure $\mathbb{Q} \sim \mathbb{P}$ such that $e^{-\int_0^t r(s)ds} S_t$ is a $\mathbb{Q}-$martingale[20], and therefore

$$V(S_0) = \mathbb{E}^{\mathbb{Q}} \frac{[\mathrm{V}(S_T, T)]}{e^{-\int_0^T rdt}}$$

(see for example [11, chap. 29]). With the hypothesis that r is constant and the measure $\mathbb{Q}$ is given, then the option price at time 0 can be evaluated by computing

$$V(S_0) = e^{-rT}\mathbb{E}^{\mathbb{Q}}[\mathcal{P}(S_T, K)] \tag{1.2}$$

It is a fundamental result from mathematical finance: under certain model assumptions, the prices of derivatives can be represented as expected values, which in turn correspond to high-dimensional integrals (in the case of multi-dimensional options). Since the integrals can in most cases not be calculated analytically, they have to be computed numerically. For this scope, two broad classes of computational methods have emerged: statistical sampling approach and grid-based methods. In this thesis, we focus on the second class.
We assume the underlying prices at maturity T to be modelled by

$$S_T = S_0 e^{(rT + X_T)},$$

where the random variable X_T describes the log-return on forward contracts to buy stocks at time T.

1.4 Stock Price Models

1.4.1 Black Scholes Model

Definition 1.1. *A standard one dimensional Wiener process (also called Brownian motion) is a stochastic process $\{W_t\}_{t\geq 0}$ with the following properties:*

- $W_0 = 0$
- *the function* $t \longmapsto W_t$ *is continuous in* t
- *the process* $\{W_t\}_{t \geq 0}$ *has stationary, independent increments*
- *the increment* $W_{t+p} - W_p$ *has a normal distribution with mean* 0 *and variance* t, $\mathcal{N}(0,t)$.

In the Black-Scholes model, published by F. Black and M. Scholes in 1973 [2], the price process S_t of the risky asset is modelled by assuming that the return due to price change in the time interval $\delta t > 0$ is

$$\frac{S_{t+\delta t} - S_t}{S_t} = \frac{\delta S_t}{S_t} = r\delta t + \sigma \delta W_t,$$

in the limit $\delta t \to 0$, i.e. that it consists of a deterministic part $r\delta t$ and a random part $\sigma \delta W_t$, where W_t is a Wiener process (Figure 1.3). In the limit $\delta t \to 0$, we obtain the stochastic differential equation

$$dS_t = rS_t dt + \sigma dW_t, \quad S_0 > 0. \tag{1.3}$$

Applying the Ito's Lemma to the function $G = \log(S)$, it is easy to verify that $\log(S)$ follows a generalized Wiener process with constant drift rate $\mu - \frac{\sigma^2}{2}$ and constant variance rate σ^2[11, chap. 13]. The change in $\log(S)$ between times 0 and T is therefore normally distributed, and

$$\log(S_T) \sim \mathcal{N}\left[ln(S_0) + \left(\mu - \frac{\sigma^2}{2}\right)T, \sigma^2 T\right] \tag{1.4}$$

It follows that (1.3) admits the unique solution (see for example [10, pag.7])

$$S_t = S_0 e^{\left(\mu - \frac{\sigma^2}{2}\right)t + \sigma W_t},$$

and that

$$X_T \sim \mathcal{N}\left(-\frac{1}{2}\sigma^2 T, \sigma^2 T\right).$$

The same Black and Scholes' assumptions are underlying different models we can find in literature. However, empirically observed log returns of risky

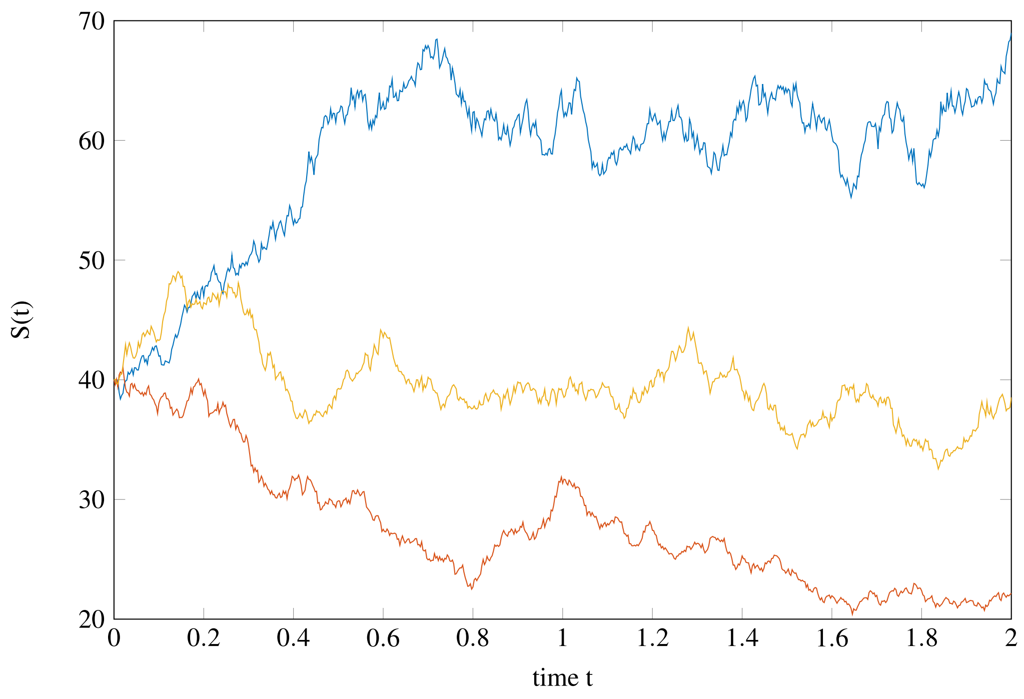

Figure 1.3: Three simulation of stock price's path in the B-S model with $S_0 = 40$, $\mu = 0.1$, $\sigma = 0.2$, $T = 2$.

assets are not normally distributed but exhibit significant skewness and kurtosis. These empirical facts are not covered by the Black and Scholes model. Moreover, a Brownian motion has continuous sample paths, when prices are in reality driven by jumps. Lastly, the model is not able to give realistic probabilities of extreme events, like crashes and defaults. It is because the Normal distribution has too light tails, and the model produces continuous sample paths. The Brownian motion needs a substantial time to reach a low barrier, where in reality jumps can cause an almost immediate move over the barrier. Other objections against the log-normal assumption are based on observed option prices. They are related to the implied volatility (computed minimizing the distance between observed option prices and those implied by the B-S model) and are best known under the name volatility smile. The Black-Scholes model assumes that the implied volatility is constant. As one computes the implied volatility, it is possible to observe that at a given maturity the volatility seems a non linear function of the strike price K. To correct the model, in order to capture this behaviour, one strategy is to introduce jumps in our model by replacing the continuous Brownian motion by a jump process.

Multidimensional B-S Model

In order to do a multivariate model, a natural way is to consider a vector of n dependent brownian motions $\mathbf{W}_t$. If $\Sigma = AA^T$ is the constant volatility matrix, the price process is therefore:

$$\frac{d\mathbf{S_t}}{\mathbf{S_t}} = rdt + Ad\mathbf{W}_t \tag{1.5}$$

The random vector $\mathbf{X}_T$ is then normally distributed

$$\mathbf{X}_T^{BS} = A\mathbf{W}_T - \frac{1}{2}diag(\Sigma)T \quad \sim \quad \mathcal{N}_d(-\frac{1}{2}diag(\Sigma)T, \Sigma T)$$

and the characteristic function is

$$\phi_T(\mathbf{z}) = e^{-\frac{1}{2}T(idiag(\Sigma)^T\mathbf{z}+\mathbf{z}^T\Sigma\mathbf{z})}. \tag{1.6}$$

For details about the multidimensional BS model see for example [20]. The multivariate Gaussian model has some additional shortcomings. Among these, the fact that the dependence between the assets does not present tail dependency, and therefore is not realistic.

1.4.2 The Variance Gamma Model

This process was originally introduced by Madan and Seneta (1990) in [15]. It is a pure jump process, and it introduces jumps, skewness, kurtosis and non-Gaussian dependence by changing the clock (the random time) of a standard Brownian motion $B(t)$ by a Gamma process. The Variance Gamma Process is defined as

$$X_t = VG(t) = \theta G(t) + \sigma B(G(t)),$$

where $G(t)$ is a gamma process and θ is a drift value. It is a positive, strictly increasing and purely discontinuous Lévy process. In Figure 1.4 are represented three simulated paths for a stock's price in the VG model.

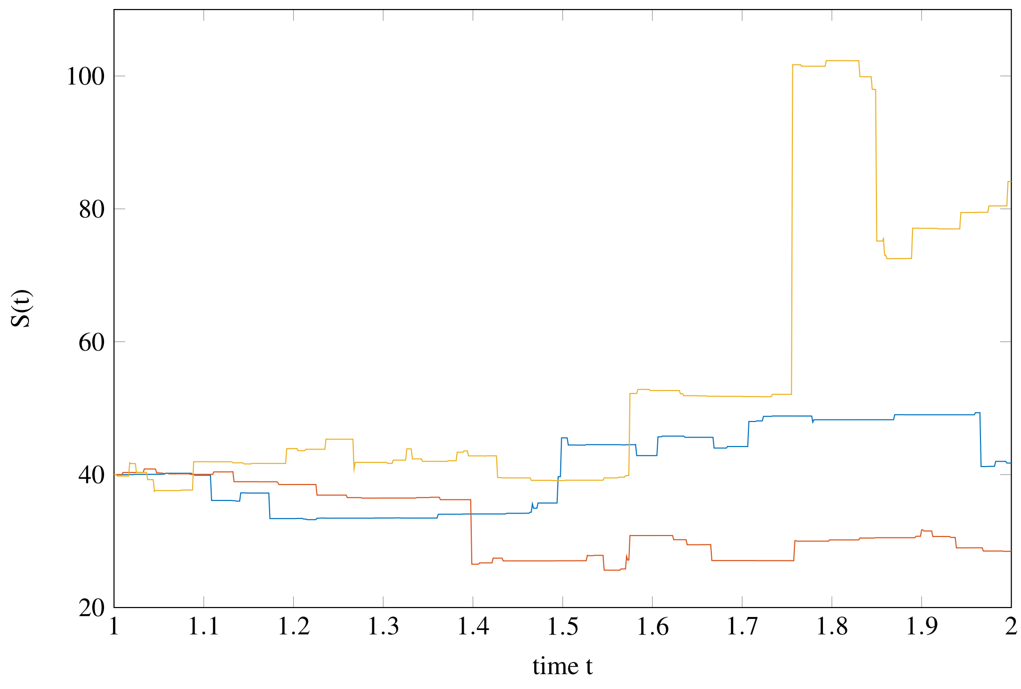

Figure 1.4: Three simulated paths of a stock's price with $S_0 = 40, \sigma = 0.2, T = 2, \nu = 0.4, \theta = 0.03$.

Generally the condition $\mathbb{E}(G(t)) = t$ is imposed; this normalization condition means that the stochastic clock is not expected to run faster or slower than the real one. Using this normalization, the density of $G(t)$ can be written as

$$f_{G(t)}\left(x; \frac{t}{\nu}, \frac{1}{\nu}\right) = x^{\frac{t}{\nu}-1} \frac{e^{-\frac{x}{\nu}}}{\Gamma(\frac{t}{\nu})\nu^{\frac{t}{\nu}}}, \quad \nu > 0 \tag{1.7}$$

where $\Gamma(\cdot)$ is the gamma function.

Using the fact that $VG(t)|G(t) = g$ has a Gaussian distribution with mean θg and variance $\sigma^2 g$, it is possible to derive density and characteristic function of a Brownian motion time changed by the subordinator:

(1.8a)

$$\varphi_{VG(t)}(u) = (1 - iu\nu\theta + \frac{1}{2}\nu u^2\sigma^2)^{-\frac{t}{\nu}},$$

(1.8b)

$$f_{VG(t)}(x) = \frac{2e^{\frac{\theta x}{\sigma^2}}}{\nu^{\frac{t}{\nu}}\sqrt{2\pi}\sigma\Gamma(\frac{t}{\nu})}\left(\frac{x^2}{\frac{2\sigma^2}{\nu}+\theta^2}\right)^{\frac{t}{2\nu}-\frac{1}{4}} K_{\frac{t}{\nu}-\frac{1}{2}}\left(\frac{1}{\sigma^2}\sqrt{x^2}(\frac{2\sigma^2}{\nu}+\theta^2)\right),$$

where $K_{\frac{t}{\nu}-\frac{1}{2}}$ is the second type modified Bessel function of order $\frac{t}{\nu}-\frac{1}{2}$ (see [14] for a detailed derivation).

Multidimensional CCVG Model

Let $B(t)$ be a d-dimensional Brownian motion process with covariance rate Σt and let $G(t)$ denote an univariate gamma process. Than the process

$$\mathbf{X}(t) = \theta G(t) + \mathbf{B}(G(t)) \tag{1.9}$$

is a multivariate gamma process, where θ is a d-dimensional drift vector.The sources of dependence are two: one is from the stochastic clock, and it means that all the process jump together. The second one comes from the correlation between Brownian motions, and it means that the amplitude of the jumps is correlated across assets. Because of the first source of dependence, the components of the process remain dependent even if $\Sigma = I_d$. The CCVG model in the particular case in which Σ is a diagonal matrix was calibrated by E. Luciano and W. Schoutens in [13] providing a good fit.
The characteristic function of the CCVG is:

$$\varphi_{CCVG}(\mathbf{u}) = (1 - \nu\mathbf{u}'\theta + \frac{1}{2}\nu\mathbf{u}'\Sigma\mathbf{u})^{-\frac{t}{\nu}}. \tag{1.10}$$

Even in multidimensional case, considering that $X(t)|G_t$ is a Gaussian random vector with mean $g\theta$ and variance covariance matrix $g\Sigma$, it is possible to compute the probability density function of the CCVG:

(1.11)

$$f_{VG(t)}(\mathbf{x}) = \frac{2e^{\mathbf{x}'\Sigma^{-1}\theta}}{\nu^{\frac{t}{\nu}}(2\pi)^{\frac{N}{2}}|\Sigma|^{\frac{1}{2}}\Gamma(\frac{t}{\nu})}\left(\frac{\mathbf{x}'\Sigma^{-1}\mathbf{x}}{(\frac{2}{\nu}+\theta'\Sigma^{-1}\theta)}\right)^{\frac{t}{2\nu}-\frac{N}{4}} K_{\frac{t}{\nu}-\frac{N}{2}}\left(\sqrt{(x'\Sigma^{-1}x)(\frac{2}{\nu}+\theta'\Sigma^{-1}\theta)}\right),$$

where $K_{\frac{t}{\nu}-\frac{N}{2}}$ is the second type modified Bessel function of order $\frac{t}{\nu}-\frac{N}{2}$ (for a detailed derivation see [6]). Notice that (1.11) reduces to (1.8) when the number of dimensions $N = 1$. In this model, the asset values follow geometrical Brownian Motions time-changed by a common Gamma business time (the stochastic clock):

$$S_i^{(i)} = S_0 e^{\theta_i G_t + \sigma_i W_{G_t}^{(i)}} = S_0 e^{X_t^{(i)}}, \quad t \geq 0, \tag{1.12}$$

where $X^{(i)} = \{X_t^{(i)}, t \geq 0\}$ is a VG process with parameters $(\sigma_i, \nu, \theta_i)$.

