Die approbierte Originalversion dieser Diplom-/Masterarbeit ist in der Hauptbibliothek der Technischen Universität Wien aufgestellt und zugänglich.
http://www.ub.tuwien.ac.at

The approved original version of this diploma or master thesis is available at the main library of the Vienna University of Technology.
http://www.ub.tuwien.ac.at/eng

TECHNISCHE
UNIVERSITÄT
WIEN

DIPLOMARBEIT

IMPLEMENTATION OF A LÉVY DRIVEN ELECTRICITY FORWARD MODEL

unter Anleitug von

Ao.Univ.Prof. Dipl.-Ing. Dr.techn. Friedrich Hubalek
Institut für Stochastik und Wirtschaftsmathematik
Technische Universität Wien

Prof. Dr. Martin Larsson
Department of Mathematics
ETH Zürich

durch

Srećko Mihaljević
Schelleingasse 36
1040 Wien

Wien, September 2016

Mihaljević

Acknowledgements

Firstly, I would like to thank Professor Martin Larsson for accepting my proposal and supervising my master thesis during my exchange semester at ETH Zürich, as well as for putting me on the master project in collaboration with Axpo Trading AG. All of this meant a great deal to me, and I consider myself very lucky for getting this opportunity. His advice was always helpful and to the point, making the work on the thesis a very pleasant experience. I am also thankful to Professor Friedrich Hubalek, whose many lectures I've attended, for his suggestions and corrections that gave a final touch to this thesis, as well as for coordinating things on the side of TU Wien.

Furthermore I am very grateful to Vlatka Komarić and Markus Regez of Axpo Trading AG for entrusting me with the master project within their company, for their guidance and cooperation. This thesis was carried out in collaboration with Axpo Trading AG and the all of the market data was provided by them. I would especially like to thank Vlatka for organizing the whole thing, as well as Markus for his support and readiness to help at any time, even on his days off, and for taking the time to prepare the data and explain the technicalities of the electricity market that he knows so well. I was delighted to get this opportunity and am very thankful for it.

I would also like to thank professors Uwe Schmock and Thorsten Rheinländer for nominating me for an exchange semester at ETH Zürich, which meant a great deal to me and without which none of this would have been possible. I would also like to express my gratitude to all lecturers and staff of TU Wien, especially FAM, for the great five years I have spent there as a student. Everyone has always been very helpful and made me feel more than welcome.

Finally, I would like to thank my family for granting me the opportunity to study abroad, as well as for supporting me throughout. I have always tried to make the most out of this opportunity. Moreover, I would also like to thank my girlfriend for her love and support during these years. Last but not least, I would like to thank my friends for all of their support.

I am sincerely grateful for all of it.

Contents

1 Basic mathematical tools 5

1.1 Random variables and probability spaces . . . 5
1.2 Characteristic and generating functions . . . 7
1.3 Convergence of random variables . . . 9
  1.3.1 Pointwise convergence . . . 9
  1.3.2 Almost-sure convergence . . . 9
  1.3.3 Convergence in probability . . . 9
  1.3.4 Convergence in distribution . . . 9
1.4 Stochastic processes, filtrations, random times, martingales . . . 10
  1.4.1 Stochastic processes . . . 10
  1.4.2 Filtrations and history . . . 10
  1.4.3 Random times . . . 11
1.5 Martingales and local martingales . . . 12
1.6 Brownian Motion . . . 12
1.7 Poisson process . . . 12
  1.7.1 Compensated Poisson process . . . 13
  1.7.2 Poisson random measure . . . 14
  1.7.3 Compensated Poisson random measure . . . 15
1.8 Lévy processes . . . 15
  1.8.1 Characteristic function . . . 16
  1.8.2 Compound Poisson process . . . 16
  1.8.3 Jump and Lévy measure . . . 17
  1.8.4 Lévy-Ito decomposition . . . 17
  1.8.5 Lévy-Khinchin representation . . . 19
  1.8.6 Exponential moments . . . 19
  1.8.7 Lévy processes and martingales . . . 20
1.9 Additive processes . . . 20
1.10 Stochastic calculus . . . 21

2 Electricity markets 25

2.1 Electricity trading . . . 27
  2.1.1 Day-ahead market . . . 29
  2.1.2 Futures and forward market . . . 30
  2.1.3 Balancing and reserve market . . . 31
  2.1.4 Market coupling . . . 32
2.2 Electricity Exchanges . . . 32

3 An overview of price models 36

3.1 Black 76 model . . . 37
3.2 Relation of the spot and forward price . . . 37
  3.2.1 Cost of carry formula . . . 38
  3.2.2 Convergence relation . . . 38
  3.2.3 Spot-forward risk neutral relation . . . 38
  3.2.4 Risk premium . . . 39

3.3 HJM-style models 39
3.3.1 Single-factor model 40
3.3.2 Modeling multiple granularities 41
3.3.3 Two-factor model 41
3.3.4 Joint modelling 42
3.3.5 Conclusion 42
3.4 Spot models 42
3.4.1 Mean-reverting diffusion model 42
3.4.2 Mean-reverting jump-diffusion model 45
3.4.3 Multi-factor models 48
3.5 Conclusion 49
3.6 Structural models 50
3.6.1 Barlow's model as a basic structural model 50
3.6.2 Cartea and Villaplana's two factor structural model 51
3.6.3 Conclusion 53
3.7 Limitations of Brownian motion driven financial modelling 54

4 COS pricing method 56
4.1 Inverse Fourier Integral via Cosine Expansion 56
4.2 Pricing European options 57
4.3 Coefficients V_k for plain vanilla options 59
4.4 Simplified formula 59
4.5 Error analysis 60
4.6 Choice of the truncation range 63

5 Parallel computing with CUDA 64
5.1 Serial and parallel computing 64
5.2 The move towards general purpose GPU computing 65
5.3 CPU and GPU comparison 67
5.4 Parallel programming with CUDA 68
5.4.1 A scalable programming model 68
5.4.2 Kernels, thread blocks and memory hierarchy 69
5.5 Parallel reduction sum algorithm 72
5.6 Speeding up pricing and calibration 74

6 Our model 75
6.1 Motivation 76
6.2 The Lévy driven model 77
6.3 Option pricing 79
6.4 Calibration 80
6.4.1 Differential evolution optimization 80
6.5 Results 81
6.6 Conclusion 85
6.7 Further steps 85
6.7.1 Two-market model proposal 86

A Implementation and source code 88

Introduction

The main aim of this thesis is to provide an overview of current developments in the world of electricity modelling and to develop a Lévy driven electricity forward model, with a strong emphasis on tractability and performance. To achieve this, the possibilities of parallel computing using graphic cards are explored. The outline of this thesis is as follows.

In chapter 1, we briefly recall some of the fundamental mathematical tools used in stochastic modelling of financial markets. General properties of electricity as a commodity and the different products traded on electricity exchanges are discussed in chapter 2. An overview of the electricity modelling approaches with some examples is given in chapter 3. In the next two chapters we study the building blocks of our pricing method to be. In chapter 4, the COS pricing method originally developed in [1] is presented. Chapter 5 provides the reader with a crash course in the CUDA technology. Finally, in chapter 6, our Lévy driven model is presented together with calibration results. Some further steps are also suggested.

In appendix A, the source code of the core functionality of our implementation is available, with a quick description of how to use it.

During one's studies of mathematical finance, one is commonly exposed to stock and interest rate modelling. Energy markets, on the other hand, are usually covered only by elective courses or or seminars, which is somewhat unfair. The modelling of energy markets is often just as, if not even more, exciting, due to the physical nature of the traded products. Electricity, as a hardly storable commodity, is especially challenging.

Chapter 1

Basic mathematical tools

In this introductory chapter, we outline the mathematical tools used in modelling of financial markets and throughout this thesis, with an emphasis on interpretation. Countless discussions of stochastic models start with the magical sentence "Let $(\Omega, \mathcal{F}, \mathbb{P})$ be a probability space", so this is where we begin our review. First we recall definitions of probability spaces, random variables and probabilistic convergence concepts. We then move on to recalling basic facts about stochastic processes and how information flow is modelled through filtrations. The concept of a martingale and its importance is shortly discussed afterwards. Next we introduce Lévy processes and their slight generalization in the form of additive processes. Finally we discuss some results from stochastic calculus. This section is heavily based on the results from [2, 3, 4].

1.1 Random variables and probability spaces

At the foundation of modern probabilistic theory and at the core of every stochastic model is the probability space. Here we recall its definition, some of its properties, and its interpretation in financial modelling.

Definition 1.1. A probability space $(\Omega, \mathcal{F}, \mathbb{P})$ consists of three parts:

1. A set of all possible scenarios (outcomes) Ω.
2. A set of events, a σ-algebra $\mathcal{F}$ on Ω. Each event is a set containing zero or more scenarios.
3. A probability measure $\mathbb{P}$, that assigns a probability between 0 and 1 to each event:

$$\begin{aligned} \mathbb{P} : \mathcal{F} &\to [0,1] \\ A &\mapsto \mathbb{P}(A). \end{aligned}$$

An event $A \in \mathcal{F}$ with $\mathbb{P}(A) = 1$ is said to occur $\mathbb{P}$-almost surely, whereas an event with $\mathbb{P}(A) = 0$ is said to be impossible under $\mathbb{P}$. A subset of an impossible event is called a $\mathbb{P}$-null set. A property is said to hold $\mathbb{P}$-almost surely ($\mathbb{P}$-a.s.) if the set of $\omega \in \Omega$ for which it does not is a $\mathbb{P}$-null set. It is always possible to complete $\mathcal{F}$ to include all null sets, meaning we consider and assign probability zero to all subsets of impossible events in our probability space, which is intuitively reasonable. In the following, we will always consider the complete version of all σ-algebras, unless stated otherwise.

The notion of comparable probability measures is introduced through *equivalent* probability measures.

Definition 1.2 (Absolutely continuous and equivalent probability measures)**.** Let $\mathbb{P}, \mathbb{Q}$ be two probability measures on $(\Omega, \mathcal{F})$. The measure $\mathbb{Q}$ is absolutely continuous w.r.t. $\mathbb{P}$, denoted by $Q \ll P$, if all $\mathbb{P}$-null sets are also $\mathbb{Q}$-null sets.

Probability measures $\mathbb{P}$ and $\mathbb{Q}$ are equivalent on $(\Omega, \mathcal{F})$ if $Q \ll P$ and $P \ll Q$, i.e., they define the same impossible events:

$$\mathbb{P} \sim \mathbb{Q} \iff [\forall A \in \mathcal{F}, \mathbb{P}(A) = 0 \iff \mathbb{Q}(A) = 0] .$$

In mathematical finance, one distinguishes between the *physical*, sometimes also called statistical, and the *risk-neutral* probability measure. The two measures are comparable, but not the same. The physical measure is supposed to reflect real-world dynamics of financial assets. On the other hand, the risk-neutral measure is the measure of choice for the valuation of derivatives. Traded instruments are supposed to be (local) $\mathbb{Q}$-martingales, hence their behaviour under $\mathbb{Q}$ typically differs from their behaviour under $\mathbb{P}$.

Definition 1.3 (Random variable)**.** Let $(\Omega, \mathcal{F}, \mathbb{P})$ be a probability space. A random variable X taking values in $(E, \mathcal{E})$ is a measurable function $X : \Omega \to E$. An element $\omega \in \Omega$ is called a scenario of randomness and $X(\omega)$ is called the realization of X in scenario ω.

Two random variables X and Y are said to be $\mathbb{P}$-almost surely equal if:

$$X = Y \ \mathbb{P}\text{-a.s.} \iff \mathbb{P}\left(\omega \in \Omega, X(\omega) = Y(\omega)\right) = 1.$$

The distribution (or law) of X is the probability measure on $(E, \mathcal{E})$ defined by:

$$\mu_X(A) = \mathbb{P}(X \in A) \text{ for } A \subset E.$$

For $\mu_X = \mu_Y$ we write $X \overset{d}{=} Y$ (equality in distribution).

For an $\mathbb{R}$-valued random variable $X : \Omega \to E \subseteq \mathbb{R}$, we define the expectation:

$$\mathbb{E}^{\mathbb{P}}[X] = \int_{\Omega} X(\omega) d\mathbb{P}(\omega) = \int_{E} y d\mu_X(y).$$

Let us now say a few words about random variables, underlying probability spaces and their intuitive interpretation. Although there is a probability space $(\Omega, \mathcal{F}, \mathbb{P})$ behind every stochastic financial model, one easily loses track of what Ω actually represents and who lives inside. Indeed, in many applications of probability and statistics, a random variable X in synonymous with its distribution μ_X. Calculations such as sums, convolutions, Fourier transformations can all be carried out relying only on the distribution of the random variable without regard of the underlying probability space.