Chapter 2

Integration on Sparse Grids

2.1 Problem Formulation

Our goal is to price multidimensional options. As expressed in the section 1.3, we can see the price of a derivative as an expected value; therefore, given a payoff function $\mathcal{P}(\boldsymbol{S}_T)$ our purpose is to compute

$$V(\mathbf{S}_0) = \mathbb{E}\left(\mathcal{P}(\boldsymbol{S}_T)\right).$$

In the particular case of a basket call option, defined in subsection 1.2.1, the expression to compute is:

$$V(\mathbf{S}_0, K) = \int_{\mathbb{R}^d} \left(\mathbf{c} \cdot \mathbf{S_0} e^{\mathbf{x}} - K\right) f_{\mathbf{X_T}}\left(\mathbf{x}\right) d\mathbf{x}. \tag{2.1}$$

A straightforward method to evaluate multidimensional integrals is to compound univariate rules coordinate-wise. The problem with this approach is in the number of function evaluations: the computing cost grows exponentially with the dimension of the problem, since using univariate rules with N evaluation points results in a d-dimensional quadrature rule with a total of N^d points. This phenomenon is known as *curse of dimensionality*, and it makes the intuitive approach useless even on modern computers for moderately high values of d. Indeed, classical product quadrature methods for the computation of multivariate integrals achieve with n evaluations of the integrand an accuracy of

$$\epsilon(n) = O(n^{-\frac{r}{d}})$$

for functions with bounded derivatives up to order r [1]. For fixed r, their convergence rates $\frac{r}{d}$ thus deteriorate as the dimensions increase and are already in moderate dimensions so small that high accuracies can no longer obtained in practise. On the positive side, the case $r = d$ indicates that the problem of high dimensions can sometimes be compensated by, e.g., a high degree of smoothness. There have been several attempts to overcome the problem: it is known from numerical complexity theory [19] that some algorithm can break the curse of dimensionality for certain classes of functions. Monte Carlo (MC) methods are one example of this class of algorithms[3]. Here, the integrand is approximated by the average of n function values at random points. For square integrable functions f, the expected mean square error of the Monte Carlo method with n sample is $\epsilon(n) = O(n^{-\frac{1}{2}})$. The convergence is thus independent of the dimension d, but quite slow.
Under more restrictive assumptions on the smoothness of the integrand, faster rate of convergence can be attained by deterministic integration methods such as quasi-Monte-Carlo methods [9] and sparse grid methods[4].
In this thesis we use the Smolyak sparse grid integration to evaluate a multidimensional basket option's price. It is important to notice that the function we want to integrate in (2.1) is not regular (Figure 2.1).

In order to gain regularity and to make our algorithms converging, it is convenient to manipulate the expression (2.1). If we set the notation:

- $\mathbf{S} = (S_1, \ldots, S_d)$,
- $\hat{\mathbf{S}} = (S_2, \ldots, S_d)$,
- $\mathbf{s} = (s_1, \ldots, s_d) \ \in \mathbb{R}^d$,
- $\hat{\mathbf{s}} = (s_2, \ldots, s_d) \ \in \mathbb{R}^{d-1}$,
- $\mathbf{c} = (c_1, \ldots, c_d)$,

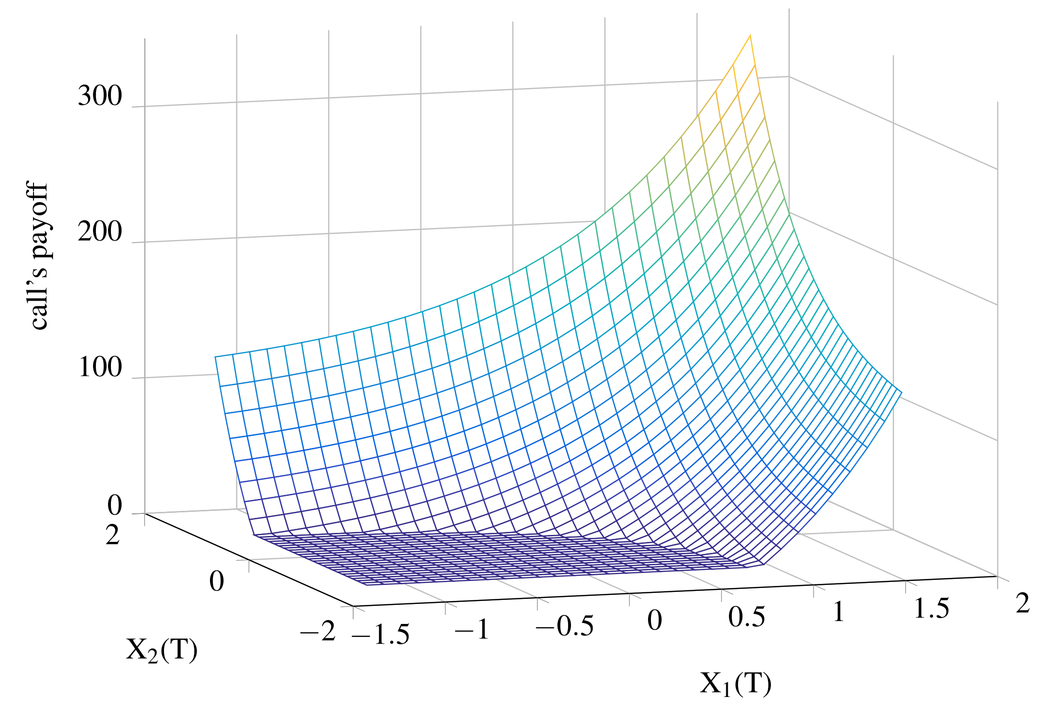

Figure 2.1: the payoff is not a regular function

- $\hat{\mathbf{c}} = (c_2, \dots, c_d)$,

and we set an analogous notation for $\mathbf{X} = \log \frac{\mathbf{S}_T}{\mathbf{S}_0}$, then we can write[1]:

(2.2)

$$\begin{aligned}\mathbb{E}\left((\mathbf{c}\cdot\mathbf{S}_T - K)^+\right) &= \mathbb{E}\left(\left(c_1 S_{1T} - \underbrace{\left(K - \hat{\mathbf{c}}\cdot\hat{\mathbf{S}}_T\right)}_{K'}\right)^+\right)\\ &= \mathbb{E}\left(\mathbb{E}\left(\left(c_1 S_{1T} - K'(\hat{\mathbf{c}}, \hat{\mathbf{S}}_T)\right)^+ \middle| \hat{\mathbf{S}}_T\right)\right)\\ &= \int_{\mathbb{R}^{d-1}}\left(\int (c_1 S_{10} e^x - K')^+ \ f_{X_{1T}}\left(x \middle| \hat{\mathbf{X}}_T = \hat{\mathbf{x}}\right) dx\right) \ f_{\hat{\mathbf{X}}_T}(\hat{\mathbf{x}})\, d\hat{\mathbf{x}}.\end{aligned}$$

It is important to remark that the function

$$\hat{\mathbf{x}}_T \longmapsto \int (c_1 S_{10} e^x - K')^+ \ f_{X_{1T}}\left(x \middle| \hat{\mathbf{X}}_T = \hat{\mathbf{x}}\right) dx \tag{2.3}$$

is more regular(Figure 2.2).

[1]Christian Bayer, personal communication

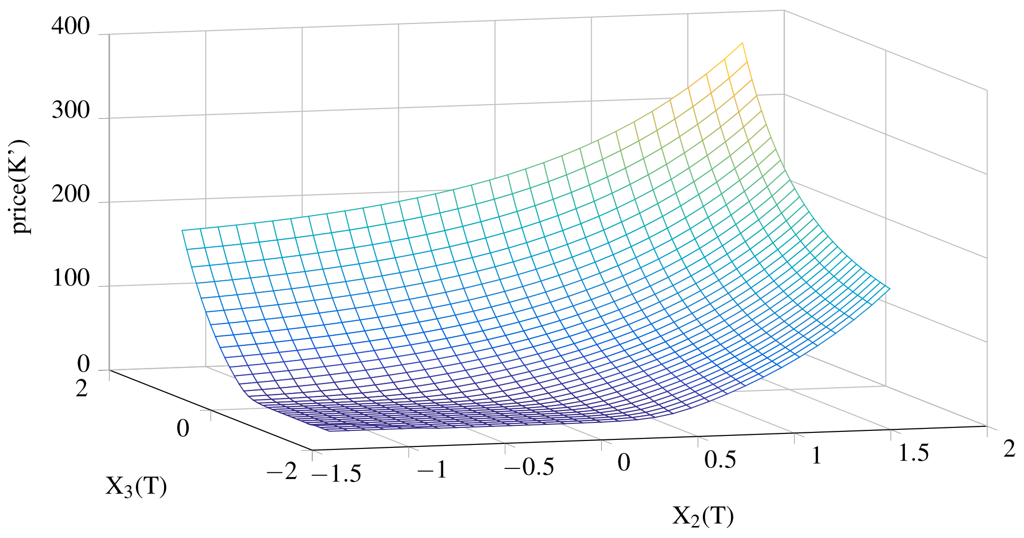

Figure 2.2: the call's price depending on (K, X_{2T}, X_{3T}) is a regular function

2.2 The Smolyak Method

In the 1963, the Russian mathematician Sergey A. Smolyak introduced a numerical technique to represent, integrate or interpolate high dimensional functions. For special function classes, such as spaces of functions which have bounded mixed derivatives, Smolyak's construction can overcome the curse of dimensionality to a certain extent. In the original paper [18] the method was developed for general tensor product spaces. We limit the dissertation on integrals over regions of $\mathbb{R}^d$, i.e. connected and non-empty subsets, which may contain some of their boundary points[12].
Let's focus on the function spaces

$$H^r(\Omega) = \left\{ f : \Omega \to \mathbb{R} : \quad \exists \quad \frac{\partial^\alpha f(x)}{\partial x^\alpha} \quad \text{and is bounded in } \Omega \text{ for all } |\alpha|_\infty \leq r \right\}$$

for a region $\emptyset \neq \Omega \subseteq \mathbb{R}^{dim}$. r is the regularity of the functions in $H^r(\Omega)$, and their norm is

$$||f||_{H^r(\Omega)} = \max_{\substack{\alpha \in \mathbb{N}^d \\ |\alpha|_\infty \leq r}} \sup \left\{ \left| \frac{\partial^\alpha f(x)}{\partial x^\alpha} \right| ; \ x \in \Omega \right\}$$

Let's define two sets Ω and Θ and two linear and bounded functionals:

$$\begin{array}{ll} \emptyset \neq \Omega \subseteq \mathbb{R}^{d_1} & \emptyset \neq \Theta \subseteq \mathbb{R}^{d_2} \\ S : H^r(\Omega) \to \mathbb{R} & T : H^r(\Theta) \to \mathbb{R} \\ Sf = \sum_{i=1}^{m} a_i f(x_i) & T\tilde{f} = \sum_{i=1}^{n} b_i \tilde{f}(y_i) \end{array}$$

$\Rightarrow \Omega \times \Theta \subseteq \mathbb{R}^{d_1+d_2}$ and the **tensor product** of S and T is the linear functional

$$S \otimes T : H^r(\Omega \times \Theta) \to \mathbb{R},$$

defined by setting:

$$S \otimes T f = \sum_{i=1}^{m} \sum_{j=1}^{n} a_i b_j f(x_i, y_j).$$

Let's consider the integral:

$$I_W^d f = \int_{I_1} \cdots \int_{I_d} W_1(x_1) \cdots W_d(x_d) f(x_1, \ldots, x_d) dx_1, \cdots, dx_d \tag{2.4}$$

Let $(U_k^{(j)})_{j=1}^{d}$ be a sequence of univariate rules, where k denotes the number of evaluation points; $(w_i^{(j)})_{i=1}^{N(j)}$ and $(x_i^{(j)})_{i=1}^{N(j)}$ are weights and nodes (evaluation points) of the rule $U_k^{(j)}$.

We can use the univariate rules to coordinate-wise evaluate the integral (2.4). If we define

$$\bigotimes_{i=1}^{1} U_k^{(i)} = U_k^{(1)} \quad \text{and} \quad \bigotimes_{i=1}^{d} U_k^{(i)} = \bigotimes_{i=1}^{d-1} U_k^{(i)} \otimes U_k^{(d)},$$

then

$$I_W^d f = \bigotimes_{i=1}^{d} U_k^{(i)} f + error.$$

Let $(U_i^{(j)})_{i=1}^{\infty}$ be a sequence of univariate quadrature rules in the interval $\emptyset \neq I_j \subseteq \mathbb{R}, \quad j = 1, \ldots, d$. We can define the *difference operators in I_j:*

$$\Delta_0^{(j)} = 0, \quad \Delta_1^{(j)} = U_1^{(j)} \quad \text{and} \quad \Delta_{i+1}^{(j)} = U_{i+1}^{(j)} - U_i^{(j)}$$

The *Smolyak quadrature rule* of order k in the hyper rectangle $I_1 \times \cdots \times I_d$ is the operator

$$Q_k^d = \sum_{\substack{|\alpha|_1 \le k \\ \alpha \in \mathbb{N}^d}} \bigotimes_{i=1}^{d} \Delta^i_{\alpha_i}. \tag{2.5}$$

Remark 2.1. *If instead of the* $|\ \ |_1$ *norm we consider the* $|\ \ |_\infty$ *norm, we obtain the ordinary full tensor product operator:*

$$\bigotimes_{i=1}^{d} U_k^i = \sum_{\substack{|\alpha|_\infty \le k \\ \alpha \in \mathbb{N}^d}} \bigotimes_{i=1}^{d} \Delta^i_{\alpha_i}.$$

The Smolyak rule is therefore a delayed version of the full tensor product.

In order to avoid the cancellation of terms due to the difference operators (see for example [12]), we need to understand the behaviour of difference operators in tensor product operations .