From a modelling perspective, the key observation to make is the distinction between the source of randomness and the random variable itself. For example, let us look at the case of modelling the price of some stock S_1 at some future date T. We may assume that the stock price is driven by many factors such as external news, market supply and demand, economic indicators etc., summed up in some abstract variable ω (scenario of a future evolution of the market), which may not even have a numerical representation! If there is only one stock with price S_1, we can label the scenario ω with $S_1(\omega)$, resulting in a one to one relationship and making the current observation of the stock price synonymous with the scenario ω. In this case, the full probabilistic description of a random variable S_1 can indeed be reduced to the distribution of S_1. However, if there are also other stocks $\{S_2, S_3, \dots\}$ present in the model, it is natural distinguish the scenario ω from random variables $\{S_1(\omega), S_2(\omega), \dots\}$ whose values are observed in this scenario. Since the distribution of S_1 is in fact only the marginal distribution, and dependence across different stocks may exist, knowing solely the distribution of S_1 does not provide a full probabilistic description of S_1. This information is stored in $\mathbb{P}$, that models the probability distribution of the scenarios $\omega \in \Omega$. However, these scenarios may be abstract variables which we never have to deal with directly. We simply model the dependence between the observable factors which translates into joint distributions of (observable) random variables.

To sum it up, the probabilistic description of a random variable X can be reduced to the knowledge of its (marginal) distribution μ_X only if the random variable X is the only source of randomness. As soon as we are concerned with a second random variable which is not a deterministic function of X, the underlying probability measure $\mathbb{P}$ contains more information on X than just its (marginal) distribution. It contains *all* of the information about the dependence of X w.r.t. *all* other random variables in the model. Specifying $\mathbb{P}$ is equal to specifying the joint distribution of all random variables constructed on Ω. $\mathbb{P}$ contains full information about all sources of randomness.

1.2 Characteristic and generating functions

When dealing with random variables, the most advanced analytical tool at our disposal is, without a doubt, the characteristic function. In this section we recall its definition and some of its properties, as well as related concepts of the moment and the cumulant generating function.

Definition 1.4. The characteristic function of an $\mathbb{R}^d$-valued random variable X is defined by:

$$\Phi_X(w) := \mathbb{E}\left[e^{iw^T X}\right] = \int_{\mathbb{R}^d} e^{iz^T x} d\mu_X(x) \text{ for } w \in \mathbb{R}^d.$$

The characteristic function of a random variable completely characterizes its distribution. It is always continuous with $\Phi_X(0) = 1$. Any additional smoothness of Φ_X depends on the existence of moments of the random variable X.

Definition 1.5 (Moments of a random variable)**.** The n-th moment of a random variable is defined by

$$m_n(X) = \mathbb{E}\left[X^n\right],$$

the n-th absolute moment by

$$m_n(|X|) = \mathbb{E}\left[|X^n|\right],$$

and the n-th centred moment by

$$\mu_n(X) = \mathbb{E}\left[(X - \mathbb{E}[X])^n\right].$$

The existence of moments of a random variable depends on how fast the distribution μ_X decays at infinity.

Result 1.6 (Characteristic function and moments [2])**.** There is a relationship between the characteristic function and the moments of a random variable.

1. If $\mathbb{E}\left[|X|^n\right] < \infty$ then Φ_X has n continuous derivatives at $z = 0$ and:

$$\forall k \in \{1 \dots n\} \quad m_k = \mathbb{E}\left[X^k\right] = \frac{1}{i^k} \frac{\partial^k \Phi_X}{\partial z^k}(0).$$

2. If Φ_X has $2n$ continuous derivatives at $z = 0$ then $\mathbb{E}\left[|X|^{2n}\right] < \infty$ and:

$$\forall k \in \{1 \dots 2n\} \quad m_k = \mathbb{E}\left[X^k\right] = \frac{1}{i^k} \frac{\partial^k \Phi_X}{\partial z^k}(0).$$

3. X possesses finite moments of all orders if and only if $z \mapsto \Phi_X(z)$ is C^∞ at $z = 0$. Then the moments of X are related to the derivatives of Φ_X by:

$$m_n = \mathbb{E}\left[X_n\right] = \frac{1}{i^n} \frac{\partial^n \Phi_X}{\partial z^n}(0).$$

Result 1.7 ([2])**.** Let $\{X_i, i = 1 \dots n\}$ be independent random variables. The characteristic function of $S_n = \sum\limits_{i=1}^{n} X_i$ is given by:

$$\Phi_{S_n}(z) = \prod_{i=1}^{n} \Phi_{X_i}(z). \tag{1.1}$$

Definition 1.8 (Moment generating function)**.** The moment generating function a $\mathbb{R}^d$ valued random variable X is defined by:

$$M_X(u) := \mathbb{E}\left[e^{u^T X}\right], \forall u \in \mathbb{R}^d. \tag{1.2}$$

Unlike the characteristic function, which always exists (as the Fourier transform of a probability measure, therefore integrable), the integral in the definition of the moment generating function may not always converge. In case M_X is well defined, it can be formally related to Φ_X through $M_X(u) = \Phi_X(-iu)$.

If M_X of an $\mathbb{R}$-valued random variable X exists on a neighbourhood of zero $[-\epsilon, \epsilon]$, then all (polynomial) moments of X exist and can be calculated using:

$$m_n = \frac{\partial^n M_X}{\partial u^n}(0).$$

Definition 1.9 (Cumulant generating function)**.** Let X be a random variable and Φ_X its characteristic function. Since $\Phi_X(0) = 1$ and Φ_X is continuous at $z = 0$ it follows that $\Phi_X \neq 0$ in a neighbourhood of $z = 0$. One can then define a continuous version of the logarithm of Φ_X. In a neighbourhood of $z = 0$, a unique continuous function Ψ_X, called *cumulant generating function* exists, such that:

$$\Phi_X(w) = e^{\Psi_X(w)} \text{ with } \Psi_X(0) = 0.$$

If $\Phi_X(w) \neq 0$ for all w, the cumulant generating function can be extended to all $w \in \mathbb{R}^d$.

The *cumulants* of X, sometimes also called semi-invariants, are defined by:

$$c_n(X) = \frac{1}{i^n} \frac{\partial^n \Psi_X}{\partial u^n}(0).$$

Expanding the exponential function at $z = 0$ and using (1.2), we derive the relationship between moments and cumulants of X:

$$\begin{aligned}
&c_1(X) = m_1(X) = \mathbb{E}[X] \\
&c_2(X) = \mu_2(X) = m_2(X) - m_1(X) = Var(X) \\
&c_3(X) = \mu_3(X) = m_3(X) - 3m_2(X)m_1(X) + 2m_1(X)^3 \\
&c_4(X) = \mu_4(X) - 3\mu_2(X)
\end{aligned}$$

Furthermore, one can now define *skewness* and *kurtosis*:

$$s(X) = \frac{c_3(X)}{c_2(X)^{\frac{3}{2}}}, \qquad \kappa(X) = \frac{c_4(X)}{c_2(X)^2}.$$

If X follows a normal distribution, Ψ_X is a second degree polynomial and therefore $\forall n \geq 3, c_n(X) = 0$. Therefore, both $s(X) = 0$ and $\kappa(X) = 0$ for a normally distributed random variable. In this context, $s(X)$, $\kappa(X)$ and higher cumulants can be seen as measures of deviation from normality. A distribution of a random variable X is called *skewed*, if $s(X) \neq 0$. Similarly, a distribution is said to be is said to be *leptokurtic*, more commonly heavy-tailed, if $\kappa(X) > 0$.

Finally, for $(X_i)_{i=1\dots n}$ independent random variables one can easily deduce from (1.1) that:

$$\Psi_{X_1+\dots+X_n}(w) = \sum_{i=1}^{n} \Psi_{X_i}(w). \tag{1.3}$$

1.3 Convergence of random variables

We now recall the different definitions of convergence in the world of random variables, starting from the "strictest".

1.3.1 Pointwise convergence

The most straightforward concept of convergence is the pointwise convergence.

Definition 1.10 (Pointwise convergence)**.** A sequence of random variables $(X_n)_{n\geq 1}$ taking values in some normed vector space E is said to converge pointwise towards a random variable X, if for all $\omega \in \Omega$, the sequence $(X_n(\omega))_{n\geq 1}$ converges to $X(\omega)$ in E.

However, this definition turns out to be too strong in many cases, since we are asking for convergence in each and every scenario, whereas some of them may be negligible from a probabilistic point of view.

1.3.2 Almost-sure convergence

The concept of almost-sure convergence takes above fact into account and requires pointwise convergence only for realizations with positive probability.

Definition 1.11 (Almost-sure convergence)**.** Let $(X_n)_{n\geq 1}$ and X be random variables on the same probability space $(\Omega, \mathcal{F}, \mathbb{P})$. Then the sequence $(X_n)_{n\geq 1}$ is said to converge $\mathbb{P}$-almost surely to X if

$$\mathbb{P}\left(\lim_{n\to\infty} X_n = X\right) = 1.$$

Almost sure convergence does not imply convergence of moments. If $X_n \to X$ $\mathbb{P}$-almost surely, $\mathbb{E}\left[X_n^k\right]$ may be defined for all $n \geq 1$ but have not a limit as $n \to \infty$.

1.3.3 Convergence in probability

While the almost-sure convergence of $(X_n)_{n\geq 1}$ deals with the behaviour of typical samples $(X_n(\omega))_{n\geq 1}$, convergence in probability *only* puts a condition on the probability of events as $n \to \infty$.

Definition 1.12 (Convergence in probability)**.** A sequence (X_n) of random variables on $(\Omega, \mathcal{F}, \mathbb{P})$ is said to converge in probability to a random variable X if:

$$\forall \epsilon > 0 \lim_{n\to\infty} \mathbb{P}\left(|X_n - X| > \epsilon\right) = 0.$$

We denote convergence in probability by $X_n \xrightarrow[n\to\infty]{\mathbb{P}} X$.

1.3.4 Convergence in distribution

Note that for above definitions of convergence, it is required for the random variables to be defined on the same probability space. Convergence in distribution does not require this, in fact, it only has to do with the distributions of the random variables and not the values of the random variables themselves.

Definition 1.13 (Convergence in distribution)**.** Let (X_n) be a sequence of random variables with corresponding cumulative distribution functions F_{X_n}. Then, (X_n) converges in distribution to a random variable X if

$$\lim_{n\to\infty} F_{X_n}(x) = F_X(x),$$

at all points where $F_X(x)$ is continuous. Hereby F_X denotes the cumulative distribution function of X.

Convergence in distribution is sometimes also called weak convergence of the measures on E. We write $X_n \xrightarrow{d} X$ or $\mu_n \to \mu$.

Result 1.14 (Alternative characterization of convergence in distribution)**.** A sequence $(X_n)_{n\geq 1}$ of random variables with values in E converges in distribution to a random variable X if and only if, for any bounded continuous function $f : E \to \mathbb{R}$

$$\mathbb{E}\left[f(X_n)\right] \xrightarrow{n\to\infty} \mathbb{E}\left[f(X)\right],$$

which is equivalent to:

$$\forall f \in C^b(E,\mathbb{R}), \qquad \lim_{n\to\infty} \int\limits_E f(x) d\mu_n = \int\limits_E f(x) d\mu(x).$$

Result 1.15 (Characteristic function and convergence in distribution [2])**.** $(X_n)_{n\geq 1}$ converges in distribution to X, if and only if

$$\forall w \in \mathbb{R}^d \quad \Phi_{X_n}(w) \to \Phi_X(w).$$

On the other hand, pointwise convergence of (Φ_{X_n}), since the limit is not necessarily a characteristic function, does not imply existence of a weak limit of (X_n), hence convergence in distribution. In addition, convergence in distribution does not necessarily mean convergence of moments. The moments of X_n, X may not even exist to begin with.

If (X_n) and X are defined on the same probability space, convergence in distribution is the weakest form of convergence among the above. Almost sure convergence implies convergence in probability, and convergence in probability implies convergence in distribution.

1.4 Stochastic processes, filtrations, random times, martingales

In this section we recall some basic definitions concerning stochastic processes. We also review how information is modelled using filtrations.

1.4.1 Stochastic processes

Stochastic processes are one of the central objects in mathematical finance. Evolution of stock, energy prices etc., are represented by a stochastic process whose dynamic is then modelled.

Definition 1.16 (Stochastic process)**.** A stochastic process is a family $(X_t)_{t\in[0,T]}$ of random variables indexed by time. For each scenario ω, the trajectory $X(\omega)$ is called the sample path of the process.

Definition 1.17 (Càdlàg)**.** A stochastic process is called càdlàg, if for each $\omega \in \Omega$, $X_t(\omega)$ is a right-continuous function with left limits for each t.

1.4.2 Filtrations and history

As time goes on, more information is progressively revealed to the observer. Quantities that are viewed at random at some time t_1 may change status and become known with the information available at some later time $t_2 > t_1$. Therefore, it is natural to add some time-dependent ingredient to the structure of our probability space $(\Omega, \mathcal{F}, \mathbb{P})$. This is usually done by introducing the concept of a *filtration.*

Definition 1.18 (Filtration)**.** A filtration on $(\Omega, \mathcal{F}, \mathbb{P})$ is an increasing family of σ-algebras $(\mathcal{F}_t)_{t\in[0,T]}$ such that for all s, t with $0 \leq s \leq t \leq T$ holds $\mathcal{F}_s \subseteq \mathcal{F}_t \subseteq \mathcal{F}_T = \mathcal{F}$. A probability space equipped with a filtration is called a filtered probability space.

We interpret $\mathcal{F}_t$ as information known at time t, which increases with time. Although intuitively it might seem reasonable to adjust the probability measure to account for the change in probability of occurrence of a random event as more information becomes available, we keep the probability measure $\mathbb{P}$ fixed. Instead, we will model the impact of information flow by conditioning on $\mathcal{F}_t$.

After introducing filtrations, we are now able to distinguish events and quantities which are known from those which are still considered random given the information at some time t. For an event $A \in \mathcal{F}_t$ with the information $\mathcal{F}_t$ at time t the observer can decide whether it has happened or not. An $\mathcal{F}_t$-measurable random variable is a random variable whose value will be revealed at time t. Of special interest to us are stochastic processes whose value at time t is revealed by the information $\mathcal{F}_t$.

Definition 1.19 (Adapted stochastic process)**.** A stochastic process $(X_t)_{t\in[0,T]}$ is said to be adapted to the filtration $(\mathcal{F}_t)_{t\in[0,T]}$ ($\mathcal{F}_t$-adapted) if, for each $t \in [0, T]$, X_t is $\mathcal{F}_t$-measurable.

Whereas $\mathcal{F}_t$ represents the information and history up to time t of the whole probability space, the history of a single stochastic process is represented by its natural filtration.

Definition 1.20 (Natural filtration of a stochastic process)**.** The natural filtration of a stochastic process $(X_t)_{t\in[0,T]}$ is given by $(F_t^X)_{t\in[0,T]}$ where F_t^X is the σ-algebra generated by the past values of the process, completed by the null sets:

$$\mathcal{F}_t^X = \sigma\left(X_s, s \in [0, t]\right) \bigvee \mathcal{N}$$

One may interpret $\mathcal{F}_t^X$ as containing all information one can get from having observed the path of X on $[0, t]$.

1.4.3 Random times

A random time as a positive random variable $\tau \geq 0$ representing a time at which some (random) event takes place. If at time t, given the available information, contained in $(\mathcal{F}_t)$, we can determine whether the event has happened $(\tau \leq t)$ or not $(\tau > t)$, then the random time τ is called a stopping time.

Definition 1.21 (Stopping time)**.** A random variable τ, taking values in $[0, \infty]$ is an $(\mathcal{F}_t)$-stopping time, if it verifies

$$\forall t \geq 0, \{\tau \leq t\} \in \mathcal{F}_t.$$

Given two stopping times τ_1 and τ_2, $\inf\{\tau_1, \tau_2\}$ is also a stopping time. For a stochastic process (X_t) and a stopping time τ, the stopped process $X_{\tau\wedge t}$ is defined by:

$$X_{\tau\wedge t} = X_t \text{ for } t < \tau \quad X_{\tau\wedge t} = X_\tau \text{ for } t \geq \tau.$$

A common example of a stopping time is the first hitting time. For an $(\mathcal{F}_t)$-adapted càdlàg stochastic process X, the hitting time of an open set A is defined by the first time when X reaches A:

$$\tau_A = \inf\{t \geq 0, X_t \in A\}.$$

An example of a random time that is typically not a stopping time is:

$$\tau_{max} = \inf\{t \in [0, T], X_t = \sup_{s\in[0,T]} X_s\}.$$

Obviously one does not know the maximum of some processes on the whole interval $[0, T]$ before observing the whole path.

1.5 Martingales and local martingales

Another one of the basic definitions is that of a martingale.

Definition 1.22 (Martingale)**.** Let $(\Omega, \mathcal{F}, (\mathcal{F}_t), \mathbb{P})$ be a filtered probability space. A stochastic process $(X_t)_{t\in[0,T]}$ is a martingale if:

1. X is $\mathcal{F}_t$-adapted.
2. $\mathbb{E}\left[|X_t|\right] < \infty$ for any $t \in [0, T]$.
3. $\forall s > t \quad \mathbb{E}\left[X_s \mid \mathcal{F}_t\right] = X_t$.

Simply put, the present value of a martingale is also the best prediction of its future value. Martingales play a special role in mathematical finance, modelling the price process under the risk-neutral measure.

A generalization of the concept of a martingale through localization with stopping times gives rise to local martingales.

Definition 1.23 (Local martingale)**.** A stochastic process $(X_t)_{t\in[0,T]}$ is called a *local martingale* if there exists a sequence of stopping times (τ_n) with $\tau_n \to \infty$ such that $(X_{t\wedge\tau_n})_{t\in[0,T]}$ is a true martingale.

Definition 1.24 (Properties that hold locally)**.** We say that a property holds *locally*, if there exists a sequence of stopping times (τ_n) with $\tau_n \to \infty$ such that the property holds for $(X_{t\wedge\tau_n})$

1.6 Brownian Motion

The most studied stochastic processes, with countless applications in mathematical finance and beyond and with unique continuity properties is Brownian motion, also called Wiener process.

Definition 1.25 (Brownian motion, Wiener process)**.** An $\mathbb{R}$-valued stochastic process B_t on $(\Omega, \mathcal{F}, \mathbb{P})$ is called a Brownian motion if:

1. $B_0 = 0$.
2. B_t is $\mathbb{P}$-almost-surely continuous.
3. B_t has independent increments.
4. $B_t - B_s \overset{\mathbb{P}}{\sim} \mathcal{N}(0, t - s)$ for $0 \le s \le t$.

A d-dimensional Brownian motion is constructed by $\boldsymbol{B_t} := \left(B_t^1, \ldots, B_t^d\right)^T$ where (B_t^i) are independent copies of the $\mathbb{R}$-valued Brownian motion defined as above. Given a covariance matrix A with decomposition $A = \Sigma\Sigma^T$ one can we can define $\boldsymbol{W_t} := \Sigma\boldsymbol{B_t}$. Then $\boldsymbol{W_t}$ is a correlated d-dimensional Brownian motion.

For Brownian motion we will usually write either W_t or B_t.

1.7 Poisson process

The Poisson process is a fundamental example of a stochastic process with discontinuous paths. Together with the Brownian motion, it plays an important role as a building block of more complicated jump processes. We now quickly review some of the underlying theory.

Definition 1.26 (Poisson process)**.** Let $(\tau_i)_{i\geq 1}$ be a sequence of independent exponential random variables with parameter λ and $T_n := \sum\limits_{i=1}^{n} \tau_i$. A Poisson process with intensity λ is defined by:

$$N_t = \sum_{n\geq 1} 1_{t\geq T_n}.$$

The Poisson process is a counting process: it counts the number of random times (T_n) which occur in $[0, t]$, where $(T_n - T_{n-1})_{n\geq 1}$ is an i.i.d. sequence of exponential variables.

Result 1.27 (Properties of a Poisson process)**.** Let $(N_t)_{t\geq 0}$ be a Poisson process on $(\Omega, \mathcal{F}, \mathbb{P})$. Then the following assertions hold:

1. For any $t > 0$, N_t is $\mathbb{P}$-almost surely finite.
2. For any ω, the sample path $t \mapsto N_t(\omega)$ is a piecewise constant function which increases by jumps of size 1.
3. The sample paths $t \mapsto N_t(\omega), \forall \omega \in \Omega$ are right continuous with left limits (càdlàg).
4. For any $t > 0$, $N_{t-} = N_t$ with probability 1.
5. (N_t) is continuous in probability: $\forall t > 0, N_s \xrightarrow[s\to t]{\mathbb{P}} N_t$.
6. For any $t > 0$, N_t follows a Poisson distribution with parameter λt:
$$\forall n \in \mathbb{N} \quad \mathbb{P}(N_t = n) = e^{-\lambda t}\frac{(\lambda t)^n}{n!}.$$
7. The characteristic function of N_t is given by
$$\mathbb{E}\left[e^{iwN_t}\right] = \exp\left(\lambda t\left(e^{iw} - 1\right)\right), \forall w \in \mathbb{R}.$$
8. (N_t) has independent increments: for any $t_1 < \ldots t_n$, $N_{t_n} - N_{t_{n-1}}, \ldots, N_{t_2} - N_{t_1}, N_{t_1}$ are independent random variables.
9. The increments of N_t are homogeneous for any $t > s$, $N_t - N_s$ has the same distribution as N_{t-s}.
10. (N_t) has the Markov property:
$$\forall t > s : \mathbb{E}\left[f(N_t) \mid N_u,\ u \leq s\right] = \mathbb{E}\left[f(N_t) \mid N_s\right].$$

Result 1.28 (Sum of independent Poisson processes [2])**.** Let $(N_t^1)_{t\geq 0}$, $(N_t^2)_{t\geq 0}$ be independent Poisson processes with intensities λ_1, λ_2. Then $(N_t^1 + N_t^2)_{t\geq 0}$ is a Poisson process with intensity $\lambda_1 + \lambda_2$.

1.7.1 Compensated Poisson process

Definition 1.29 (Compensated Poisson process)**.** Let $(N_t)_{t\geq 0}$ be a Poisson process. The compensated Poisson process is defined as:

$$\tilde{N_t} = N_t - \lambda t,$$

where the deterministic part λt, called the *compensator*, makes the process a martingale. Unlike the Poisson process, it is neither a counting process nor $\mathbb{N}$-valued. Its characteristic function is given by:

$$\Phi_{\tilde{N}_t}(w) = \exp\left(\lambda t\left(e^{iw} - 1 - iw\right)\right).$$

1.7.2 Poisson random measure

The Poisson process is a counting process. Let $(N_t)_{t\geq 0}$ be a Poisson process and $\{T_1, T_2, \dots\}$ a sequence of jump times. Then, for $t > s$, $N_t - N_s$ is the number of jumps in the interval $(s, t]$:

$$N_t - N_s = \#\{i \geq 1, T_i \in (s,t]\}.$$

The jump times $\{T_1, T_2, \dots\}$ form a random configuration of points on $[0, \infty)$ and the Poisson process counts the number of such points in the interval $[0, t]$. This counting procedure can be used to define a *random jump measure* M on $[0, \infty)$ associated to the Poisson process $(N_t)_{t\geq 0}$. For any measurable set $A \in \mathbb{R}^+$ we set

$$M(\omega, A) := \#\{i \geq 1, T_i(\omega) \in A\}. \tag{1.4}$$

Then $M(\omega, .)$ is an $\mathbb{N}$-valued random measure and $\mathbb{P}(M(A) < \infty) = 1$ holds for any bounded set A. It is a random measure because it depends on the scenario ω. From $\mathbb{E}\left[M(A)\right] = \lambda\,|A|$, where $|A|$ is the Lebesgue measure, it follows that the average value of M is determined by the intensity λ of the corresponding Poisson process.