Proposition 2.2. *Let* $\boldsymbol{\alpha} \in \mathbb{N}^d$ *and* $\boldsymbol{\alpha} \ge \mathbf{1}$*. Then,*

$$\bigotimes_{i=1}^{d} \Delta^{(i)}_{\alpha_i} = \sum_{\substack{\gamma \in \{0,1\}^d \\ \alpha - \gamma \ge \mathbf{1}}} (-1)^{|\gamma|_1} \bigotimes_{i=1}^{d} U^{(i)}_{\alpha_i - \gamma_i}. \tag{2.6}$$

Using (2.6) it is already possible write (2.5) without the difference operators. Through other straightforward manipulations, it is possible to derive the *combination method,* in the form presented by Wasilkowski and Wozniakowski in [21].

Proposition 2.3. Combination method

Let $U_i^{(j)}$ *be univariate quadrature rules in the interval* $\emptyset \ne I_j \subseteq \mathbb{R}$ *and suppose that* $k \ge d$*. Then, we can rewrite the* (2.5)*:*

$$Q_k^d = \sum_{\substack{max\{d, k-d+1\} \le |\alpha|_1 \le k \\ \alpha \in \mathbb{N}^d, \alpha \ge \mathbf{1}}} (-1)^{k - |\alpha|_1} \binom{d-1}{k - |\alpha|_1} \bigotimes_{i=1}^{d} U^{(i)}_{\alpha_i}. \tag{2.7}$$

We computed the expected value of a basket call option with 5 uncorrelated assets using the naive full tensor approach and the Smolyak rule. The comparison between the rates of convergence is represented in Figure 2.3

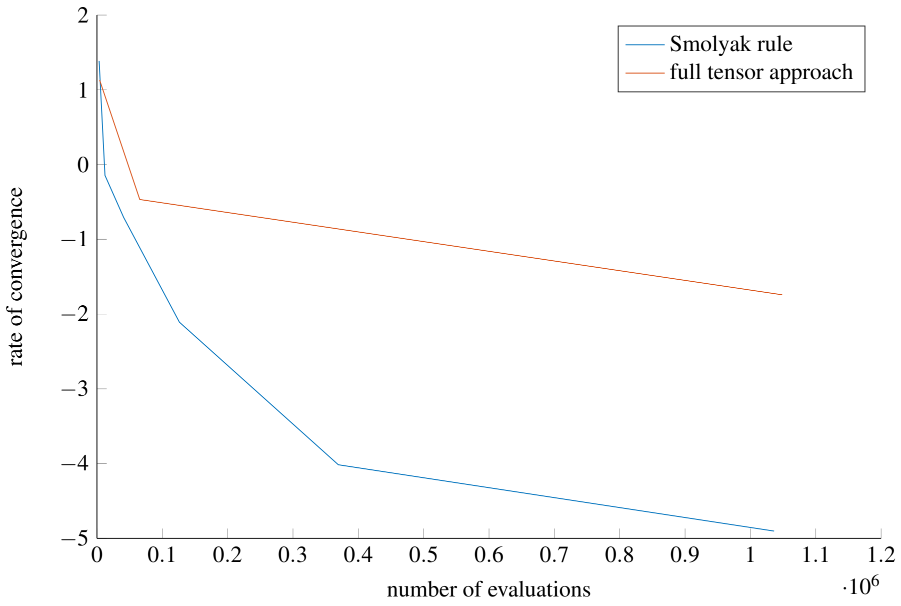

Figure 2.3: Comparison of the rate of convergence, defined as the $\log_{10}$ of the difference between two consecutive evaluations.

2.2.1 Complexity

(2.7) renders the evaluation points sets on the Smolyak rule explicitly known. Let $U_i^{(j)}$ be a univariate quadrature rules in I_j and let $X_i^{(j)}$ be point sets containing the respective quadrature rules' evaluation points. Then the evaluation points of Q_k^d, according with (2.7), form the set

$$\eta(k,d) = \sum_{\substack{max\{d,k-d+1\} \leq |\alpha|_1 \leq k \\ \alpha \in \mathbb{N}^d, \alpha \geq \mathbf{1}}} \boldsymbol{X}_{\alpha_1}^{(1)} \times \cdots \times \boldsymbol{X}_{\alpha_d}^{(d)} \quad \text{for all } k \geq d.$$

The elements of the set $\eta(k,d)$ are the *nodes* of Q_k^d. The cardinality of the set $\eta(k,d)$ is the *cost* of Q_k^d, since it is the minimum number of function

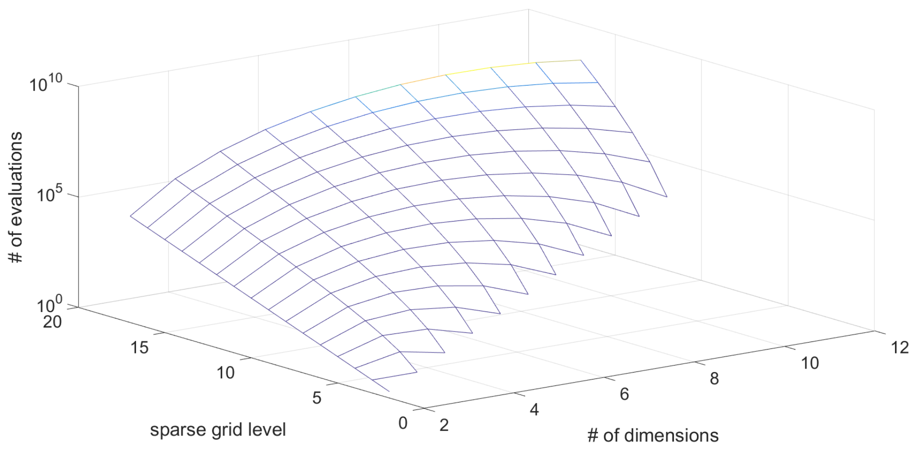

Figure 2.4: Number of evaluations in a non-nested grid, depending on the dimension and on the sparse grid level sgl.

evaluations to compute the Smolyak rule (see figure Figure 2.4).

Let's call $u_i^{(j)}$ the number of evaluation points of $U_i^{(j)}$. If $n_i^{(j)} \leq n_{i+1}^{(j)}$ and the univariate rules are nested (i.e. $\boldsymbol{X}_i^{(j)} \subseteq \boldsymbol{X}_{i+1}^{(j)}$), then the nodes of Q_k^d form the set

$$\eta(k,d) = \bigcup_{\substack{|\alpha|_1 = k \\ \alpha \in \mathbb{N}^d, \alpha \geq \mathbf{1}}} \boldsymbol{X}_{\alpha_1}^{(1)} \times \cdots \times \boldsymbol{X}_{\alpha_d}^{(d)} \quad \text{for all } k \geq d.$$

If $n_k^{(1)} = O(2^k)$, the order of Q_k^d is [16]:

$$\eta(k,d) = O(2^k \cdot k^{(d-1)}).$$

2.2.2 Error Bound

For all interpolatory 1-dimensional quadrature formulas with positive weights, for functions $f \in C^r$, holds the bound

$$|E_l^1 f| = O((n_l^1)^{-r}).$$

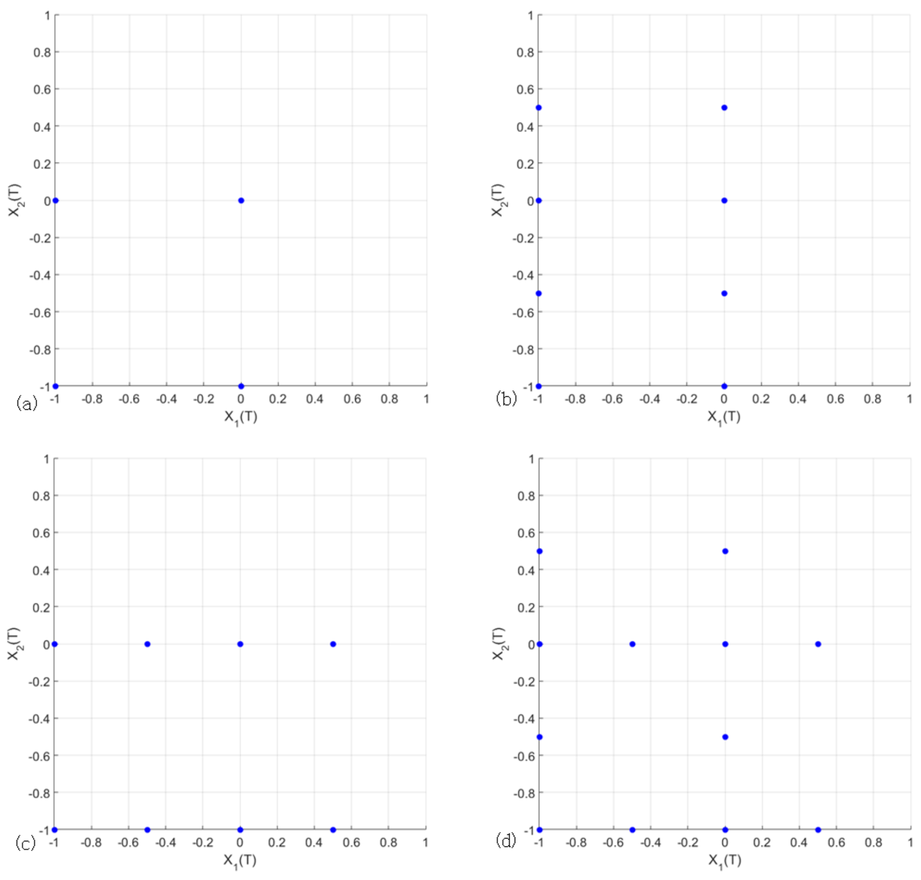

Figure 2.5: Example of sgl=3 sparse grid construction in the 2-dim square $[-1,1]^2$: $\alpha = [1,1](a)$, $\alpha = [1,2](b)$, $\alpha = [2,1](c)$, and the final construction (d).

Taking one such quadrature formula as 1-dimensional basis, if $f \in W_d^r$ and $n_l^1 = O(2^l)$, the error of Smolyak's quadrature formula is of order

$$|E_l^d f| = O(2^{-lr} \cdot l^{(d-1)\cdot(r+1)}).$$

For more details about this result, see [21]. In [17] E. Novak and K. Ritter proved that error bounds for classical spaces C_d^r indicate an exponential dependence on the dimension.

Chapter 3

The Algorithms

As we expressed in section 2.1, in the case of a basket call option our goal is to compute the integral (2.1):

$$V(\mathbf{S}_0, K) = \int_{\mathbb{R}^d} (\mathbf{c} \cdot \mathbf{S_0} e^{\mathbf{x}} - K)\, f_{\mathbf{X_T}}(\mathbf{x})\, d\mathbf{x}.$$

3.1 Algorithm A: Computing Price using FFT

In order to make our code converging integrating over sparse grid, it is necessary to manipulate the integrand function, increasing its regularity. By a straightforward manipulation (2.2), we can write:

$$V(\mathbf{S}_0, K) = \int_{\mathbb{R}^{d-1}} \left(\int (c_1 S_{10} e^x - K')^+ \; f_{X_{1T}} \left(x | \, \hat{\mathbf{X}}_T = \hat{\mathbf{x}} \right) dx \right) \; f_{\hat{\mathbf{X}}_T}(\hat{\mathbf{x}})\, d\hat{\mathbf{x}}.$$

3.1.1 One-Dimensional Price

We compute the 1-dimensional inner integral

$$O(S_{10}, K) = \int (c_1 S_{10} e^x - K')^+ \; f_{X_{1T}} \left(x | \, \hat{\mathbf{X}}_T = \hat{\mathbf{x}} \right) dx \tag{3.1}$$

using a quadrature formula adapted to the discontinuity $D = \log\left(\frac{K'}{c_1 S_{10}}\right)$. In order to evaluate the conditional expected value, we need to compute the normalization coefficient:

$$B = \int_{-\infty}^{+\infty} f_{\mathbf{X}_T}(x, \hat{\mathbf{x}})\, dx \tag{3.2}$$

The result of the computation (3.1) is therefore:

$$\begin{cases} \frac{1}{B} \cdot \int_D^{+\infty} f_{\mathbf{X}_T}(x, \hat{\mathbf{x}})\ \left(c_1 S_{10} e^x - K'\right) dx & \text{if } K' \geq 0, \\ \frac{1}{B} \cdot \left(\int_{-\infty}^{+\infty} f_{\mathbf{X}_T}(x, \hat{\mathbf{x}})\ c_1 S_{10} e^x dx \right) - K' & \text{if } K' < 0. \end{cases} \tag{3.3}$$

In 3.1.1 are represented the results of (3.3) depending on K', in a 2-dimensional example.

Remark 3.1. *Since for the model we discuss in this thesis the pdf decays faster than the inverse of the payoff functions, we can approximate the integrals over an infinite regions by cutting the integration region.*

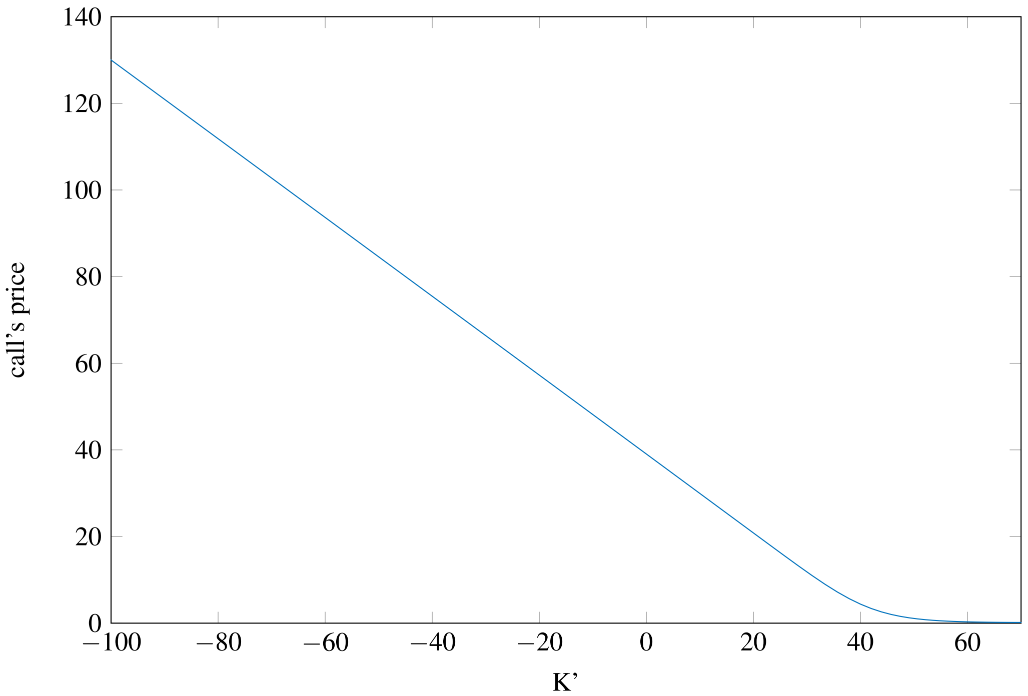

Figure 3.1: price of a 1-dim call option depending on $K' = K - S_{20}e^{X_{2T}}$, between $X_{2T} = 1.5$ and $X_{2T} = -1.5$.