The Poisson process $(N_t)_{t\geq 0}$ may be expressed in terms of the random measure M:

$$N_t(\omega) = M\left(\omega, [0,t]\right) = \int\limits_{[0,t]} M(\omega, ds).$$

The random jump measure associated to the Poisson process inherits some similar properties:

1. $M\left([t_1, t_2]\right)$ is the number of jumps of the associated Poisson process in $[t_1, t_2]$. More precisely, $M\left([t_1, t_2]\right) \sim \mathit{Poisson}\left(\lambda(t_2 - t_1)\right)$.
2. For two disjoint intervals $[t_1^a, t_2^a]$ and $[t_1^b, t_2^b]$, $M([t_1^a, t_2^a])$ and $M([t_1^b, t_2^b])$ are independent random variables.
3. For any measurable set A with $0 < |A| < \infty$ it holds $M(A) \sim \mathit{Poisson}(\lambda|A|)$, where $|A|$ is the Lebesgue measure.

Analogously one can associate a random measure to the compensated Poisson process $\tilde{N}_t$ giving rise to the *compensated random measure.*

$$\tilde{M}(\omega, A) = M(\omega, A) - \int\limits_A \lambda dt = M(\omega, A) - \lambda|A|. \tag{1.5}$$

However, $\tilde{M}$ is a signed measure. It is neither integer valued nor necessarily positive.

The measure M defined in (1.4) is a random counting measure on $\mathbb{R}^+$, such that for any measurable $A \subset \mathbb{R}^+$ it holds $\mathbb{E}\left[M(A)\right] = \lambda|A|$. This construction can be generalized, replacing $\mathbb{R}^+$ by $E \subset \mathbb{R}^d$ and the Lebesgue measure by a Radon measure μ on E.

Definition 1.30 (Poisson random measure)**.** Let $(\Omega, \mathcal{F}, \mathbb{P})$ be a probability space, $E \subset \mathbb{R}^d$ and μ a (positive) Radon measure μ on $(E, \mathcal{E})$. A Poisson random measure on E with intensity measure μ is an integer valued random measure:

$$\begin{aligned} M : \Omega \times \mathcal{E} &\to \mathbb{N} \\ (\omega, A) &\mapsto M(\omega, A), \end{aligned}$$

such that

1. For almost all $\omega \in \Omega$, $M(\omega, .)$ is an integer-valued Radon measure on E. For any bounded measurable $A \subset E, M(A) < \infty$ is an integer valued random variable.
2. For each measurable set $A \subset E$, $M(., A) \sim \mathit{Poisson}(\mu(A))$.

3. For all disjoint measurable sets $A_1, \dots, A_n \in \mathcal{E}$, the variables $M(A_1), \dots, M(A_n)$ are independent.

Result 1.31 (Construction of Poisson random measures [2]). For any Radon measure μ on $E \in \mathbb{R}^d$, there exists a Poisson random measure M on E with intensity μ.

1.7.3 Compensated Poisson random measure

Definition 1.32 (Compensated Poisson random measure). Similar to the definition of the compensated Poisson process, we define the compensated Poisson random measure $\tilde{M}$:

$$\tilde{M}(A) := M(A) - \mu(A).$$

For disjoint compact sets $A_1, \dots, A_n \in \mathcal{E}$, the variables $\tilde{M}(A_1), \dots, \tilde{M}(A_n)$ are independent with:

$$\mathbb{E}\left[\tilde{M}(A_i)\right] = 0 \qquad Var\left[\tilde{M}(A_i)\right] = \mu(A_i).$$

1.8 Lévy processes

Lévy processes can be seen as the continuous time equivalent of discrete time random walks.

Definition 1.33 (Lévy process). A cadlag stochastic process $(X_t)_{t\geq 0}$ is said to be a Lévy process on $(\Omega, \mathcal{F}, \mathbb{P})$ if it possesses the following properties:

1. $X_0 = 0$ $\mathbb{P}$-almost surely.
2. Independent increments: for every increasing sequence of times $\{t_0, ..., t_n\}$, the random variables $\{X_{t_0}, X_{t_1} - X_{t_0}, X_{t_n} - X_{t_{n-1}}\}$ are independent.
3. Stationary increments: the distribution of $X_{t+h} - X_t$ does not depend on t.
4. Stochastic continuity: $\forall \epsilon > 0, \lim\limits_{h\to 0} \mathbb{P}\left(|X_{t+h} - X_t| \geq \epsilon\right) = 0$.

Note that the last condition does not imply in any way that the *sample paths* are continuous! It merely serves to exclude processes with jumps at fixed nonrandom times, which in the world of finance might be interpreted as "calendar effects". It means that for a given time t, the probability of seeing a jump at t is zero: discontinuities occur exclusively at random times.

In fact, sampling a Lévy process at regular time intervals $\{0, \delta, 2\delta ...\}$ and setting:

$$S_n(\delta) := \sum_{k=0}^{n-1} Y_k \text{ with } Y_k = X_{(k+1)\delta} - X_{k\delta},$$

gives a random walk. Y_k are i.i.d random variables with $Y_k \overset{d}{=} X_\delta$. This can be done for any sampling interval δ, giving rise to a whole family of random walks $S_n(\delta)$ associated with a given Lévy process X_t.

For any $t > 0$, choosing n, δ such that $n\delta = t$, we can see that $X_t = S_n(\delta)$ can be represented as a sum of n i.i.d. random variables whose distribution is the same as that of $X_{\frac{t}{n}}$. We can "divide" X_t into n i.i.d parts. This motivates the following definition.

Definition 1.34 (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 $Y_1, ..., Y_n$ such that $\sum\limits_{k=1}^{n} Y_k$ has distribution F.

An infinitely divisible distribution can also be defined as a distribution F for which the n-th convolution root is again a probability distribution, for any $n \geq 2$. The distribution of i.i.d. sums is given by convolution of the distribution of the summands. If $Y_k \sim \mu$ then $F = \mu * \cdots * \mu$ is the n-th convolution of μ.

It follows, for a Lévy process X_t and any $t > 0$ the distribution of X_t is infinitely divisible. Unlike in the case of the discrete time random walk, where we are free to choose the distribution of the increments, the distribution of increments of a Lévy process has to be infinitely divisible.

Some common examples of infinitely divisible distribution are: the Gaussian distribution, the gamma distribution, α-stable distributions and the Poisson distribution. A random variable having any of these distributions can be decomposed into a sum of n i.i.d. parts having the same distribution but with modified parameters. Less trivial examples are the log-normal, Pareto, Student distributions. An example of a distribution which is not infinitely divisible is the uniform distribution on an interval.

It also works the other way around. By chopping an infinitely divisible distribution F into $n \geq 1$ i.i.d components, we can construct a random walk model on a time grid with step size $\frac{1}{n}$, such that the law of the position at time $t = 1$ is given by F. In the limit, a Lévy process X_t with $X_1 \sim F$ can be constructed. The close relationship between Lévy processes and infinitely divisible distributions is summarized in the following proposition.

Result 1.35 (Infinite divisibility and Lévy processes)**.** Let X_t be a Lévy process. Then for every t, X_t has an infinitely divisible distribution. Conversely, if F is an infinitely divisible distribution then there exists a Lévy process (X_t) such that the distribution of X_1 is given by F.

1.8.1 Characteristic function

The characteristic function of a Lévy process X_t is a time-dependent function given by:

$$\Phi_t(w) = \Phi_{X_t}(w) = \mathbb{E}\left[e^{iw^T X_t}\right], w \in \mathbb{R}^d.$$

For $t, s \in \mathbb{R}^+$ we can write $X_{t+s} = X_s + (X_{t+s} - X_s)$. Because of the independent increments property of Lévy processes, $X_{t+s} - X_s$ is independent of X_s. We obtain that $t \mapsto \Phi_t(z)$ is a multiplicative function:

$$\Phi_{t+s}(z) = \Phi_{X_{t+s}}(z) = \Phi_{X_s}(z)\Phi_{X_{t+s}-X_s}(z) = \Phi_{X_s}(z)\Phi_{X_t}(z) = \Phi_s(z)\Phi_t(z).$$

The stochastic continuity of $t \mapsto X_t$ implies convergence in distribution of $X_s \xrightarrow[s \to t]{d} X_t$. Therefore, $\Phi_{X_s}(z) \to \Phi_{X_t}(z)$ for $s \to t$ so $t \mapsto \Phi_t(z)$ is a continuous function of t. Together with the multiplicative property this implies that it is an exponential function.

Definition 1.36 (Characteristic exponent of a Lévy process)**.** Let $(X_t)_{t \geq 0}$ be a Lévy process on $\mathbb{R}^d$. The *characteristic exponent* of X is a continuous function $\psi : \mathbb{R}^d \mapsto \mathbb{R}$ such that:

$$\mathbb{E}\left[e^{iz^T X_t}\right] = e^{t\psi(z)}, z \in \mathbb{R}^d.$$

In fact, ψ is the cumulant generating function of X_1, and the cumulant generating function of X_t varies linearly in t. The distribution of X_t is determined by the knowledge of the distribution of X_1. We now see that the only degree of freedom we have in specifying a Lévy process is specifying the distribution of X_t for a single time (usually $t = 1$)!

1.8.2 Compound Poisson process

A well known example of a Lévy process is the compound Poisson process.

Definition 1.37 (Compound Poisson process)**.** A compound Poisson process with intensity $\lambda > 0$ and jump size distribution F is a stochastic process X_t defined as

$$X_t = \sum_{i=1}^{N_t} Y_i,$$

where jumps sizes Y_i are i.i.d with $Y_i \sim F$ and $(N_t)_{t \geq 0}$ is a Poisson process with intensity λ, independent from the sequence $(Y_i)_{i \geq 1}$.

Its characteristic function has the following form:

$$\mathbb{E}\left[e^{iw^T X_t}\right] = \exp\left(t\lambda \int_{\mathbb{R}^d} \left(e^{iw^T x} - 1\right) f(dx)\right), \forall w \in \mathbb{R}^d.$$

The sample paths of a compound Poisson process are piecewise constant functions. In fact, it can be proven that it is the only Lévy process with this property.

Result 1.38 ([2])**.** $(X_t)_{t \geq 0}$ is a compound Poisson process if and only if it is a Lévy process and its sample paths are piecewise constant functions.

1.8.3 Jump and Lévy measure

We now review the concept of a jump measure.

Definition 1.39 (Jump measure)**.** Let X_t be a càdlàg stochastic process. For any measurable set $B \subset \mathbb{R}^d \times [0, \infty)$ the jump measure of X is defined as:

$$J_X(B) = \#\left\{t, X_t - X_{t-} \in B\right\}.$$

For every measurable set $A \in \mathbb{R}^d$, $J_X(A \times [t_1, t_2])$ counts the number of jumps of X occurring in the interval $[t_1, t_2]$ whose amplitude belongs to A.

Result 1.40 (Jump measure of a compound Poisson process [2])**.** Let $(X_t)_{t \geq 0}$ be a compound Poisson process with intensity λ and jump size distribution F. The jump distribution of X is a Poisson random measure on $\mathbb{R}^d \times [0, \infty)$ with intensity measure $\mu(dx \times dt) = \nu(dx)dt = \lambda F(dx)dt$.

Definition 1.41 (Lévy measure)**.** Let $(X_t)_{t \geq 0}$ be a Lévy process on $\mathbb{R}^d$. The measure ν on $\mathbb{R}^d$. The Lévy measure of X is defined by:

$$\nu(A) = \mathbb{E}\left[\#\{t \in [0,1] : \Delta X_t \neq 0, \Delta X_t \in A\}\right], A \in \mathcal{B}(\mathbb{R}^d)$$

$\nu(A)$ is the expected number, per unit time, of jumps whose size belongs to A.

1.8.4 Lévy-Ito decomposition

Let (X_t^0) be a piecewise constant Lévy process. From result 1.38 it follows, that X_t^0 is a compound Poisson process with some jump intensity λ and jump size distribution F. By introducing a Brownian motion with drift $\gamma t + W_t$, independent from X^0, we can define a new Lévy process:

$$X_t = X_t^0 + \gamma t + W_t,$$

which admits the following decomposition:

$$\begin{aligned} X_t = \gamma t + W_t + \sum_{s \in [0,t]} \Delta X_s &= \gamma t + W_t + \int\limits_{[0,t] \times \mathbb{R}^d} x J_x(ds \times dx) \\ &= \gamma t + W_t + \lambda \int_0^t \int\limits_{\mathbb{R}^d} x F(dx) ds. \end{aligned}$$

Does every Lévy process admit such a decomposition? Given an arbitrary Lévy process (X_t), we can still define its Lévy measure ν as above. Lévy measure $\nu(A)$ of any compact subset A with $0 \notin A$ is finite, otherwise the process would have an infinite number of finite jumps on $[0, T]$, contradicting its càdlàg property. Therefore, ν is a Radon measure on $\mathbb{R}^d \setminus \{0\}$. However, ν is not necessarily finite, it can still blow up at zero. X may exhibit an infinite number of small jumps on $[0, T]$, that add up to an infinite series that converges only if we impose additional conditions on the measure ν.

Result 1.42 (Lévy-Ito decomposition [2]). Let $(X_t)_{t\geq 0}$ be a Lévy process on $\mathbb{R}^d$ and ν its Lévy measure.

- ν is a Radon measure on $\mathbb{R}^d \setminus 0$ and verifies:

$$\int\limits_{|x|\leq 1} |x|^2 \nu(dx) < \infty, \int\limits_{|x|\geq 1} \nu(dx) < \infty$$

- The jump measure of X, indicated denoted by J_X, is a Poisson random measure on $[0, \infty) \times \mathbb{R}^d$ with intensity measure $\nu(dx)dt$.
- There exist a vector γ and a d-dimensional Brownian motion $(W_t)_{t\geq 0}$ with covariance matrix A such that:

$$\begin{aligned} X_t &= \gamma t + W_t + X_t^l + \lim_{\epsilon \to 0} \tilde{X}_t^\epsilon, \text{where} \\ X_t^l &= \int\limits_{|x|\geq 1, s\in[0,t]} x J_X(ds \times dx) \\ \tilde{X}_t^\epsilon &= \int\limits_{\epsilon\leq|x|\leq 1, s\in[0,t]} x\left(J_X(ds \times dx) - \nu(dx)\right) \\ &= \int\limits_{\epsilon\leq|x|\leq 1, s\in[0,t]} x\tilde{J}_X(ds \times dx). \end{aligned} \tag{1.6}$$

  The terms in (1.6) are independent and the convergence in the last term is almost sure and uniform in t on $[0, T]$.

The first term, $\gamma t + W_t$, is a continuous Gaussian Lévy process. Every Gaussian Lévy process is continuous and can be written in this form. In fact, the only continuous Lévy process is the Brownian motion with drift, so the continuous part of any Lévy process can be represented in this form.

The other two terms, X_t^l and $\tilde{X}_t^\epsilon$ are discontinuous processes incorporating the jumps of X_t, described by the Lévy measure ν. The condition $\int_{|y|\geq 1} \nu(dy) < \infty$ ensures that the number of jumps of X with absolute value larger than 1 is finite. Then the sum:

$$X_t^l = \sum_{0\leq s\leq t}^{|\Delta X_s|\geq 1} \Delta X_s$$

has an almost surely finite number of terms and X_t^l is a compound Poisson process. Note that the threshold $\Delta X = 1$ is completely arbitrary. For any $\epsilon > 0$, the sum of jumps with amplitude in $[\epsilon, 1)$:

$$X_t^\epsilon = \sum_{0\leq s\leq t}^{1>|\Delta X_s|\geq\epsilon} \Delta X_s = \int\limits_{\epsilon\leq|x|<1, s\in[0,t]} x J_X(ds \times dx)$$

is a well-defined compound Poisson process. However, ν can still have a singularity at zero due to infinitely many small jumps. In order to obtain convergence when $\epsilon \to 0$, we replace the integral by its compensated version:

$$X_t^\epsilon = \int\limits_{\epsilon\leq|x|<1, s\in[0,t]} x\tilde{J}_X(ds \times dx)$$

which is a martingale.

From the Lévy-Ito decomposition it follows that for every Lévy process (X_t) there exist a vector γ, a positive definite matrix A and a positive measure ν that uniquely determine its distribution. The triplet (A, ν, γ) is called the *Lévy triplet* or *characteristic triplet* of the process (X_t).

1.8.5 Lévy-Khinchin representation

The knowledge of the Lévy-Ito decomposition process allows us to derive another fundamental result of the theory, an expression for the characteristic function in terms of its Lévy triplet (A, ν, γ).

Result 1.43 (Lévy-Khinchin representation [2]). Let $(X_t)_{t\geq 0}$ be a Lévy process on $\mathbb{R}^d$ with characteristic triplet (A, ν, γ). Then the characteristic function is of the following form:

$$\mathbb{E}\left[e^{iz^T X_t}\right] = e^{t\psi(z)}, z \in \mathbb{R}^d$$
$$\psi(z) = -\frac{1}{2}z^T Az + i\gamma^T z + \int_{\mathbb{R}^d} \left(e^{iz^T x} - 1 - iz^T x 1_{|x|\leq 1}\right) \nu(dx).$$

If the Lévy measure satisfies the additional condition $\int_{|x|>1} |x|\, \nu(dx) < \infty$ then one can use the simpler form, since there is no need to truncate large jumps:

$$\psi(z) = -\frac{1}{2}z^T Az + i\gamma_c^T z + \int_{\mathbb{R}^d} \left(e^{iz^T x} - 1 - iz^T x\right) \nu(dx), \text{ with}$$
$$\gamma_c = \gamma + \int_{|x|>1} x\nu(dx).$$

More generally, one can replace $1_{|x|<1}$ by any bounded measurable function $g : \mathbb{R}^d \to \mathbb{R}$ satisfying $g(x) = 1 + o(|x|)$ as $x \to 0$ and $g(x) = O(\frac{1}{|x|})$ as $x \to \infty$.

$$\psi(z) = -\frac{1}{2}z^T Az + i\gamma_g^T z + \int_{\mathbb{R}^d} \left(e^{iz^T x} - 1 - iz^T x g(x)\right) \nu(dx), \text{where}$$

The function g is called a *truncation function* and the characteristic triplet (A, ν, γ_g) is called the characteristic triplet of X w.r.t. the truncation function g. It is important to remember that A and ν are independent of the choice of g, but γ is not. For this reason one should avoid calling it the "drift" of the process.

1.8.6 Exponential moments

Result 1.44 (Exponential moments of a Lévy process [2]). Let $(X_t)_{t\geq 0}$ be a Lévy process on $\mathbb{R}$ with characteristic triplet (A, ν, γ) and ψ its characteristic exponent. Then it holds:

$$\mathbb{E}\left[e^{uX_t}\right] < \infty \iff \int_{|x|\geq 1} e^{ux}\nu(dx) < \infty,$$
$$\text{in this case: } \mathbb{E}\left[e^{uX_t}\right] = e^{t\psi(-iu)}.$$

Definition 1.45 (Total variation). Total variation of a function $f : [a, b] \to \mathbb{R}^d$ is defined by:

$$TV(f) = \sup_{a=t_0<t_1<\cdots<t_n=b} \sum_{i=1}^{n} |f(t_i) - f(t_{i-1})| \tag{1.7}$$

In one dimension every increasing or decreasing function is of finite variation and every function of finite variation is a difference of two increasing functions.

1.8.7 Lévy processes and martingales

Martingales play a crucial role in mathematical finance. The independent increment property allows us to construct different martingales from Lévy processes.

Result 1.46 (Martingale construction using Lévy processes [2])**.** Let $(X_t)_{t\geq 0}$ be an $\mathbb{R}$-valued process with independent increments. Then it holds:

1. $\left(\frac{e^{iwX_t}}{\mathbb{E}[e^{iwX_t}]}\right)_{t\geq 0}$ is a martingale $\forall w \in \mathbb{R}$.
2. If for some $u \in \mathbb{R}$, $\forall t > 0$ $\mathbb{E}\left[e^{uX_t}\right] < \infty$ then $\left(\frac{e^{uX_t}}{\mathbb{E}[e^{uX_t}]}\right)_{t\geq 0}$ is a martingale.
3. If $\forall t \geq 0$ $\mathbb{E}\left[|X_t|\right] < \infty$ then $M_t := X_t - \mathbb{E}\left[X_t\right]$ is a martingale with independent increments.
4. If $\forall t \geq 0$ $Var\left[X_t\right] < \infty$ then $(M_t)^2 - \mathbb{E}\left[(M_t)^2\right]$ is a martingale.

When X is a Lévy process, it suffices that the corresponding moments be finite for one value of t for the above processes to be martingales.

The following proposition gives us a way to check if a given Lévy process X is a martingale.

Result 1.47 (When is a Lévy process a martingale? [2])**.** Let $(X_t)_{t\geq 0}$ be an $\mathbb{R}$-valued Lévy process with characteristic triplet (A, ν, γ).

1. (X_t) is a martingale $\iff \int\limits_{|x|\geq 1} |x|\,\nu(dx) < \infty$ and $\gamma + \int\limits_{|x|\geq 1} x\nu(dx) = 0$.
2. (e^{X_t}) is a martingale $\iff \int\limits_{|x|\geq 1} e^x \nu(dx) < \infty$ and $\frac{A}{2} + \gamma + \int\limits_{-\infty}^{\infty} \left(e^x - 1 - x1_{|x|\leq 1}\right)\nu(dx) = 0$.

1.9 Additive processes

Additive processes are obtained from Lévy processes by relaxing the condition of stationarity of increments.

Definition 1.48 (Additive process)**.** A stochastic process $(X_t)_{t\geq 0}$ on $\mathbb{R}^d$ is called an *additive process*, if it is càdlàg , satisfies $X_0 = 0$ and has the following properties:

1. Independent increments: for every increasing sequence of times $t_0 \dots t_n$, the random variables $X_{t_0}, X_{t_1} - X_{t_0}, \dots, X_{t_n} - X_{t_{n-1}}$ are independent.
2. Stochastic continuity: $\forall \epsilon > 0, \mathbb{P}\left[|X_{t+h} - X_t|\right] \to 0$ for $h \to 0$.

Result 1.49 (Lévy processes with deterministic volatility [2])**.** Consider a continuous function $\sigma(t) : \mathbb{R}^+ \to \mathbb{R}^+$. Let $(L_t)_{t\geq 0}$ be a Lévy process on $\mathbb{R}$. Then

$$X_t = \int\limits_0^t \sigma(s)dL_s$$

is an additive process. The independent increments property is straightforward. One way to show the stochastic continuity is to decompsose L into a sum of a compound Poisson process incorporating big jumps and the residual process having only jumps of size smaller than one. Then the integral with respect to the Poisson part can be easily shown to be stochastically continuous and the rest can be shown to be L^2-continuous hence also stochastically continuous.

Result 1.50 (Properties of additive processes, theorems 9.1-9.8 in [5])**.** Let $(X_t)_{t\geq 0}$ be a an additive process in $\mathbb{R}^d$. Then, for all t, X_t has an infinitely divisible distribution. The distribution of $(X_t)_{t\geq 0}$ is uniquely determined by its spot characteristics $(A_t, \mu_t, \Gamma_t)_{t\geq 0}$:

$$\mathbb{E}\left[e^{iu^T X_t}\right] = e^{\Psi_t(u)}$$

$$\Psi_t = -\frac{1}{2}u^T A_t u + iu\Gamma_t + \int\limits_{\mathbb{R}^d} \left(e^{iu^T x} - 1 - iu^T x 1_{|x|\leq 1}\right) \nu_t(dx).$$

The spot characteristics $(A_t, \mu_t, \Gamma_t)_{t\geq 0}$ satisfy the following conditions:

1. For all t, A_t is a positive definite $d \times d$ matrix and μ_t is a positive measure on $\mathbb{R}^d$ satisfying $\mu_t(\{0\}) = 0$ and $\int\limits_{\mathbb{R}^d} (|x|^2 \wedge 1)\mu_t(dx) < \infty$.
2. Positiveness: $A_0 = 0, \nu_0 = 0, \gamma_0 = 0$ and for all s, t sucht that $s \leq t$, $A_t - A_s$ is a positive definite matrix and $\mu_t(B) \geq \mu_s(B)$ for all measurable sets $B \in \mathcal{B}(\mathbb{R}^d)$.
3. Continuity: if $s \to t$ then $A_s \to A_t$, $\Gamma_s \to \Gamma_t$ and $\mu_s(B) \to \mu_t(B)$ for all $B \in \mathcal{B}(\mathbb{R}^d)$ such that $B \subset \{x : |x| \geq \epsilon\}$ for some $\epsilon > 0$.

Conversely, for a family of triplets $(A_t, \mu_t, \Gamma_t)_{t\geq 0}$ satisfying the above conditions, there exists and additive process $(X_t)_{t\geq 0}$ with $(A_t, \mu_t, \Gamma_t)_{t\geq 0}$ as spot characteristics.