3.1.2 (d-1)-Dimensional Integration

In order to use the Smolyak rule to compute the integral

$$\int_{\mathbb{R}^{d-1}} O(S_{10}, K) f_{\hat{\mathbf{X}}_T}(\hat{\mathrm{x}})\, d\hat{\mathrm{x}}, \tag{3.4}$$

we need to know the probability density function of the (d-1)-dimensional process, $f_{\hat{\mathbf{X}}_T}(\hat{\mathrm{x}})$. The less expensive way to compute it is by applying the fft to the characteristic function.

Definition 3.2. *The characteristic function of the stochastic process $\boldsymbol{X}_T$ is defined by:*

$$\varphi_T(\boldsymbol{z}) = \mathbb{E}\left[e^{i\boldsymbol{z}^T\boldsymbol{X}_T}\right] = \int_{\mathbb{R}^d} e^{i\boldsymbol{z}'\boldsymbol{x}} pdf(\boldsymbol{x}) d\boldsymbol{x} \tag{3.5}$$

Consequently, we can compute the pdf from the characteristic function $\varphi(\boldsymbol{z})$:

$$pdf(\boldsymbol{x}) = \frac{1}{(2\pi)^d} \int_{\mathbb{R}^d} e^{-i\boldsymbol{z}'\boldsymbol{x}} \varphi(\boldsymbol{z}) d\boldsymbol{z} \tag{3.6}$$

3.1.3 FFT to Compute the (d-1)-Dimensional pdf

The d-dimensionaf Fourier transform is defined by

$$\mathcal{F}\{g(\boldsymbol{x})\}(\boldsymbol{z}) = \int_{-\infty}^{+\infty} \cdots \int_{-\infty}^{+\infty} e^{i\boldsymbol{z}^T\boldsymbol{x}} g(\boldsymbol{x}) d\boldsymbol{x}, \tag{3.7}$$

where $\boldsymbol{z} \in \mathbb{R}^d$. The function $g(\boldsymbol{x})$ is called Fourier integrable if $\mathcal{F}\{g(\boldsymbol{x})\}(\boldsymbol{z})$ exists and is analytic for all $\boldsymbol{z} \in \mathbb{R}^d$. The inverse Fourier transform is consequently defined by:

$$g(\boldsymbol{x}) = \frac{1}{(2\pi)^d} \int_{-\infty}^{+\infty} \cdots \int_{-\infty}^{+\infty} e^{-i\boldsymbol{x}^T\boldsymbol{z}} \mathcal{F}\{g(\boldsymbol{x})\}(\boldsymbol{z}) d\boldsymbol{z} \tag{3.8}$$

Remark 3.3. *Because of the scalar product in* (3.5)*, we can apply the inverse Fourier transform to the d-dimensional characteristic function with input* $[0, \hat{\mathbf{X}}_T]$ *and compute the (d-1)-dimensional pdf.*

Let's focus on the 2-dimensional case, and consider the grid

$$\boldsymbol{x_p} = (\boldsymbol{x}_0 + \boldsymbol{p}\Delta\boldsymbol{x}) = (x^1_{p^1}, x^2_{p^2}) = (x^1_0 + p^1\Delta x^1, x^2_0 + p^2\Delta x^2)$$

and the corresponding grid in the dual space

$$\boldsymbol{z_j} = (\boldsymbol{z}_0 + \boldsymbol{j}\Delta\boldsymbol{z}) = (z^1_{j^1}, z^2_{j^2}) = (z^1_0 + j^1\Delta z^1, z^2_0 + j^2\Delta z^2)$$

If we have the characteristic function $\varphi(\boldsymbol{z})$ and we want to compute the pdf value in a node $\boldsymbol{x_p}$, we can write:

$$\begin{aligned}
pdf(\boldsymbol{x_p}) &= \frac{1}{(2\pi)^2}\int_{\mathbb{R}^2} e^{-i\boldsymbol{z}'\cdot\boldsymbol{x_p}}\varphi(\boldsymbol{z})d\boldsymbol{z} = \frac{1}{(2\pi)^2}\int_{\mathbb{R}^2} e^{-i\left(z^1\cdot x^1_{p^1}+z^2\cdot x^2_{p^2}\right)}\varphi(z^1,z^2)dz^1dz^2 = \\
&= \frac{1}{(2\pi)^2}\int_{\mathbb{R}}\sum_{j^1=0}^{n^1-1} e^{-i\left(z^1_{j^1}\cdot x^1_{p^1}+z^2\cdot x^2_{p^2}\right)}\varphi(z^1_{j^1},z^2)\Delta z^1 dz^2 = \\
&= \frac{1}{(2\pi)^2}\int_{\mathbb{R}}\sum_{j^1=0}^{n^1-1} e^{-i\left(\left(z^1_0+j^1\Delta z^1\right)\cdot x^1_{p^1}+z^2\cdot x^2_{p^2}\right)}\varphi(z^1_{j^1},z^2)\Delta z^1 dz^2 = \\
&= \frac{1}{(2\pi)^2}\sum_{J^1=0}^{n^1-1} e^{-i\left(z^1_0+j^1\Delta z^1\right)x^1_{p^1}}\sum_{J^2=0}^{n^2-1} e^{-i\left(z^2_0+j^2\Delta z^2\right)x^2_{p^2}}\varphi\left(z^1_{j^1},z^2_{j^2}\right)\Delta z^1\Delta z^2
\end{aligned}$$

since, given the vector of indexes from $\mathbf{0}$ to $\boldsymbol{n}-\mathbf{1} = (n^1-1, n^2-1)$

$$\boldsymbol{j} = (j^1, j^2) \quad \text{and} \quad \boldsymbol{p} = (p^1, p^2),$$

the bi-dimensional FFT is defined as

$$X_p = \sum_{j^1=0}^{n^1-1}\sum_{j^2=0}^{n^2-1} x_{\boldsymbol{j}} e^{(-i2\pi\boldsymbol{p}\cdot(\boldsymbol{j}./\boldsymbol{n}))}, \tag{3.9}$$

we can continue writing

$$\begin{aligned}
pdf(\boldsymbol{x_p}) &= \frac{1}{(2\pi)^2}\sum_{j^1=0}^{n^1-1} e^{-1\left(z^1_0+j^1\Delta z^1\right)x^1_{p^1}}\sum_{j^2=0}^{n^2-1} e^{-1\left(z^2_0+j^2\Delta z^2\right)x^2_{p^2}}\varphi\left(z^1_{j^1},z^2_{j^2}\right)\Delta z^1\Delta z^2 = \\
&= \frac{1}{(2\pi)^2}\sum_{j^1=0}^{n^1-1} e^{-1\left(z^1_0+j^1\Delta z^1\right)\left(x^1_0+p^1\Delta x^1\right)}\sum_{j^2=0}^{n^2-1} e^{-1\left(z^2_0+j^2\Delta z^2\right)\left(x^2_0+p^2\Delta x^2\right)}\varphi\left(z^1_{j^1},z^2_{j^2}\right)\Delta z^1\Delta z^2
\end{aligned}$$

in order to get a sum in the form of (3.9), we are forced to impose the following condition on the grid spacings:

$$\Delta z^i \Delta x^i = \frac{2\pi}{n^i}, \quad \text{for} \quad i = 1,2 \tag{3.10}$$

under this condition, we can continue manipulating our equation, obtaining:
(3.11)

$$\begin{aligned}
pdf(\boldsymbol{x_p}) = {} & \frac{1}{(2\pi)^2} e^{-i\left(\boldsymbol{z}_0' \cdot \boldsymbol{x_p}\right)} \Delta z^1 \Delta z^2 \sum_{j^1=0}^{n^1-1} \sum_{j^2=0}^{n^2-1} e^{-i\left(j^1 \Delta z^1 x_0^1 + j^2 \Delta z^2 x_0^2\right)} \cdot \\
& \cdot \underbrace{e^{-i\left(j^1 p^1 \frac{2\pi}{n^1}\right)} e^{-i\left(j^2 p^2 \frac{2\pi}{n^2}\right)}}_{e^{-i2\pi \boldsymbol{p} \cdot (\boldsymbol{j} ./ \boldsymbol{n})}} \varphi\left(z_{j^1}^1, z_{j^2}^2\right) = \\
& = \frac{1}{(2\pi)^2} e^{-i\left(\boldsymbol{z}_0' \cdot \boldsymbol{x_p}\right)} \Delta z^1 \Delta z^2 \sum_{j^1=0}^{n^1-1} \sum_{j^2=0}^{n^2-1} \underbrace{e^{-i\left(j^1 \Delta z^1 x_0^1 + j^2 \Delta z^2 x_0^2\right)} \varphi\left(z_{j^1}^1, z_{j^2}^2\right)}_{\text{factor } m_{\boldsymbol{pj}} \text{ to which apply the FFT}} e^{-i2\pi \boldsymbol{p} \cdot (\boldsymbol{j} ./ \boldsymbol{n})} = \\
& = \frac{1}{(2\pi)^2} e^{-i\left(\boldsymbol{z}_0' \cdot \boldsymbol{x_p}\right)} \Delta z^1 \Delta z^2 FFT\{m_{\boldsymbol{pj}}\}.
\end{aligned}$$

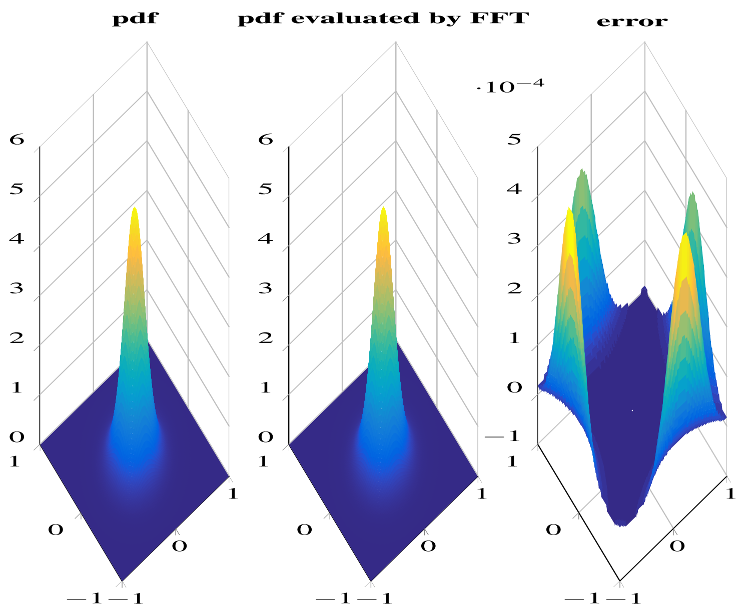

Figure 3.2: 2-dimensional CCVG model's pdf

3.1.4 The Code

The algorithm is essentially composed by three main parts, each one with a certain assignment:

- build the sparse grid algorithm
- evaluate a full tensor product
- compute a unidimensional integral

The implementation results therefore in three principal functions. To make the writing of the code straightforward, we used a function based on Algorithm 8.1.1 in [8]. Given two number d and l, this function determines all the d-uples the sum of whose elements is equal to l.
A Matlab implementetion of the code is in Appendix A.

Algorithm 1 basket call option pricing using the 1-dimensional trick and integrating over sparse grid

```
Sparse grid algorithm
Require: $sgl, S_0, k, r, T, C,$ model's parameters
1: for l=max(dim-1,,sgl-dim) to sparse grid level sgl do
2:     compute the coeff(sgl,dim,l)
3:     evaluate all the (dim-1)-uple $\alpha_i : |\alpha_i|_1 = l$
4:     for every $\alpha_i$ do
5:         if it is not already stored, compute the full tensor product by evaluatefulltensor and store it
6:         integral ← integral+coeff · full tensor product($\alpha_i$)
7:     end for
8: end for
9: option's price ←integral
```

Algorithm 2 evaluates the full tensor product corresponding to $\boldsymbol{\alpha}$ of the function $\mathrm{O}(K, S_{2_T}, \dots, S_{d_T}) \cdot pdf(S_{2_T}, \dots, S_{d_T})$

evaluatefulltensor

```
Require: $sgl, S_0, K, r, T, C,$ model's parameters
    build the grids with $N = 2^{\alpha(i)}$ nodes in every dimension in the real space
    and in the Fourier one, with the condition $\boldsymbol{\Delta z} = 2\pi/\boldsymbol{n}. \cdot \boldsymbol{\Delta x}$
2:  $\sharp$ = prod($\boldsymbol{n}$) % number of points in each grid
    for j=1:$\sharp$ do
4:      if the 1-dimensional integral is still to be computed then
            compute it by oneintegral and store it
6:      end if
        M(j)$\leftarrow \varphi(0, \boldsymbol{x}_j) \cdot e^{-i\boldsymbol{z}_j \cdot \boldsymbol{x}_0}$
8:      SUPP(j)$\leftarrow \frac{1}{(2\pi)^d} \cdot prod(\boldsymbol{\Delta z}) \cdot e^{-i\boldsymbol{z}_0 \cdot \boldsymbol{x}^j} \cdot$1-dimensional integral$\cdot prod(\boldsymbol{\Delta x})$
    end for
10: fulltensor=$sum(SUPP. \cdot (fft(M)))(:)$
```

Algorithm 3 given the dim-1 final components corresponding to the node $\boldsymbol{x}_j$, this function computes the corresponding call's price, considering the new K' and the model's parameters.

oneintegral

```
Require: $S_0, K, r, T, C,$ model's parameters
  $K' = K - \sum_{j=2}^{dim} C_j X_j$
  if $K' > 0$ then
      oneprice=$\mathbb{E}\left(C_1 S_{1_T} - K'\right)^+$ adapting the integration to the discontinuity D
  else
      oneprice=$(C_1 \mathbb{E}(S_{1_T}) - K')$
  end if
```

3.2 Algorithm B: Using the Normality of the CCGM's conditional pdf

As expressed in section 1.4.2, in a CCVG process the random vector $\mathbf{X}(t)|G_t = g$ is Gaussian with mean $g\theta$ and variance covariance matrix $g\Sigma$. Therefore the density of $\mathbf{X}(t)$ can be obtained integrating:

$$CCVG_{pdf}(\mathbf{X};\nu,\Sigma,\theta) = \int_0^{+\infty} \mathcal{N}_{pdf}(\mathbf{X};\theta g, g\Sigma) f_{\text{gamma}}\left(g;\frac{t}{\nu},\frac{1}{\nu}\right) dg. \tag{3.12}$$