Examples of spot characteristics $(A_t, \mu_t, \Gamma_t)_{t\in[0,T]}$ satisfying these conditions can be constructed using:

- A continous, matrix valued function $\sigma : [0, T] \to M_{d\times n}(\mathbb{R})$ such that $\sigma(t)$ is symmetric and $\int\limits_0^T \sigma^2(t)dt < \infty$.
- A family $(\nu_t)_{t\in[0,T]}$ of Lévy measures verifying

$$\int\limits_0^T \int (1 \wedge |x|^2)\nu_t(dx)dt < \infty.$$

- A deterministic function with finite variation $\gamma : [0, T] \to \mathbb{R}$ (e.g. a piecewise continuous function).

Then the spot characteristic $(A_t, \mu_t, \Gamma_t)_{t\in[0,T]}$ defined by

$$A_t = \int\limits_0^t \sigma^2(s)ds$$

$$\mu_t(B) = \int\limits_0^t \nu_s(B)ds \forall B \in \mathcal{B}(\mathbb{R}^d)$$

$$\Gamma(t) = \int\limits_0^t \gamma(s)ds,$$

satisfy the conditions and therefore define a unique additive process $(X_t)_{t\in[0,T]}$ with spot characteristics. $(\sigma^2(t), \nu_t, \gamma(t))_{t\in[0,T]}$ are called local characteristics of the additive process.

1.10 Stochastic calculus

We now recall some fundamental results from stochastic calculus.

Definition 1.51. A *semi-martingale* X is an adapted stochastic process that admits the following decomposition

$$X = X_0 + M + A \tag{1.8}$$

where M is a local martingale and A a process of finite (total) variation. If $M_0 = A_0 = 0$, (1.8) is called a semi-martingale decomposition of X.

In general this decomposition in not unique. For instance, consider the compensated Poisson process $N - \lambda t$, where N is a Possion process with intensity λ. This process is of finite variation and also a martingale, hence it is a matter of discretion as to wheter M or A is identically zero.

Definition 1.52 (Predictable σ-algebra, predictable process)**.** The predictable σ-algebra $\mathcal{P}$ on $\mathbb{R}^+ \times \Omega$ is the σ-algebra generated by the adapted, left-continuous processes. A $\mathcal{P}$-measurable process is called predictable.

For instance, every continuous-time adapted process that is left continuous is a predictable process

Definition 1.53. A *special semi-martingale* is a semi-martingale X with decomposition $X = X_0 + M + A$ such that the finite variation part A is *predictable.* This *canonical decomposition* is unique.

Back to our example of the compensated Poisson process, it is now clear that, since $N - \lambda t$ is not predictable, it forms the martingale part of the canonical decomposition and the finite variation part is identically zero.

Definition 1.54 (Quadratic variation)**.** Let X be a semi-martingale and (Π_n) a sequence of random partitions. Each random partition is a finite sequence of finite stopping times $0 = T_0^n \leq T_1^n \leq \ldots T_{k_n}^n = T$. The grid size $\|\Pi_n\| = \sup_i |T_{i+1}^n - T_i^n|$ of the sequence (Π_n) is assumed to converge a.s. to zero. The *quadratic variation process* $[X]$ is the increasing, adapted process defined by

$$[X] = \lim_n \sum_i \left(X^{T_{i+1}^n} - X^{T_i^n} \right)^2 \qquad \text{in } \textit{ucp.}$$

The new notion of convergence for stochastic processes used here is *ucp*, that is, uniformly on compacts in probability. A sequence (X^n) of processes converges to a process X in this sense if $\sup_{0 \leq t \leq T} |X_t^n - X_t|$ converges to 0 in probability.

Definition 1.55 (Quadratic covariation)**.** The *quadratic co-variation* or *square bracket process*

$$[X, Y] = \frac{1}{2}([X + Y] - [X] - [Y])$$

for two semi-martingales X and Y, can be defined by polarisation.

The quadratic co-variation process is a bilinear form which is invariant under equivalent probability measure changes. This follows since a sequence converges in probability if and only if one can extract from every subsequence an almost surely convergent subsequence, and the null sets are the same for equivalent measures.

Examples of the quadratic variation of some common processes include:

1. Brownian motion B: $[B]_t = t$.
2. Pure jump process X: $[X]_t = \sum_{s \leq t} (\Delta X_s)^2$.
3. Poisson process N: $[N] = N$ since the jump sizes equal one.
4. Continuous processes X of finite variation: $[X] = 0$.

Definition 1.56 (L^2-martingale)**.** A martingale M is called *square-integrable* (in short: L^2-martingale) if $E[[M]_T] < \infty$. A semi-martingale X is called *square-integrable* if it is special with canonical decomposition $X = M + A$ where M is an L^2-martingale and A has square-integrable variation.

Result 1.57 (Conditional quadratic variation [4])**.** Let X, Y be locally square integrable semi-martingales. Then there exists a unique predictable process $\langle X, Y\rangle$, called *angle bracket process*, such that

$$[X, Y] - \langle X, Y\rangle$$

is a local martingale.

We set $\langle X\rangle = \langle X, X\rangle$. More generally, $\langle X, Y\rangle$ exists if $[X, Y]$ is of locally integrable variation.

Note that since the martingale property is not preserved by a change of measure, the angle bracket process is *not* invariant under measure changes and may even not exist under an equivalent measure. Despite this, it inherits many properties from the square bracket:

1. $\langle X\rangle$ is increasing
2. $\langle X, Y\rangle$ is a bilinear form
3. When X, Y are both continuous, $\langle X, Y\rangle = [X, Y]$. More generally:

$$[X]_t = \langle X\rangle_t + \sum_{s\leq t}(\Delta X_s)$$

Some examples:

1. Brownian motion: $\langle B\rangle_t = [B]_t = t$.
2. Poisson process N with intensity λ: $\langle N\rangle_t = \lambda t$.

This second example is a bit tricky, since the equality is not due to the fact that $N - \lambda t$ is a martingale; but rather that $[N] - \lambda t$ is a martingale because $[N] = N$.

We now turn our attention to stochastic integration. The definition of a stochastic integral with respect to semi-martingale integrators is well beyond the scope of this thesis. We refer to [3] for that matter. We simply focus on recalling some main properties of stochastic integrals that will be used.

Let X be a semi-martingale. In the case when $\int \vartheta dX$ is well-defined, we call ϑ *integrable* with respect to X, consult [3] for details. The set of X-integrable processes, which we will not characterise here, is denoted by $L(X)$. It is important to note that the only reasonable integrands are *predictable* processes.

Result 1.58 (Quadratic co-variation of stochastic integrals [4])**.** Let X, Y be two semi-martingales, and $\vartheta \in L(X)$, $\eta \in L(Y)$. Then

$$\left[\int \vartheta dX, \int \eta dY\right] = \int \vartheta\eta d[X, Y].$$

In particular, the quadratic variation of the stochastic integral process $\int \vartheta dX$ is given by

$$\left[\int \vartheta dX\right] = \int \vartheta^2 d[X].$$

We now recall the fundamental rules of calculus for stochastic integrals.

Result 1.59 (Integration by parts [4])**.** Let X, Y be two semi-martingales. Then XY is a semi-martingale and

$$XY - X_0Y_0 = \int X_- dY + \int Y_- dX + [X, Y].$$

The following result is a generalisation of the chain rule.

Result 1.60 (Itô's formula [4])**.** Let X be a semi-martingale, and $f : \mathbb{R}_+ \times \mathbb{R} \to \mathbb{R}$ a $\mathcal{C}^{1,2}$-function. Then $f(t, X_t)$ is a semi-martingale, and we have, with $[X^c]_t = [X]_t - \sum_{s \le t} (\Delta X_s)^2$,

$$\begin{aligned} f(t, X_t) - f(0, X_0) = &\int_0^t \frac{\partial}{\partial t} f(s, X_s) ds + \int_0^t f'(s, X_{s-}) dX_s \\ &+ \frac{1}{2} \int_0^t f''(s, X_s) d[X^c]_s \\ &+ \sum_{0 < s \le t} \{ f(s, X_s) - f(s, X_{s-}) - f'(s, X_{s-}) \Delta X_s \}. \end{aligned}$$

Result 1.61 (Martingale property of stochastic integrals [4])**.** For a local martingale integrator M, and a locally bounded predictable ϑ, the stochastic integral process $\int \vartheta dM$ is a local martingale. The same conclusion holds in case ϑ is adapted and left-continuous with right-limits.

Chapter 2

Electricity markets

In 1981, Chile was the first country to deregulate its electricity market and quote an electricity spot price, laying the foundation of modern electricity trading. Twelve years later, in 1993, the first future contracts on electricity were quoted on the Scandinavian market 1993. Ever since, quantitative finance literature has followed the liberalisation of the electricity industry all over the world.

Electricity as a form of energy is used in a very wide range of applications. It has the advantage of being relatively easily controllable, portable and non-polluting at the location of its usage. Since every electron is the same, quality is not a question in that sense. As a secondary energy source electricity is generated by conversion of other energy sources, like coal, natural gas, oil, nuclear power, hydropower and other renewable sources.

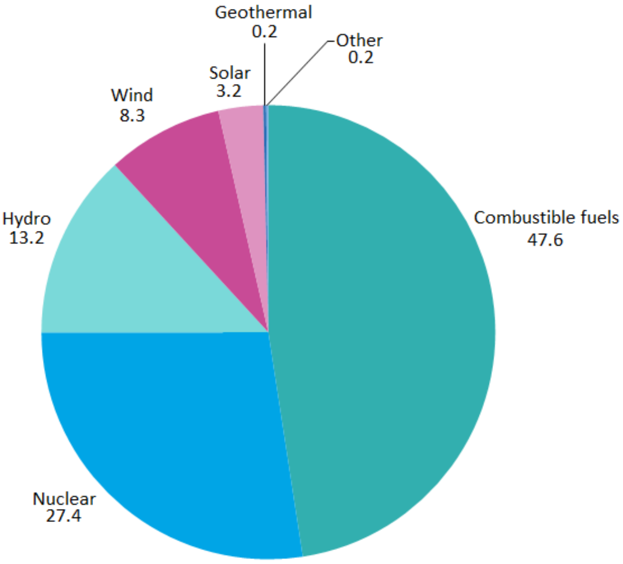


Figure 2.1: Percentage of total net electricity generation per source in EU-28 for year 2014. Percentages do not sum to 100% due to rounding. Source: Eurostat (online data coda: `nrg_105a`).

This entails that electricity markets and electricity prices are fundamentally linked to markets for primary fuels and environmental conditions. A understanding of the electricity generation process and fuel markets is a prerequisite for a sound grasp of electricity markets and price mechanisms.

Electricity as a commodity, due to its unique properties, differs significantly not only from stocks, bonds and other frequently studied financial instruments, but from other commodities as well. These fundamental properties impact both the constitution of the electricity market as well as price behaviour. Some of characteristic properties of electricity and electricity markets include:

1. *Electricity cannot be stored at a reasonable cost.* Since electricity consumption is a continuous phenomenon, this raises an energy and a capacity problem (the rate at which energy can be released). Perhaps somewhat surprisingly, the most economical way of storing a large amount of electricity are still hydroelectric reservoirs. However, this is far from a universal solution because it relies on the hydroelectric potential of a country. The hydroelectric capacity of most countries is small compared with their total consumption. Also, enough capacity has to be available in order to satisfy the power demand at any point in time. For instance, the total installed capacity of France is around 110GW while the annual maximum power demand has reached 100GW in the last several decades. One possible solution of the capacity problem would be installing enough capacity to satisfy demand in any conceivable scenario. However, this would require too many power plants that would be used for only a few hours in their lifetime, making it expensive and impractical.

2. *The transport of electricity adheres to specific laws.* The transport of electricity follows Kirchhoff's laws. Basically, the laws state that the intensity at each node should be zero and the tension in each loop should also be zero. It then follows that, in a meshed electricity network, power goes from one point to another through all available paths. Because of this, the transfer capacities require some electricity generation hypothesis before they can be computed.

3. *Electricity price evolution process is unique.* A quick look at the empirical data suffices to reveal the truly unusual characteristics of electricity prices. What first captures the eye is the extraordinarily high volatility. Indeed, the volatility of electricity is an order of magnitude higher than the volatility of foreign exchange rates, interest rates, or equity prices. It is even higher than all other energy markets. Just compare the typical volatility of USDE/JPY or USD/EUR exchange rates (10%–20%), LIBOR rates (10%–20%), SP500 index (20%–30%), NASDAQ (30%–50%), natural gas prices (50%–100%) and electricity (100%–500% and higher). Electricity prices also exhibit pervasive spikes of extraordinary magnitude, mean reversion, regime switching, stochastic volatility, volatility smiles, and the host of other interesting properties that make processes describing the evolution of energy prices very different from their more standard financial counterparts. Correlation across electricity markets is another distinguishing feature and the importance of joint distributions is impossible to underestimate.

4. *Electricity derivatives are unique.* Standard products like futures, forwards, and options still have unique features due to their physical nature. They are settled and defined differently compared to their counterparts in the financial markets. Power delivered at any particular hour, block of hours, day, week, month, and so on, represents a different commodity due to the non-storability property. In addition, specific sorts of derivatives, such as swing options, power plant and tolling agreements, which have been developed to manage risks associated with meeting the demand, have no parallel in financial markets. These more complex derivatives in the electricity markets are often path dependent, making their valuation much harder.

Since electricity requires a grid infrastructure, electricity markets are more regional than other energy markets. However, over the past decades significant effort has been put towards integrating neighbouring market zones (e.g., Europe) with the goal of fostering competition and optimising usage of the grid and generation infrastructure.

Traditionally, electricity markets of many countries were dominated by vertically integrated "incumbent" utility companies that owned generation assets, grid infrastructure and the retail business. Grid ownership has put those incumbents in a natural monopoly position with high entry barriers for potential competitors. Very often, incumbent utilities were either state owned or at least regulated. Since the 1990s some form of liberalisation of electricity markets has taken place in many countries. The details of market design and regulation still differ substantially between countries, but share some main elements:

1. A distinction between natural monopoly areas (e.g., grid operation) and areas where competition shall be established. A clear separation of monopoly areas from competition areas is necessary, as

to make access to grid and other infrastructure non-discriminatory for all market participants.

2. Wholesale markets should be designed in a way that incentivises optimal economic usage of infrastructure, such as power plants and interconnections between market areas. In addition, sufficient incentives should be provided for building new generation capacity if required.
3. Established regulation ensuring security of supply and preventing market abuse.
4. Incorporated mechanisms for environmental protection (e.g., carbon emissions).

We distinguish between the transport and the distribution level of electricity. That is, the level at which the wholesale markets operate and the level at which the electricity is delivered to the consumer. The object of our study will be models of the prices in the the wholesale market, i.e., at the transport level. However, it is important to emphasize that, although the distribution network was designed to transfer power from the transport level to the consumer, with the introduction and increased presence of renewable energy sources at the distribution level, now power sometimes flows "upward" to the transport network. In future, this may require an adaptation in the way distribution networks are monitored and operated (e.g., smart grids).

Liberalisation of the electricity market on the EU territory started in the UK at the beginning of the 1990s, soon followed by Scandinavia. In 1996 and 2003 the Electricity Market Directives were issued by the European Commission defining steps for deregulation of electricity markets on an EU-wide scale, which were subsequently to be implemented by the EU member states. The goal of these measures was to foster competition and improve integration of national electricity wholesale markets. Since the introduction of the European Trading System (EU ETS) for carbon emissions in 2005, the European electricity markets are closely connected to European environmental policy, since emission certificates (EUAs) have become an important driver for wholesale electricity prices. More recently, a new important driver has arisen in the form of increasing solar and wind generation fostered by national subsidy schemes, with ambitious growth targets (e.g., in Germany).

2.1 Electricity trading

In case of electricity, contrary to other commodities, it is not as obvious what the actual trading product is. For instance, unlike in the case of oil, an individual cannot simply buy electricity at 10am in the morning and sell it at 2pm when the price is higher. In contrast to other commodities electricity is hardly storable.

Non-storability of electricity, in case of of a tight or excessive supply situation, can result in spiky behaviour and high volatility of power prices in the spot market. In the forward market the price exhibits much smaller movements, because the availability of power plants and the weather-dependent demand are still unknown at the time. Another main property of electricity preventing a global market is the necessity for a transmission network.

The lack of storability requires an exact matching of supply and demand at any point in time. Since it is impossible for the merchant to forecast the demand of his customers exactly, an entity responsible for keeping the system balanced was introduced. This is the duty of the transmission system operator (TSO), who charges the merchant directly or the retail customers via transmission fees for this service. A balancing period, defined by the TSO, is the granularity of the measured electric energy supply. The continuously varying power requirements of retail customers are integrated over the balancing period (e.g., 15 minutes in Germany, 30 minutes in the UK) and the average power is the size that is forecast and should be delivered by the supplying merchant. This results in the merchant delivering energy on a discrete time series with time steps according to the balancing period and constant power during these time periods. Figure 2.2 illustrates this.

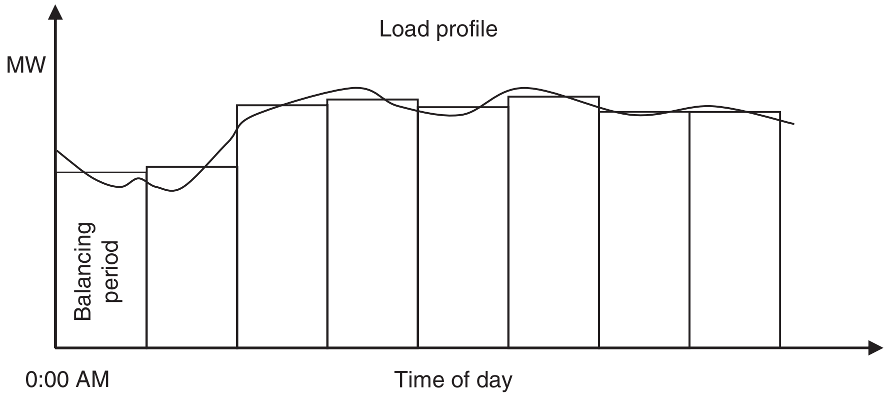


Figure 2.2: Balancing period.

The principal products traded in the electricity markets are delivery schedules in a granularity not finer than the balancing period. The usual granularity is one hour. The power balancing during the balancing period itself is the task of the TSO. Usually the TSO has no own-generation capacities and has to buy products which allow the increase or decrease of production (including import and export) in its transmission system at short notice.

We now take a look at the different products traded in the electricity markets. Note that the products in regional markets may vary, since there is no global electricity market. Nevertheless, the electricity market can generally be divided into the following categories:

1. *Day-ahead market*: The products traded in the day-ahead market products are delivered on the next trading day. Day-ahead products are common spot products and can be traded either on a power exchange or as bilateral agreement.
2. *Futures and forward market*: The products traded in this market play an important role in risk management by enabling the participants to hedge their positions. On the end, traders actively take positions in this market providing liquidity for the hedgers. The agreed delivery period of the products traded in this market usually refers to specific calendar weeks, calendar months or calendar years.
3. *Intra-day market*: The products traded in the intra-day market are delivered within the same day. This market is typically not a market for pure trading purposes, but instead enables the electricity producers a short-term load-dependent optimisation of their generation. Intra-day products are traded either on a power exchange or bilaterally. The intra-day market enables the participants to submit buy or sell bids up to an hour prior to delivery.
4. *Balancing and reserve market*: These markets are country and regulator specific, resulting in multiple definitions. The reserve market is the market that allows the TSO the purchase of products needed for the compensation of imbalances between supply and demand in the electricity system at short notice. The balancing market (also known as real-time market) is the market that allows the merchant to buy or sell additional additional energy in order to balance his accounting grid. The balancing service is provided by the TSO and the TSO usually charges or pays the merchant for additional energy. Only in some national markets it is possible for the merchant to buy or sell this balancing energy from or to someone other than the TSO. Therefore, the balancing market is a market only in a broader sense.

The balancing and reserve market aside, products traded in the electricity market are characterized by a discrete time series describing the delivery schedule of the product. Usually the granularity of the time series is one hour. Each value of the associated time series specifies the constant power delivered in the corresponding hour. Furthermore, we distinguish between baseload and peakload constracts. In a baseload contract the delivered power is constant over some delivery period $[T_1, T_2]$. If the delivered power is constant in the predefined delivery when the consumption is usually high, know as peakload hours, we are talking about a peakload contract. Peakload hours vary across markets. Common peakload hours are from 8:00 am to 8:00 pm on peakload days, which are usually Monday-Friday, including public holidays.

2.1.1 Day-ahead market

The day-ahead market is generally based on a fixed trading auction. Each day, up to a certain deadline (usually around 10:00 a.m.), the market participants submit bids (sales or purchases) for a particular hour of the next day or for a set of hours (order block). The bids of the market participants for a particular hour are combined into sales and purchases curves. Then, at a latter time, usually around 12:00 p.m., the market organiser performs a clearing of the market, determining the electricity price for each hour of delivery and matching the sellers with the buyers. This gives the market players enough time to send generation orders to their power plants and their schedule to the TSO.

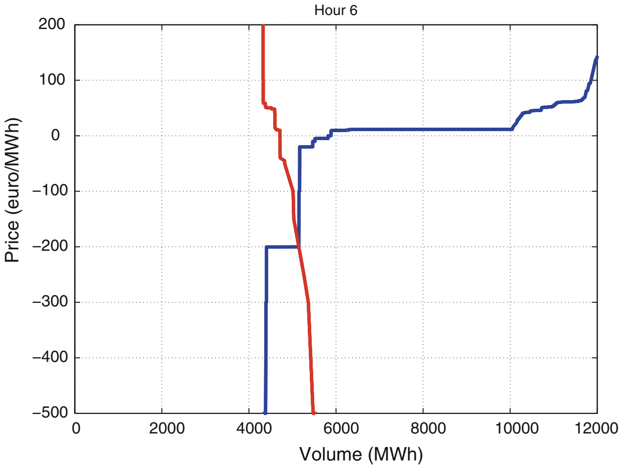


Figure 2.3: Day-ahead market volume of sales and purchases on 2013-06-16. Source: EpexSpot.

Figure 2.3 illustrates the bid and ask curves and reveals an interesting property of electricity prices – they can get negative! It is completely valid and possible to submit negative prices for buying and selling. A negative sale price means that the seller is willing to pay to sell, and a negative purchase price indicates that the buyer is willing to be paid to buy. This phenomenon, unique to the electricity market, is caused by the lack of flexibility and restart costs of some thermal power plants. Sometimes it may be less expensive to let a coal-fired plant run during hours of the day when the spot price is below its fuel cost than shutting it down and starting it up again later. On the demand side, for some buyers it might make economical sense to increase their consumption if they get compensated for the cost of changing their production schedule.

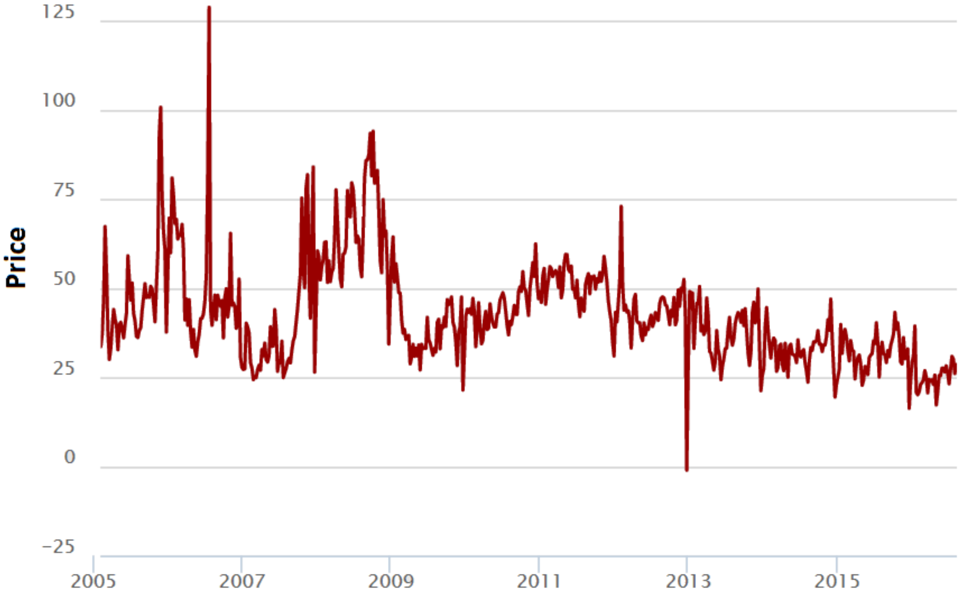


Figure 2.4: Phelix Day Base from 2005-02-08 to 2016-08-10. Source: EEX.