Given a payoff function $\mathcal{P}(\boldsymbol{X}_T)$, we can embed the equation (3.12) in (2.1) and write:

$$\begin{aligned} V(\boldsymbol{X}_0) &= \int_{\mathbb{R}^d} \mathcal{P}(\mathbf{x}) \cdot CCVG_{\text{pdf}}(\boldsymbol{x}) d\mathbf{x} = \\ &= \int_{\mathbb{R}^d} \mathcal{P}(\mathbf{x}) \cdot \int_0^{+\infty} \mathcal{N}_{pdf}(\mathbf{x};\theta g, g\Sigma) f_{\text{gamma}}\left(g;\frac{t}{\nu},\frac{1}{\nu}\right) dg d\mathbf{x} = \\ &= \int_0^{+\infty} \int_{\mathbb{R}^d} \mathcal{P}(\mathbf{x}) \cdot \mathcal{N}_{pdf}(\mathbf{x};\theta g, g\Sigma) d\mathbf{x} f_{\text{gamma}}\left(g;\frac{t}{\nu},\frac{1}{\nu}\right) dg. \end{aligned}$$

The embedded d-dimensional integral is computed splitting it into two part as in the previous algorithm, with the difference that now the probability density function has a normal distribution. In this case we know analytically either the conditional 1-dimensional pdf and the (d-1)-dimensional pdf. Moreover, it is possible do derive a closed expression for the two cases in algorithm 3. For example, if we want to consider a basket call's payoff, we can name

$$w = log\frac{K'}{C_1 S_{1_0}}$$

and evaluate analytically the payoff's expected value, in the case $K' > 0$:

$$\begin{aligned} &\int_w^{+\infty} \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \left(C_1 S_{1_0} e^x - K'\right) dx = \\ &= K'\left(\phi(K',\mu,\sigma) - 1\right) + \int_w^{+\infty} C_1 S_{1_0} e^x \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} dx = \\ &= K'\left(\phi(K',\mu,\sigma) - 1\right) + C_1 S_{1_0} e^{\frac{\sigma^2+2\mu}{2}} \int_w^{+\infty} \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-(\mu+\sigma^2)^2}{2\sigma^2}} dx = \\ &= K'\left(\phi(K',\mu,\sigma) - 1\right) + C_1 S_{1_0} e^{\frac{\sigma^2+2\mu}{2}} \left(1 - \phi\left(K',\mu+\sigma^2,\sigma\right)\right). \end{aligned} \tag{3.13}$$

Analogously, in the case $K' \leq 0$ we can write:

(3.14)

$$-K' + \int_{-\infty}^{+\infty} \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} C_1 S_{1_0} e^x dx =$$
$$= -K' + C_1 S_{1_0} e^{\frac{\sigma^2+2\mu}{2}} \int_{-\infty}^{+\infty} \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-(\mu+\sigma^2))^2}{2\sigma^2}} e^x dx = -K' + C_1 S_{1_0} e^{\frac{\sigma^2+2\mu}{2}}.$$

3.2.1 The Code

The algorithm embeds the one described in subsection 3.1.4. It includes the three main parts:

- build the sparse grid algorithm
- evaluate full tensor product
- compute a unidimensional integral

The first one corresponds to algorithm 5, while in the other two parts the implementation take in account that now the multidimensional density is normal, and therefore we don't need to apply the FFT to the characteristic function anymore.

Algorithm 4 basket call option pricing computing 1-dimensional grid's values integrating over sparse grid.

```
CCVG pricing algorithm
Require: $sgl, S_0, k, r, T, C,$ model's parameters
  discretize the gamma function's support in $\sharp_g$ points
  for i=1:$\sharp_g$ do
    g(i)=gammapdf($g_i$, $\frac{T}{\nu}\frac{1}{\nu}$)
    p(i)=sparsegridalgo(sgl,$S_0, K, r, SIGMA.*g_i$,,T,C,$g_i.*\theta$)
  end for
  option's price=quadraturerule(g(i),p(i))
```

Algorithm 5 basket call option pricing using the 1-dimensional trick and integrating over sparse grid

Sparse grid algorithm

```
Require: $sgl, S_0, k, r, T, C,$ model's parameters
1: for l=max(dim-1,sgl-dim) to sparse grid level sgl do
2:     compute the coeff(sgl,dim,l)
3:     evaluate all the (dim-1)-uple $\alpha_i : |\alpha_i|_1 = l$
4:     for every $\alpha_i$ do
5:         if it is not already stored, compute the full tensor product by evaluatefulltensor and store it
6:         integral ← integral+coeff · full tensor product($\alpha_i$)
7:     end for
8: end for
9: option's price =integral
```

Algorithm 6 evaluates the full tensor product corresponding to $\boldsymbol{\alpha}$ of the function O$(K, S_{2_T}, \ldots, S_{d_T}) \cdot pdf(S_{2_T}, \ldots, S_{d_T})$

evaluatefulltensor

```
Require: $\alpha, sgl, S_0, K, r, T, C,$ model's parameters
Ensure: full tensor product
  build the grid with $n = 2^{\alpha(i)}$ nodes in every dimension i=$1, \cdots, \mathrm{dim} - 1$:
  $\sharp$ = prod($\boldsymbol{n}$) % number of grid's points
  for j=1:$\sharp$ do
      M(j)← $pdf(\boldsymbol{x}_j)\cdot$ oneintegral ($\boldsymbol{x}_j, K,$ model's parameters)
  end for
  fulltensor=$sum(M(:))$
```

Algorithm 7 given the dim-1 final components corresponding to the node $\boldsymbol{x}_j$, this function computes the corresponding call's price, considering the new K' and the model's parameters.

oneintegral

```
Require: $S_0, K, r, T, C,$ model's parameters
  $K' = K - \sum_{j=2}^{dim} C_j X_j$
  compute the conditioned 1-dimensional $\mu$
  compute the conditioned 1-dimensional $\sigma$
  if $K' > 0$ then
    oneprice is computed by (3.13)
  else
    oneprice is computed by (3.14)
  end if
```

Chapter 4

Numerical Results

In this chapter we show the outputs of the two algorithms presented in chapter 3. Although the second algorithm embeds the first one, it is quite difficult to make a fair comparison. In fact, in the first one every evaluation is very expensive, while in Algorithm B the closed formulas reported in (3.3) make the computation quite faster. On the other hand, in Algorithm B there are some inputs parameters, not present in Algorithm A, that can compromise the quality of the rate of convergence. Besides, Algorithm A is more optimised, as it includes hashmaps for direct access to already computed data and a more sophisticated parallelization of the code has been implemented. In Figure 4.1 is shown a comparison between the running time of Algorithm A in a 3-dimensional case and the time that Algorithm B needs to compute a 3-dimensional integral in a single 1-dimensional point.
Given that we did not include in this thesis an analysis of the errors in the two algorithms, we refer to a reference value, computed either "overkilling the algorithm" and running a Monte Carlo simulation for an amount of time such that we can reasonably trust the result.
We run the codes in different dimensions. In the 3-dimensional and in the 5-dimensional cases we give as inputs the parameters of the model fitted by Luciano and Schoutens in [13].

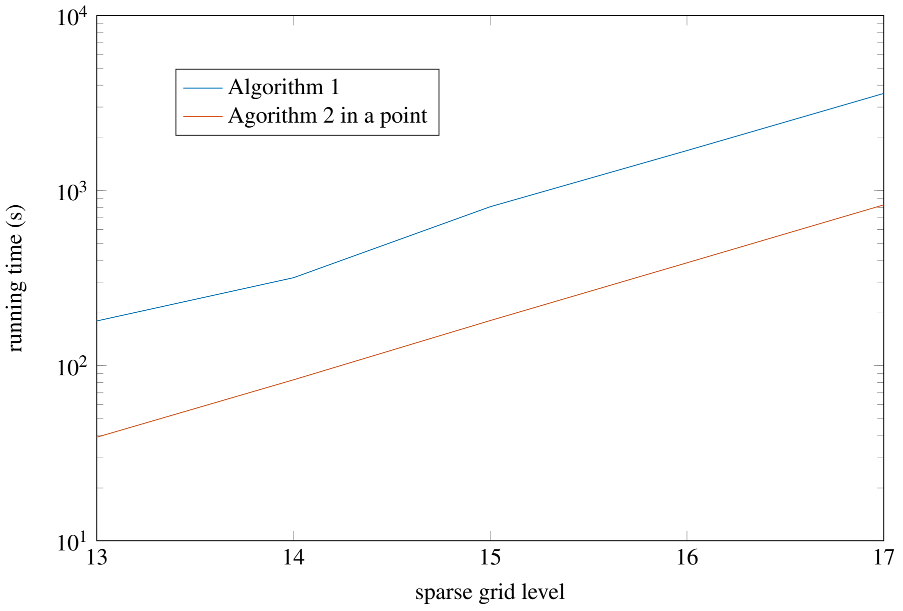

Figure 4.1: comparison between the running time in a 3-dimensional analogous computation.

4.1 The Importance of the cut-off

As we mentioned in 3.1, we can approximate the exact result of the integration by integrating in a (d-1)-dimensional hyperrectangle. Neverthless, it is very important, in order to obtain a good result and a good rate of convergence, to choose properly the cut-off. It is a common aspect in the two algorithms, and it is illustrated in Figure 4.2.

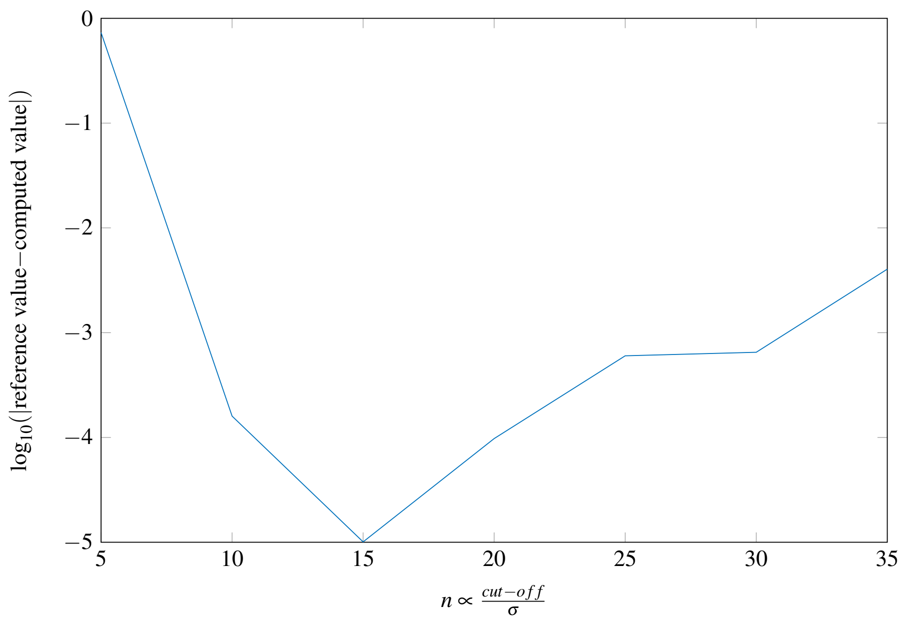

Figure 4.2: comparison between the reference value, computed by MC simulation, and the computed value depending on the cut-off in a 1-dimensional point of the example in subsection 4.2.2

.

4.2 A 3-Dimensional Example

The model's parameters are:

- $S_0 = [10, 5, 15]$;
- $C = [2, 3, 1]$;
- $T = 3$;
- $K = 45$;
- $\nu = 0.7$;
- $\theta = [-0.1, -0.2, -0.01]$;

- $\Sigma = \begin{bmatrix} 0.12 & 0.036 & 0.018 \\ 0.036 & 0.27 & 0.0405 \\ 0.018 & 0.0405 & 0.27 \end{bmatrix}$

4.2.1 Algorithm A

Chosen a cut-off value, the result of the computation goes towards the reference value (computed overkilling Algorithm B and corroborating the result running a MC simulation) as the sparse grid level increases. The $\log_{10}$ of the absolute value of the difference between the two evaluations is represented in Figure 4.3 depending on the sparse grid level sgl. It is possible to observe that the cut-off we chose to run the code makes every computation with a sparse grid level higher that 13 essentially useless. How the running time is related with the sparse grid level is represented in Figure 4.4.

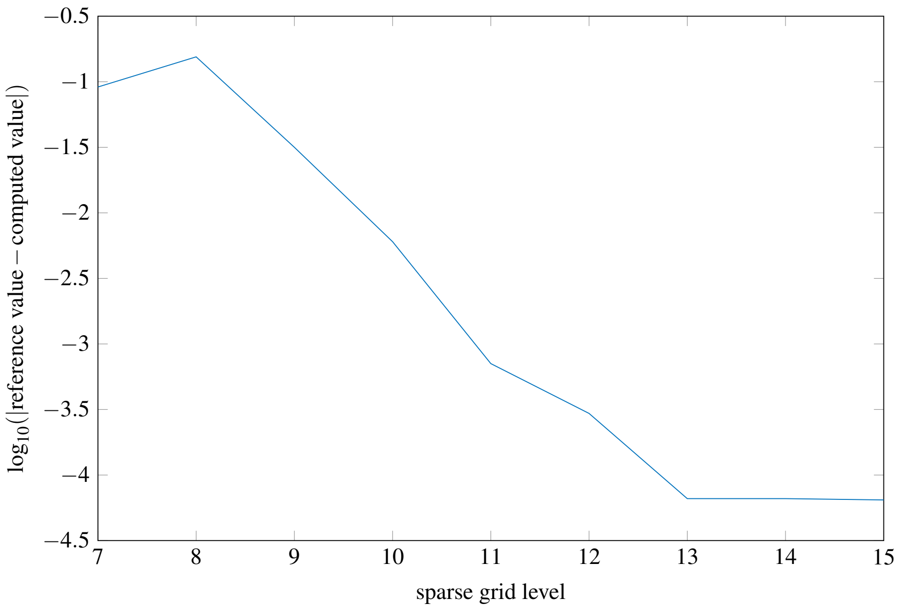

Figure 4.3: comparison between the reference value and the computation, depending on the sparse grid level sgl.

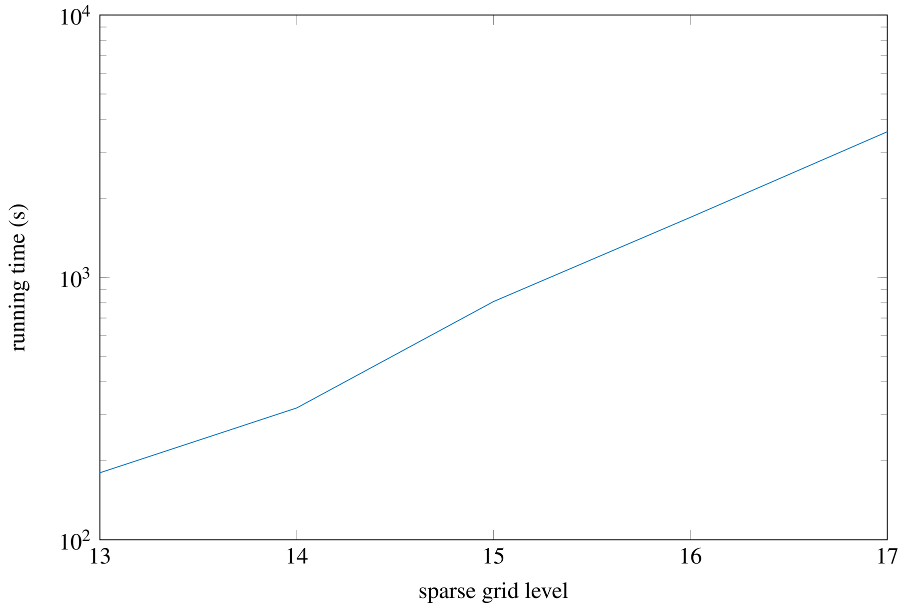

Figure 4.4: the running time grows exponentially as the sparse grid level increases.

4.2.2 Algorithm B

Algorithm B requires to compute a multidimensional integral in every node of a 1-dimensional grid. These values are then used to compute a 1-dimensional integral. In Figure 4.5 is represented the convergence rate in a single 3-dimensional computation, corresponding to a 1-dimensional node.

The quality of the whole result can be improved both by adding points in the 1-dimensional computation (and therefore evaluating more d-dimensional integrals) and by increasing the number of nodes in the computation evaluated in every point. In Figure 4.6 is represented how the result steers towards the reference value as the number of points increases. On the other hand, Figure 4.7 shows how increasing the sparse grid level affects the results, keeping the number of 1-dimensional points fixed.
Figure 4.8 shows how both the sparse grid level in every 1-dimensional point and the number of 1-dimensional integration points affects the time needed in the whole computation.

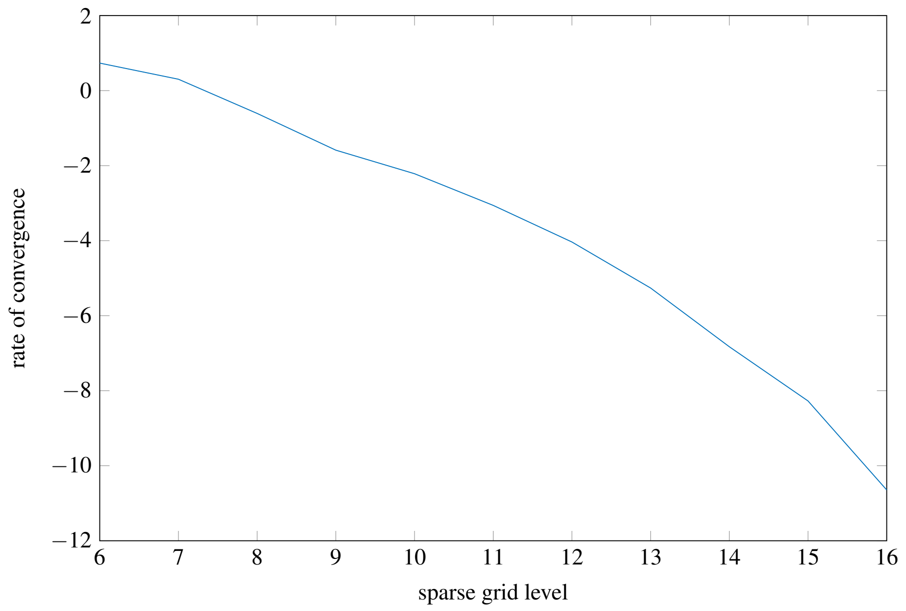

Figure 4.5: rate of convergence in a single point, defined as the $\log_{10}$ of the difference of two consequent computations, depending on the sparse grid level.

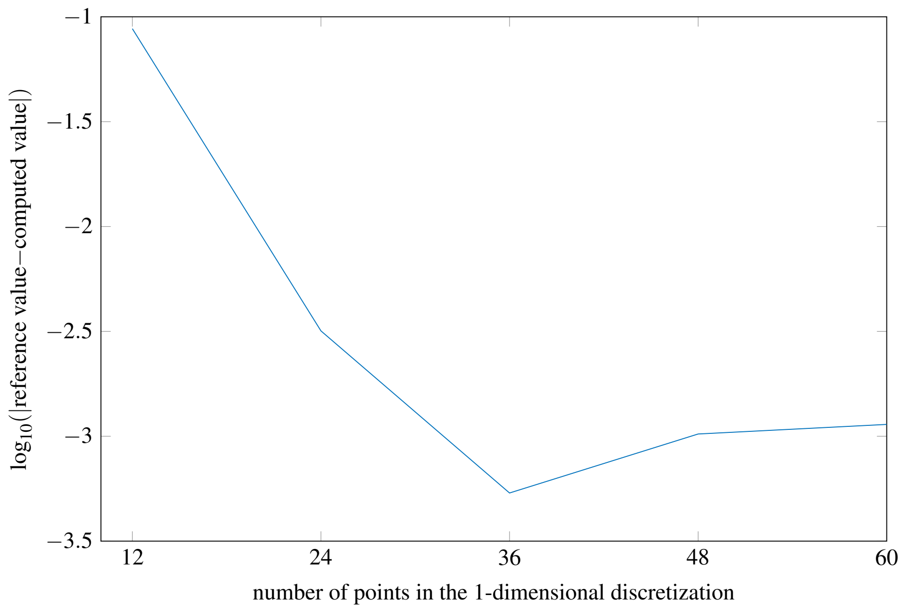

Figure 4.6: comparison between the computed price and the reference one increasing the number of points, keeping sgl=9.

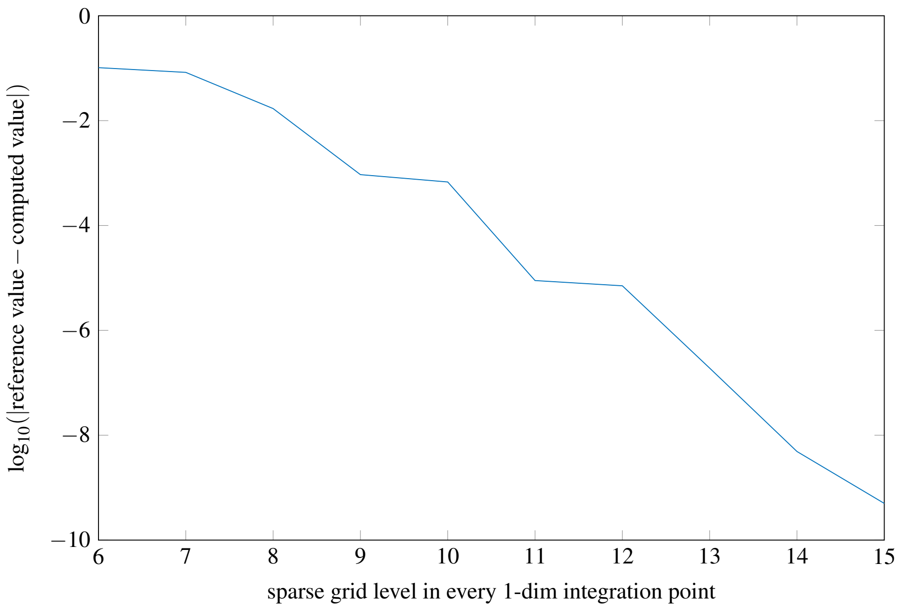

Figure 4.7: comparison between reference value (computed overkilling the algorithm) and computed value increasing the sgl in 60 integration points.

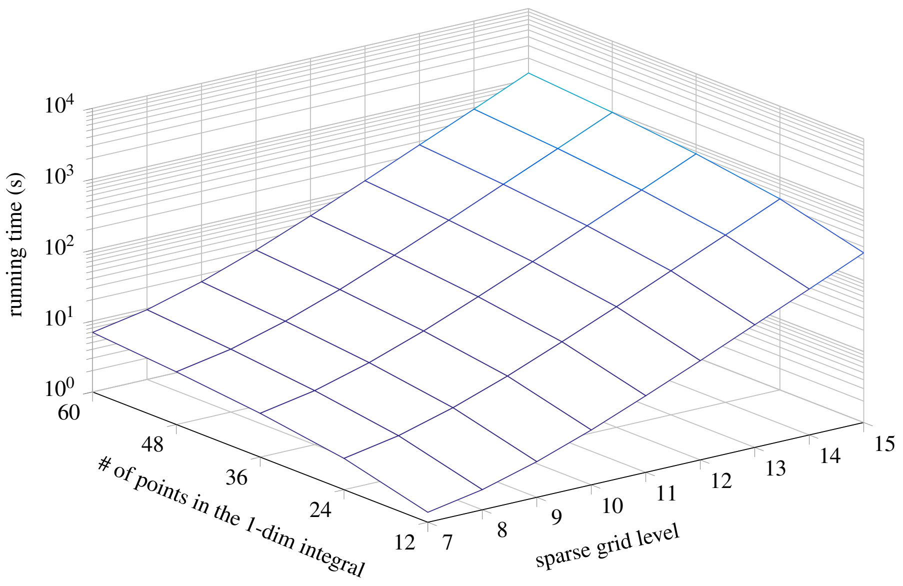

Figure 4.8: running time vs sparse grid level in every point vs number of points.

4.3 A 5-Dimensional Example

The model's parameters are:

- $S_0 = [5, 6, 10, 9, 11]$;
- $C = [1, 1, 2, 1, 1]$;
- $T = 2$;
- $K = 47$;
- $\nu = 0.4$;
- $\theta = [-0.03, -0.02, -0.01, -0.02, -0.03]$;
- $\Sigma = \begin{bmatrix} 0.08 & 0.012 & 0.0006 & 0 & 0 \\ 0.012 & 0.045 & 0 & 0 & 0 \\ 0.0006 & 0 & 0.002 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0.00648 \end{bmatrix}$

4.3.1 Algorithm A

The reference value has been computed running a MC simulation for several hours.

As we can see in Figure 4.9, in 5 dimensions we already need a high sparse grid level to obtain a value near to the reference one. Moreover, choose the right cut-off becomes increasingly more important. In general, the Monte Carlo simulation is already less expensive than the algorithm, as the sgl=13 computation takes around 3.4 hours.

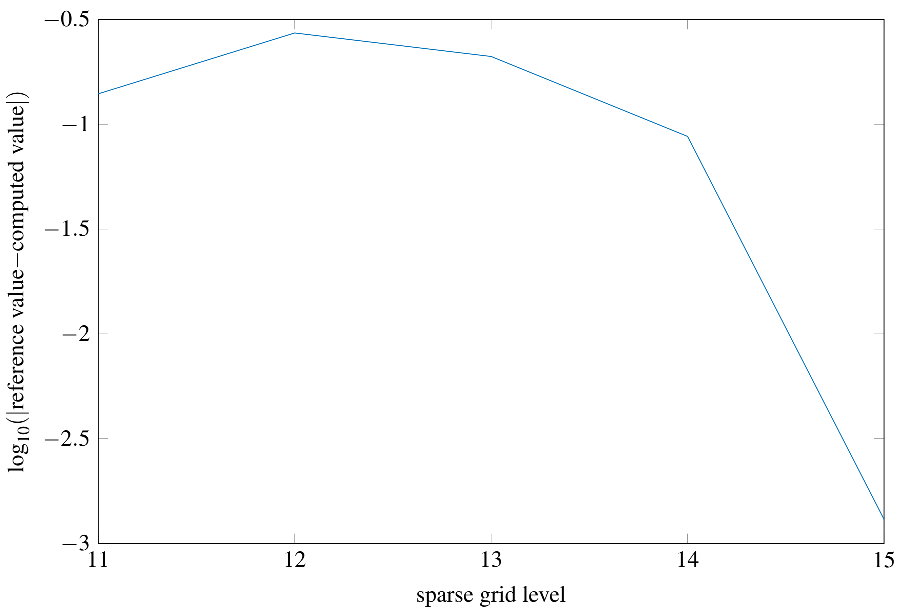

Figure 4.9: comparison of the computed value with the reference one, depending on the sparse grid level sgl, in a 5-dimensional simulation.

4.3.2 Algorithm B

As in the 3-dimensional example, we can steer towards the reference result both increasing the number of nodes to discretize the 1-dimensional interval (Figure 4.10) or increasing the sparse grid level in every point (Figure 4.11). The running time of the algorithm depending on the number of points used to discretize the 1-dimensional interval and on the sgl is represented in Figure 4.12. This algorithm reveals to be faster than the Monte Carlo simulation even in a 5-dimensional example, as we need to run the MC simulation for 4.5 hours (computed in parallel using 12 cores) to obtain a result x : the exact $\dot{x} \in (x - 0.0004, x + 0.0004)$ with a confidence of the 99%.

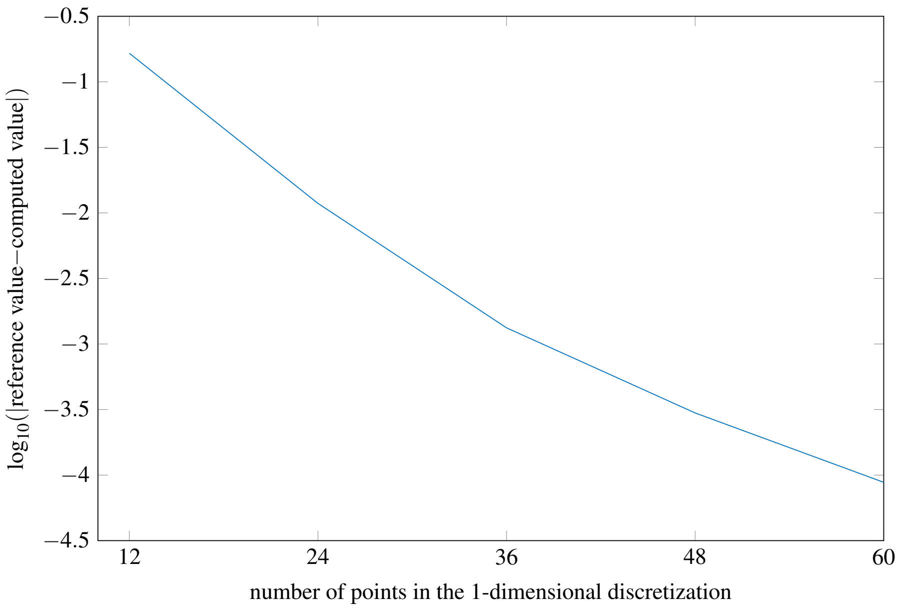

Figure 4.10: comparison with the reference value increasing the number of points, keeping sgl=14.

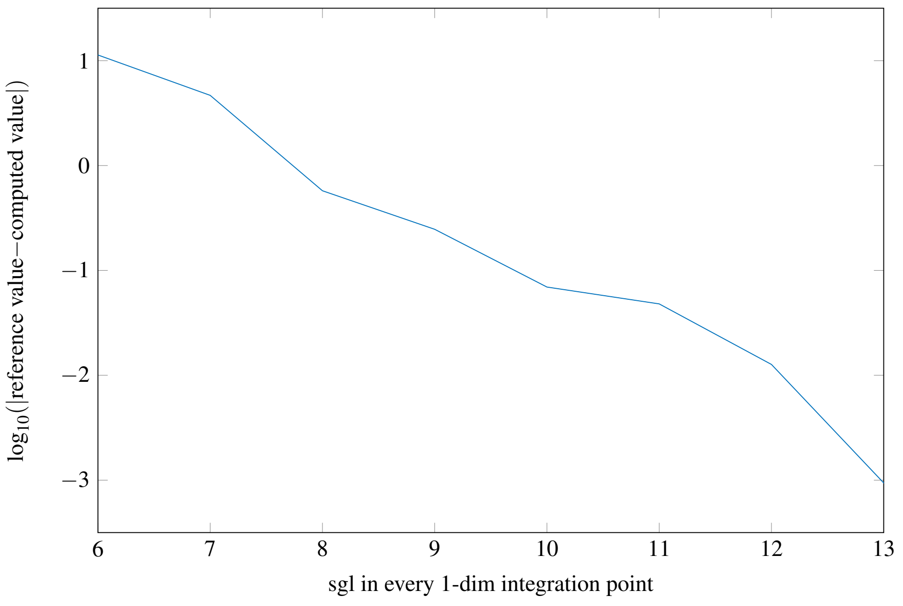

Figure 4.11: comparison with the reference value increasing the sgl in 48 integration points.

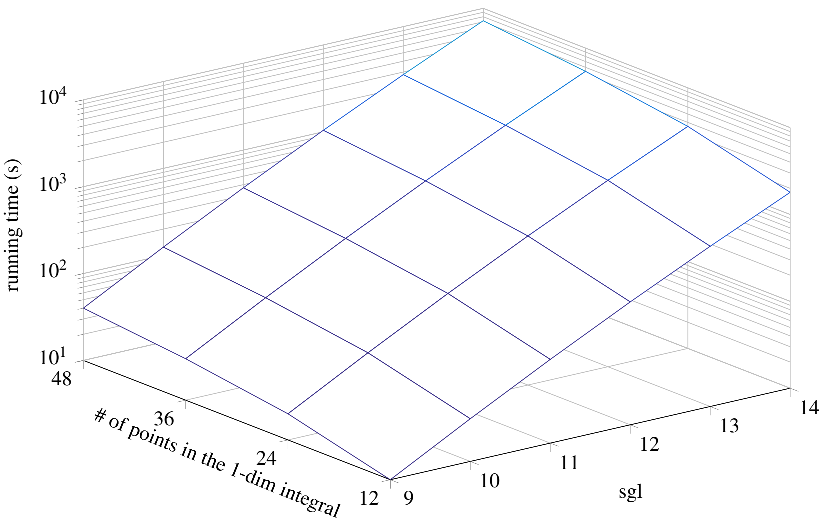

Figure 4.12: running time vs sparse grid level in every point vs number of points.

4.4 A 7-Dimensional Example

The model's parameters are:

- $S_0 = [10, 5, 12, 8, 7, 15, 9]$;
- $C = [1, 2, 1, 1, 1, 1, 1]$;
- $T = 1$;
- $K = 70$;
- $\nu = 0.1$;
- $\theta = [0.004, 0.002, 0.001, 0.002, 0.003, 0.005, 0.002]$;

- $$\Sigma = \begin{bmatrix} 0.04 & 0.006 & 0 & 0 & 0 & 0 & 0 \\ 0.006 & 0.009 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0.009 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.04 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0.001 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0.04 & 0.002 \\ 0 & 0 & 0 & 0 & 0 & 0.002 & 0.01 \end{bmatrix}$$

4.4.1 Algorithm A

Algorithm A proved to be unable to solve the problem in a reasonable amount of time, as it is shown in Figure 4.13. The simulation with sparse grid level 13 takes around 6000 seconds, and the method is therefore much less efficient than a Monte Carlo simulation.

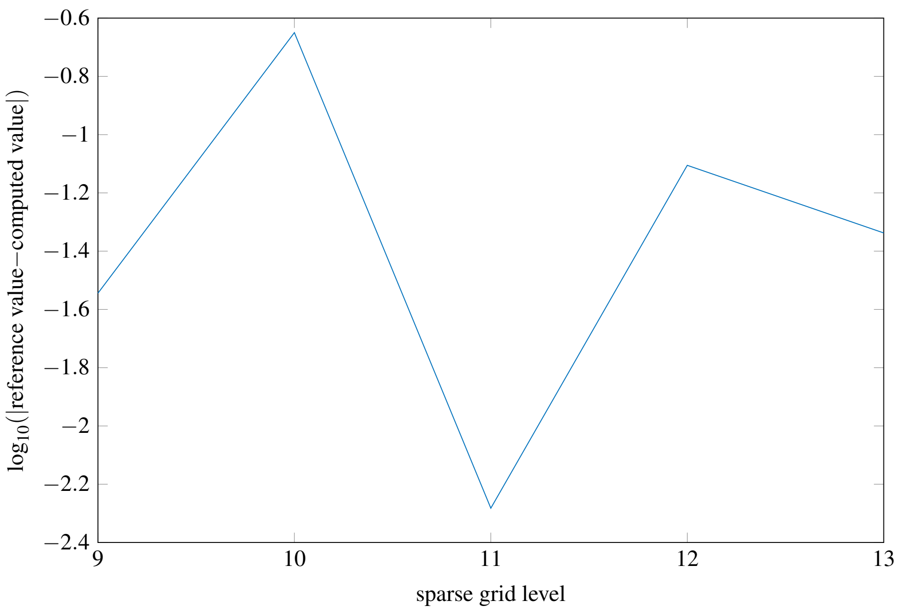

Figure 4.13: comparison with the reference MC value and the computation, depending on the sparse grid level sgl, in a 7-dimensional simulation.

4.4.2 Algorithm B

Even Algorithm B reveals to be unsuitable to compute the 7-dimensional problem. Either adding number of points in the 1-dimensional grid (Figure 4.14) or increasing the sparse grid level in every points (Figure 4.15), we are not able to obtain a satisfying result in a reasonable time. These verifications lead to our suggestions about the further works.

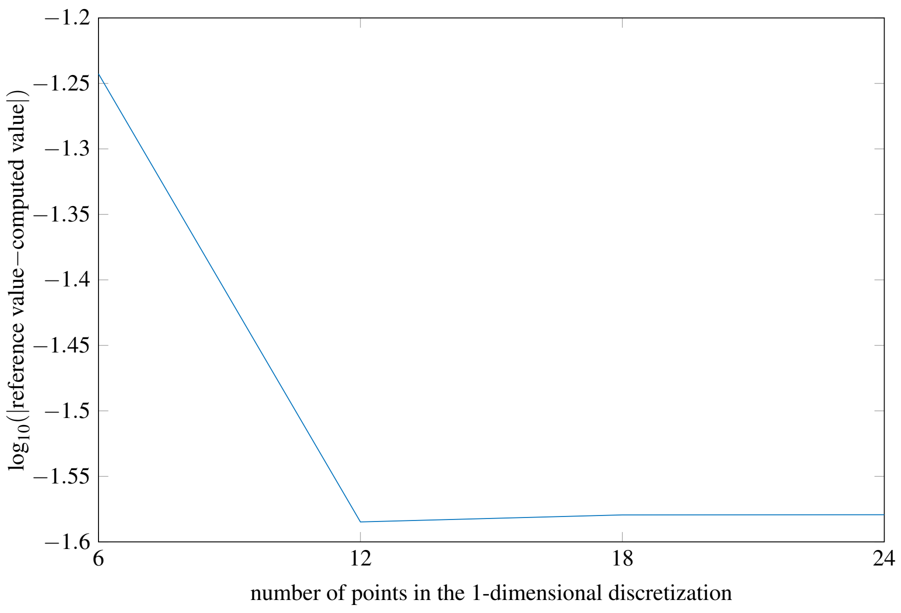

Figure 4.14: comparison between the MC evaluated value and the computed one, increasing the number of points with sgl=14.

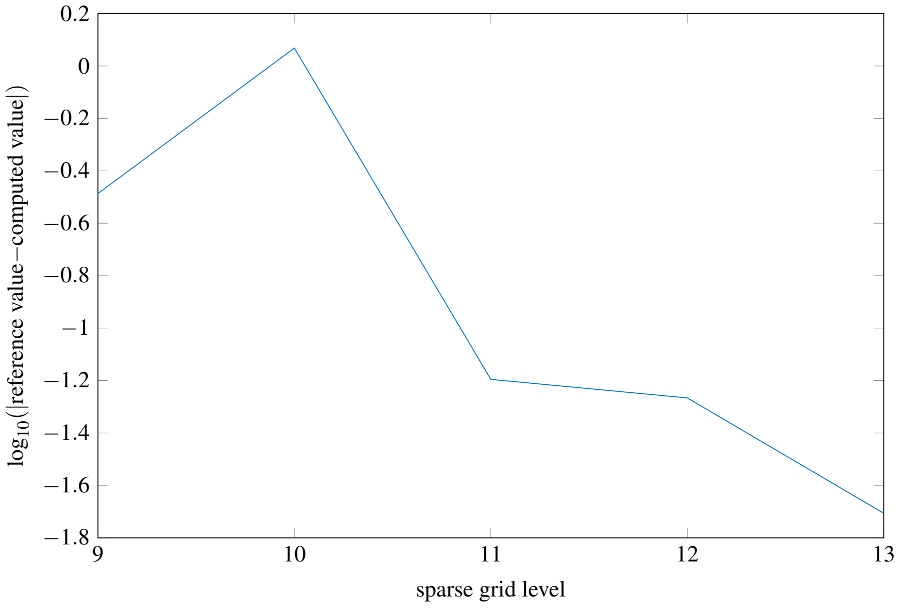

Figure 4.15: comparison with the MC value depending on the sparse grid level in 12 integration points.

Conclusions

We formulated two numerical algorithms to evaluate basket options' prices seen like expected values of the payoff given a so-called risk-neutral measure, considering the Common Clock Gamma Model introduced by Madan and Seneta in [15]. E. Luciano and W. Schoutens calibrated this model in [13], in the particular case in which Σ is a diagonal matrix, providing a good fit.

By the fundamental theorem of option pricing, our problem has been reduced to the computation of an expected value, or what is the same, to the computation of a multidimensional integral. To overcome the curse of dimensionality in the mentioned integral we used sparse grid in the form introduced by Smolyak in [18]. As the sparse grid integration requires a special class of integrand functions, we manipulated the d-dimensional integral using the law of total expectation to split it into two parts. In this way, our integrand function gained the needed regularity.
The general method implemented in Algorithm A requires to know both the characteristic function and the probability density function of the multidimensional stochastic process. In every evaluation it requires to evaluate the FFT of the probability distribution and to compute two numerical integrals, and for these reasons it is quite expensive from a computational point of view. It works fine in not too high dimensions if we choose the (d-1)-dimensional interval of integration in a proper way. In fact, to choose a too high (d-1)-dimensional interval implies a rate of convergence too slow, while the opposite choice renders in a faster convergence to a wrong result. This aspect is common between the two algorithms.

As Algorithm B does not involve the FFT computation, and every evaluation admits a closed form, it runs quite fast even if it has not been optimized as much as the previous one. Nevertheless, it requires an additional 1-dimensional integration, and to choose the evaluation points in a proper way is very important to obtain a satisfying result. Unlike Algorithm A, choosing in a proper way the interval of integration Algorithm B reveals to converge faster than the Monte Carlo simulation even in the five-dimensional example we run.

Further Work

The implementation of the two algorithms brought several difficulties. We have dealt with many of them, but due to time restrictions some of these problems still need to be addressed.

Estimation of the Error

As every algorithm involves several numerical computations, it is necessary to estimate in a proper way what is the error we are committing running the two methods, and how they depend on the input parameters, like for example the length of the cut-off. F. Crocce, Y. Häppölä and R. Tempone did a similar investigation in 1 dimension in [5].

Use more Efficient Integration Rules

In our work, we only use simple rectangular integration rules. To change integration rule and the respective grid, maybe involving adaptive methods, could surely entail a more efficient computation.

Algorithm B: Optimize the 1-dimensional Integral

In Algorithm B we compute a d-dimensional integral in every point of a 1-dimensional grid. As every computation is expensive, it is important in order to increase the cost-effectiveness of the method, to choose that points in a proper way. Moreover, it should be interesting to adapt the sparse grid level to the importance that every point has in the computation of the 1-dimensional integral.

Appendices

Appendix A

Matlab Code: Algorithm A

```
function ...
    [price]=FFT_VGmodel(sgl,S_0,K,r,SIGMA,ISIGMA,T,C,mu,theta)

% appreciates the value of an European call in multi-Dim
% in the VG Model using Smolyak sparse grid.
% INPUT:
%     sgl:                 sparse grid level: max 1-norm ...
    that the
%                          alfa-vector can assume
%     S_0                  1xdim vector, initial prices of ...
    underlyings
%     K                    strike price
%     r                    risk-free rate
%     SIGMA                dim x dim SIGMA matrix of the ...
    brownian
%                          motion
%     T                    expiration date
%
%     C                    1xdim vector, c(i) exprimes the ...
    quantity
%                          of the ith underlying in my wallet
%     mu                   GV model parameter, proportional ...
   to the #
%                          of jumps per annum
%     theta                multidimensional drift
%
% OUTPUT:
%     prezzosparsegrid:    price call option
%
%



% set heaviside function 0 in the origin
oldparam = sympref('HeavisideAtOrigin',0);
w=size(S_0);
dim=w(1,2)-1;   % # underlyings -1, because of the 1-dim trick
int=0;
m=max(dim,sgl-dim+1);
%recall  global matrix

global FULLTENSORS

for l=sgl:-1:m
    disp('level')
    disp(l)
    coeff=(-1)^(sgl-l)*nchoosek(dim-1,sgl-l);  %  ...
       coefficient of sparse grid
    [a,b]=Drop(dim,l);    %   all the vectors with 1-norm ...
       = l, by row
     if (l==dim)
        a=ones(1,dim);
        b=1;
     end
    for cont=1:b     %  b = # combinations
    alfa=a(cont,:);      %alpha(i) = log2(# points in the ...
       i-th dimension)
     % check the development of the computation
        if (mod(cont,ceil(b/10))==0)
            disp('multiindex')
            disp(alfa)
        end
        key=mat2str(alfa);
        % check for product corresponding to alpha already ...
            computed
        if FULLTENSORS.containsKey(key)
         int=int+coeff*FULLTENSORS.get(key);
        else
         ft=trickfullprodVGmodel(S_0,alfa,C,K,SIGMA,ISIGMA,r,T,mu,sgl,theta);
         FULLTENSORS.put(key,ft);
         int=int+coeff*ft;
        end
    end
end
price=real(int);
```

```
function ...
    [addint]=trickfullprodVGmodel(S_0,vett_tens,C,K,SIGMA,ISIGMA,r,
T,mu,sgl,theta)
% for the given vett_tens in input, the function fullprod ...
    uses the
% corresponding full tensor product to evaluate the ...
    product of
% the payoff*probability of the payoff, computed by the
% fft from the characteristic function

% INPUT:
%           sgl                 sparse grid level
%           S_0                 1xdim vector, initial prices ...
    of underlyings
%           vett_tens           every component of the vector ...
    is the log2
%                               of the # of points in the ...
    corresponding dimension
%           K                   strike price
%           r                   risk-free rate
%          SIGMA            SIGMA matrix of the ...
   multivariate brownian
%                           motion
%          T                expiration date
%
%          C                1xdim vector, C(i) exprimes ...
   the quantity of the ith
%                           underlying in my wallet
%          mu               GV model parameter, ...
   proportional to the #
%                           of jumps per annum
%          theta            multidimensional drift
%
% OUTPUT:  addint:          full tensorial product ...
   corresponding to
%                           vett_tens
%
%




global INTMATRIX
%global FULLTENSORS

dim=size(S_0);
dim=dim(1,2)-1; %one dim-trick
n = vett_tens;
N = 2.^(n);

sigma = sqrt(diag(SIGMA))';
% let's build the grid
dx = 40 * sigma(1,2:end)./N;
dt=2*pi./(N.*dx);
x_0 =   0.5.*N.*dx;
omega_0 = 0.5.*N.*dt;
parfor l = 1:dim
```

```
        % in t(i) there are the points, in the F world, of the ...
            i-th+1 dimension
        t{l} = (omega_0(l):-dt(l):omega_0(l)-(N(l)-1)*dt(l));
        number{l} = (1:N(l));  %in N there is not the 1st ...
            dimension
        % in space(i) there are the points, in the real world, of
        %  the i-th+1 dimension (the log-values)
        space{l} = (x_0(l):-dx(l):x_0(l)-(N(l)-1)*dx(l));

    end
    % define the C.F.
    car_fun=@(z) ...
        ((1-1i*[0,z].*mu*theta+0.5*mu*[0,z]*SIGMA*[0,z]')^(-T/mu));
    %  z is a row vector
    pairs = combvec(t{:});     % every column is a vector with ...
        the coord
    %                             in the F world
    positions = combvec(number{:});   % in every column there ...
        are the
    %                                    indexes of the ...
        stroring matrix
    %                                    to whom apply the fft
    spacecoord = combvec(space{:}); % every column is a vector ...
        with the coord
    %                                  in the real world

    % prallocate for speed
    prealloc=(max(positions,[],2))';
    M=zeros(prealloc);
    SUPP=zeros(prealloc);
    evalint=0;  % # ov one-dim integral to evaluate
    k=cell(prod(N),1);
    parfor q = 1:prod(N)
        k{q}=mat2str(round(spacecoord(:,q),4));
    end

    if sum(vett_tens)==sgl  % there are one dim-integral still ...
        to evaluate
```

```
     % set the parallel 1-dim integral computing
    for q=1:prod(N)
        if INTMATRIX.containsKey(k{q})==0   % I didn't ...
           compute the onetrick yet
           evalint=evalint+1;
           combund_(evalint)=((C(1,2:end).*S_0(1,2:end))*exp(spacecoord(:,q)));
           spacecoord_{evalint}=spacecoord(:,q);
        end
    end
     if evalint>0
         % compute the one-dim integral in parallel
       parfor j=1:evalint
         IM(j)=onetrickVGM(S_0,combund_(j),C,K,SIGMA,ISIGMA,r,T,
         mu,spacecoord_{j},theta);
       end
     end
end
cont=1;
for q = 1:prod(N)
    coord=num2cell(positions(:,q));  % take the ...
        coordinates from positions
    if sum(vett_tens)==sgl
       if INTMATRIX.containsKey(k{q})==0
 %I didn't compute the one-dim integral in the previous ...
    call to this function
           INTMATRIX.put(k{q},IM(cont));
           cont=cont+1;
       end
    end
    % is the the matrix to which apply he FFT
     M(coord{:})=car_fun((pairs(:,q))')*...
     exp(-1i.*((positions(:,q)-1)'*(dt.*x_0)'));
    % let's build the matrix to multiply for FFTMATRIX to ...
        abtain the final
    % full tensor (inside there is even the payoff)
    if INTMATRIX.get(k{q})==-1
        SUPP(coord{:})=0;
    else
```

```
            SUPP(coord{:})=(1/(2*pi)^(dim).*prod(dt).*...
            exp(-1i.*omega_0*spacecoord(:,q))*(INTMATRIX.get(k{q}))*prod(dx));
        end
    end
    FFTMATRIX=fftn(M);
    PROB=SUPP.*FFTMATRIX;
    addint=real(sum(PROB(:)));
```

Appendix B

Matlab Code: Algorithm B

```
% define the input variables:
% S_0,C,r,T,K,SIGMA,nu




ndisc=48; %number points in 1-dim integral
intdisc=linspace(0.1,3.5,ndisc);
u=11; % sgl in every point
theta=[0.004;0.002;0.001];   %multidimensional drift
tic


for i=1:ndisc



g(i)=gampdf(intdisc(i),T/nu,nu);
BSextended(i)=sparsegrid(u,S_0,K,r,SIGMA.*intdisc(i),
T,C,intdisc(i).*theta');

end
pricerect=(intdisc(3)-intdisc(2))*(g*BSextended');
pricetrpz=trapz(intdisc,g.*BSextended);
```

```
% format long e
timeyuho=toc


function ...
    [prezzosparsegrid]=sparsegrid(sgl,S_0,K,r,SIGMA,T,C,mu)

% appreciates the value of an European call assuming ...
    normal
%multivariate distribution using Smolyak rule.
% INPUT:
%           sgl:        sparse grid level: max 1-norm ...
    that the alfa-vector can
%                       assume
%           S_0         1xdim vector, initial prices of ...
    underlyings
%           K           strike price
%           r           risk-free rate
%
%           T           expiration date
%
%           C           1xdim vector, c(i) exprimes the ...
    quantity of the ith
%                       underlying in my wallet
%
% OUTPUT:  prezzosparsegrid:     price call option
%
%




SGSIGMA=SIGMA(2:end,2:end);
ISGSIGMA=SGSIGMA^(-1);

w=size(S_0);
dim=w(1,2)-1;    % # underlyings -1!!!!!
```

```
int=0;
m=max(dim,sgl-dim+1);

for l=m:sgl
coeff=(-1)^(sgl-l)*nchoosek(dim-1,sgl-l);  %  ...
   coefficient of sparse grid
[a,b]=Drop(dim,l);    %  all the vectors with 1-norm ...
   = l, by row
if (l==dim)
a=ones(1,dim);
b=1;
end
for cont=1:b     %  b = # combinations

alfa=a(cont,:);
int=int+tensprod(S_0,alfa,C,K,coeff,SIGMA,r,T,mu,SGSIGMA,ISGSIGMA);
end
end
prezzosparsegrid=real(int);


function [addint]=tensprod(S_0,vett_tens,C,K,coeff,
SIGMA,r,T,mu,SGSIGMA,ISGSIGMA)
% for the given vett_tens in input, the function ...
   fullprod uses the corresponding
% full tensor product to evaluate the product of the ...
   payoff*probability of
% the payoff, computed by the normal pdf


% INPUT:
%          S_0                  1xdim vector, initial ...
   prices of underlyings
%          vett_tens            every component of ...
   the vector is the log2
%                               of the # of points in ...
   the corresponding dimension
```

```
%           K                       strike price
%           r                       risk-free rate
%           SIGMA                   SIGMA matrix of the ...
   multivariate normal
%                                   function
%           T                       expiration date
%           mu                      row vector
%
%           C                       1xdim vector, c(i) ...
   exprimes the quantity of the ith
%                                   underlying in my wallet
%
% OUTPUT:  addint:                  full tensorial ...
   product corresponding to
%                                   vett_tens
%
%


dim=size(S_0);
dim=dim(1,2)-1;
n = vett_tens;
N = 2.^n;

sigma = sqrt(diag(SIGMA))';
musg=(mu(1,2:end))';
dx = 10 * sigma(1,2:end)./N;

x_0 =  0.5.*N.*dx+musg';
pdf_=@(z) (1/(sqrt((2*pi)^dim*det(SGSIGMA)))*...
exp(-0.5*(z-musg')*ISGSIGMA*(z'-musg)));

for l=1:dim
number{l} = [(1:N(l))];  %ho tolto la prima dimensione!
space{l} = [(x_0(l):-dx(l):x_0(l)-(N(l)-1)*dx(l))];
% in space(i) there are the points, in the real world, of
%the i-th+1 dimension
end
```

```
positions = combvec(number{:});
spacecoord = combvec(space{:}); % vector of nodes, ...
    corresponding to positions


for q = 1:prod(N)
coord=num2cell(positions(:,q));  % take the ...
    coordinates from positions




M(coord{:})=pdf_((spacecoord(:,q))')*...
(onetrick(S_0,(C(1,2:end)*(S_0(1,2:end)'...
.*exp(spacecoord(:,q)))),C,K,SIGMA,r,T,mu,spacecoord(:,q)))...
*prod(dx)*coeff;
end
addint=real(sum(M(:)));


function ...
    [Onetrickprice]=onetrick(S_0,combund,C,K,SIGMA,r,T,mu_,spacecoord)

%conditioned mu, fixing the n-1 underlyings
mu=mu_(1)+SIGMA(1,2:end)*SIGMA(2:end,2:end)^(-1)*...
(spacecoord-mu_(1,2:end)');
%conditioned sigma^2
sigma=SIGMA(1,1)-SIGMA(1,2:end)*SIGMA(2:end,2:end)^(-1)*SIGMA(2:end,1);
s=sqrt(sigma);

STRIKE=K-combund;

if (STRIKE>0)

Onetrickprice=exp(-r*T)*((S_0(1,1)...
.*C(1,1)*exp(0.5*(s^2+2*mu))*...
(1-normcdf(log(STRIKE/(C(1,1)*S_0(1,1))),s^2+mu,s)))...
+STRIKE*(normcdf(log(STRIKE/(C(1,1)*S_0(1,1))),mu,s)-1));
else
Onetrickprice=exp(-r*T)*(S_0(1,1)*C(1,1)...
*exp(0.5*(s^2+2*mu))-STRIKE);
end
```

Bibliography

[1] Timothy J Barth and Herman Deconinck. *Error estimation and adaptive discretization methods in computational fluid dynamics*, volume 25. Springer Science & Business Media, 2013.

[2] Fischer Black and Myron Scholes. The pricing of options and corporate liabilities. *The journal of political economy*, pages 637–654, 1973.

[3] Phelim P Boyle. Options: A monte carlo approach. *Journal of financial economics*, 4(3):323–338, 1977.

[4] Hans-Joachim Bungartz and Michael Griebel. Sparse grids. *Acta numerica*, 13:147–269, 2004.

[5] Fabián Crocce, Juho Häppölä, Jonas Kiessling, and Raúl Tempone. Error analysis in fourier methods for option pricing. *to appear in Journal of Computational Finance. arXiv preprint arXiv:1503.00019*, 2015.

[6] Griselda Deelstra and Alexandre Petkovic. How they can jump together: Multivariate lévy processes and option pricing. *Belgian Actuarial Bulletin*, 9(1):29–42, 2010.

[7] Freddy Delbaen and Walter Schachermayer. A general version of the fundamental theorem of asset pricing. *Mathematische annalen*, 300(1):463–520, 1994.

[8] Thomas Gerstner. Sparse grid quadrature methods for computational finance. *Habilitation, University of Bonn*, 2007.

[9] Paul Glasserman. *Monte Carlo methods in financial engineering*, volume 53. Springer Science & Business Media, 2003.

[10] Norbert Hilber, Oleg Reichmann, Christoph Schwab, and Christoph Winter. *Multidimensional Feller Processes.* Springer, 2013.

[11] John C Hull. *Options, futures, and other derivatives.* Pearson Education India, 2006.

[12] Vesa Kaarnioja et al. Smolyak quadrature. 2013.

[13] Elisa Luciano and Wim Schoutens. A multivariate jump-driven financial asset model. *Quantitative finance*, 6(5):385–402, 2006.

[14] Dilip B Madan, Peter P Carr, and Eric C Chang. The variance gamma process and option pricing. *European finance review*, 2(1):79–105, 1998.

[15] Dilip B Madan and Eugene Seneta. The variance gamma (vg) model for share market returns. *Journal of business*, pages 511–524, 1990.

[16] E Novak and K Ritter. Simple cubature formulas for d-dimensional integrals with high polynomial exactness and small error. *Report, Institut f ur Mathematik, Universit Erlangen-Nurnberg*, 1997.

[17] Erich Novak and Klaus Ritter. The curse of dimension and a universal method for numerical integration. In *Multivariate approximation and splines*, pages 177–187. Springer, 1997.

[18] Sergey A Smolyak. Quadrature and interpolation formulas for tensor products of certain classes of functions. In *Dokl. Akad. Nauk SSSR*, volume 4, page 123, 1963.

[19] Joseph Frederick Traub and H Wozniakowski. Information-based complexity: new questions for mathematicians. *The Mathematical Intelligencer*, 13(2):34–43, 1991.

[20] Silvan Villiger and Christoph Schwab. *Basket option pricing on sparse grids using fast Fourier transforms.* PhD thesis, Citeseer, 2007.

[21] Grzegorz W Wasilkowski and Henryk Wozniakowski. Explicit cost bounds of algorithms for multivariate tensor product problems. *Journal of Complexity*, 11(1):1–56, 1995.