Figure 2.4 shows the evolution of the Physical Electricity Index (Phelix). Extreme spikes, characteristic of electricity prices, are observable, which are a consequence of the non-storability property. Furthermore, electricity day ahead prices exhibit seasonal patterns corresponding to the economic activity of a country. Daily, weekly, and annual seasonality in demand are all present, as well as intra-day seasonalities (intensive use of home appliances, sleep, closing hours of offices etc.). Electricity day-ahead prices are also characterized by fat tails, long memory and correlations with the temperatures in countries with electric heating or air conditioning.

Within the European Union, each country has its own electricity day-ahead market cleared by its own market operator. Since the day-ahead prices quoted by the market operator have a transparency function, a mechanism has been developed to ensure a consistent relation between cross-border transactions and local day-ahead prices. This mechanism consists in a market coupling process, which is further discussed in section 2.1.4.

We refer to the day-ahead market as the electricity spot market. In a sense where the spot price refers to a price for instantaneous delivery, the intraday market may appear as the real spot market. However, the reference price for delivery in the forward market is the day-ahead market. Therefore, since the relation between the spot and forward price is one of the most important relations in pricing theory, the day-ahead price is the natural choice for the spot price.

2.1.2 Futures and forward market

Similar to other futures contracts, electricity futures are usually settled daily until the end of the contract. Since electricity futures contracts do not have a single delivery date but a delivery period, the variation margin must also be calculated during the delivery period. Contracts with a long delivery period (e.g., a year or a quarter) are split into futures contracts with a shorter delivery period (e.g., a quarter or a month) through a procedure called cascading, as illustrated in fig. 2.5. On the other hand, electricity forwards are usually not settled prior to the due date. They are usually traded OTC, but some exchanges offer them as well. Organised markets even report the prices and quantities of the traded OTC contracts.

The futures market in many aspects similar to those of storable commodities such as oil, coal, or metals. Standardised contracts for future delivery of electricity specify the currency, the underlying volume, the

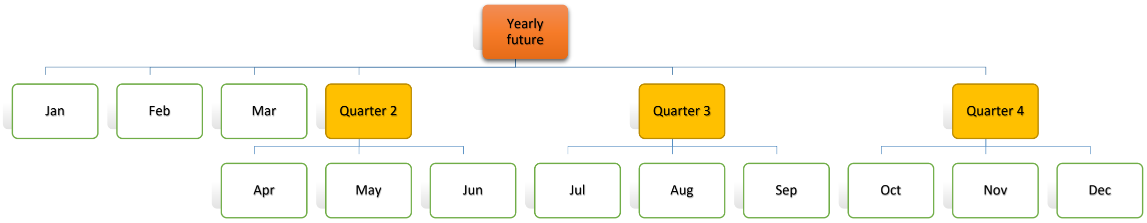


Figure 2.5: An illustration of the cascading procedure.

location of delivery, the trading period, and the tick size. Although physical delivery is often preferred, financial settlement is becoming more frequent as the markets mature.

Electricity futures are settled against the average day-ahead price of the delivery period. This fulfils the hedging needs of markets participants, but simultaneously raises the question of convergence of the futures to spot the spot price, discussed later in section 3.2.

Term structure of electricity futures prices sets them apart from those of storable commodities. Since electricity is not storable and the spot market is the day-ahead market with an hourly time grid, a power operator who wants to hedge its generation for the next year needs to obtain contracts for all of the hours in the year.

For a non-leap year, this amounts to a whopping number of 8,760 hourly contracts. Because this would result in a huge dispersion with the market participants spreading their needs across all of those contracts, it is much more convenient to aggregate hours during pre-defined delivery periods. The delivery period specifies all of the hours during which the electricity should be delivered and it refers to a month, a quarter, a year, or even a week or a day. The contract also precisely specifies the schedule of the delivery: base-load if the electricity is to be delivered during all of the hours of the period and, peak-load or off-peak hours if the electricity is to be delivered only during a special period which reflects large or low demand.

The result of this aggregation mechanism is the sparse structure of the electricity forward curve. The further the maturity, the longer the delivery period.

Some organised electricity markets such as EEX and NordPool also quote prices of vanilla options on futures contracts.

2.1.3 Balancing and reserve market

While forwards and futures market share many similarities across different regions, the balancing and reserve markets are much more dependent on national regulation which defines the exact function of the TSO.

In Europe, the European Network of Transmission System Operators for Electricity (ENTSO-E) is an association of 41 TSOs from 34 countries coordinating overarching grid topics. One of the key functions of a TSO is to ensure a constant power frequency in the transmission system. A change in frequency indicates to the TSO a shortage or a surplus of energy in the system. In physical terms, this means that there is a deceleration or acceleration of turbines since it is their kinetic energy that balances consumption and generation. For such a case, a clear set of rules and measures for system stabilization have to exist.

Since even 15 minutes of disequilibrium may result in a dramatic blackout, a strict minute by minute control of the equilibrium between consumption and generation is necessary. In order to cope with possible instabilities in the electricity system, the responsible TSO must have at its disposal *operating reserves*, generation capacities that can be mobilised within a short time-frame. The Operation Handook of the

ENTSO-E specifies control actions in the case of a disturbance. The frequency control actions are performed in different successive steps, each with different characteristics and qualities, and all depending on each other.

The products traded in the reserve market are constructed to support these control actions. Compared with the products in the forward, futures or spot market, the reserve market products are more much technical and may refer to specified power plants. These power plants must be capable of reducing or increasing production at short notice. Unlike most other electricity products, in which one only pays for the only delivered energy, reserve energy products often involve an additional payment for the availability of the reserved capacity.

The prices for balancing power are usually include the price for the delivered energy only. These often represent additional costs for the merchant. Prices for balancing power differ widely and are only sometimes related to spot or futures market prices.

2.1.4 Market coupling

Neighbouring local electricity markets are usually not completely disconnected but coupled via transmission capacities owned by the TSOs. An optimal usage of transmission capacities in the economical sense entails that in case of price differences between two local markets, one would transmit electricity from the higher-priced market A to the lower-priced market B. As a result, the generation demand in B would fall and generation demand in A would rise, pushing prices towards an equilibrium. Therefore, a price difference between connected markets should only be possible in case of congestions in transmission capacities. One possibility of fostering optimal usage of transmission capacities is through creation of a dedicated market for such capacities (e.g., via auctioning). Another, more convenient way is to integrate allocation of transmission capacities in a price-finding algorithm of two or more collaborating spot exchanges via an implicit auctioning. This is generally regarded as more efficient since optimal allocation is ensured by the exchanges and a separate auctioning of transmission capacities is no longer required. This results in a single price in all participating markets provided there is no congestion of capacities.

2.2 Electricity Exchanges

Energy exchanges are major marketplaces for electricity. In recent years more and more countries have founded exchanges for electricity, most of which are located in Europe and North America. While some of them are only a basic marketplace for spot products, the major exchanges feature a derivatives market with a high trading volume. The landscape of power exchanges in Europe has been changing rapidly over recent years.

In the following, we provide a brief overview of some of the biggest markets.

Nord Pool Spot

Nord Pool Spot is a multinational electricity exchange of Northern Europe. It started in 1993 under the name Nord Pool ASA (then Statnett Marked AS) as a Norwegian market for physical contracts following the deregulation of the Norwegian electricity market in 1991. Three years later, in 1996, together with Sweden the world's first multinational exchange for trade in power contracts, the joint Norwegian–Swedish power was started. Some time after, Finland and Denmark decided to join Nord Pool. A separate company focusing on short term trading, Nord Pool Spot, was established in 2002. In 2008 the derivatives trading business was bought by NASDAQ OMX Commodities. In 2010, a joint venture of Nord Pool Spot and NASDAQ OMX Commodities, a market for UK energy contracts known as N2EX was launched.

Today, Nord Pool Spot comprises the Nordic countries of Denmark, Finland, Norway and Sweden and the Baltic countries of Estonia, Latvia and Lithuania. More than 80% of the total consumption of electrical energy in the Nordic market is traded through Nord Pool Spot. It operates a day-ahead market Elspot and an intra-day market Elbas.

Elspot is based on an auction trade system, where the bids for purchase and sale of power contracts of one-hour duration cover all 24 hours of the next day. There is a noon deadline for participants to submit bids, right after which all buy and sell orders are gathered into an aggregate demand curve and an aggregate supply curve, respectively, for each power-delivery hour. Then one determines the spot price for each hour by the intersection of the aggregate supply and demand curves. This price is also known as the system price. Nord Pool is a multinational exchange, therefore possible grid congestions require a partition into separate bidding areas. Separate price areas can only occur if the contractual flow between bidding areas exceeds the capacity for spot contracts allocated by TSOs. In case of no such capacity constraints, the system price equals the spot price throughout the different bidding areas.

After the Elspot results get published, trading continues in the physical intra-day market Elbas. The Elbas market is based on hourly contracts and provides continuous power trading 24 hours a day, up to one hour prior to delivery.

NASDAQ OMX Commodities Europe

NASDAQ OMX Commodities Europe started in 1993, as the former Nord Pool AS established a forward market in Norway with physical delivery. In 2008, NASDAQ OMX Commodities Europe bought Nord Pool's financial derivatives business. This market consists of futures, forwards, options and contracts for differences (CfDs).

Futures contracts traded are standardised day and week contracts, where weeks are listed in a continuous rolling cycle of 6 weeks. The settlement of futures contracts involves a daily mark-to-market settlement and a final spot (system price) reference cash settlement after the contract reaches its due date.

Forward contracts are offered for the delivery periods of month, quarter and year. Months are listed in a continuous rolling cycle of 6 months. Years cascade into quarters, and quarters cascade into months. In this context the term forward contract also refers to an exchange-traded product. Unlike futures, the forward products offered do not have get settled prior to the due date. The mark-to-market value is accumulated as daily loss or profit but not realised throughout the trading period. Only during the delivery period the difference between the price when the contract was entered into and the spot reference price gets settled.

NASDAQ OMX Commodities Europe also offers CfDs that settle on the difference between the system price and the area price, to provide service for the market participants who rely on financial market derivatives to hedge spot market prices and thus remain exposed to the risk that the system price will differ from the actual area price of their spot purchases or sales. This way it is possible to obtain a perfect hedge using a combination of a forward contract and a CfD.

Options contracts on standard forwards as the underlying contract are also traded. Only European-style to be precise, meaning they can only be exercised at the exercise date. As the price of the underlying forward instrument moves, options with new strike prices are automatically generated.

In addition to products for the Nordic market, products for the German and Dutch market including CfDs to neighbouring countries are also offered. Further, UK power futures are which are settled against the N2EX day-ahead index are traded.

The NASDAQ OMX Commodities Clearinghouse provides a clearing service for contracts traded through the NASDAQ OMX Commodities Europe exchange and those traded OTC and registered for clearing. To be accepted for clearing, a bilateral market electricity contract must conform to the standardised products traded at the exchange. This clearinghouse guarantees the settlement of all cleared financial and physical derivative contracts.

N2EX

N2EX was established in 2010 as a joint venture of Nord Pool Spot and NASDAQ OMX Commodities Europe. It offers a day-ahead auction market for the UK and a continuously traded spot and prompt market. The prompt market covers the period 48 hours out up to 7 days out, afterwards the products are transferred to the spot market. The day-ahead prices are used as underlying for UK futures contracts offered by NASDAQ OMX Commodities Europe.

European Energy Exchange (EEX)

The EEX is one of the leading exchanges for electricity and gas in Central Europe. It operates market platforms for trading in electric energy, natural gas, CO2 emission allowances and coal. In 2013, the trading volume of electricity on the EEX Power Derivatives Market amounted to 1,263.9 TWh (terawatt hours) compared with 931.4 TWh in 2012. It is situated in Leipzig, Germany. The current EEX emerged as a result of a merger between LPX Leipzig Power Exchange and the Frankfurt-based EEX in 2002. In 2008, the spot market operated by EEX was transferred into EPEX, a joint venture with the French exchange Powernext.

The EEX Power Derivatives GmbH offers futures for German and French power with weekly, monthly, quarterly and yearly delivery periods. Additional products are traded for Germany: single days and weekends are available. The day-ahead auction results from the EPEX spot market serves as the underlying of the financially settled futures. Futures are settled using a daily mark-to-market approach. Yearly and quarterly futures are cascaded. At the end of a month the last payment for monthly futures is established on the basis of the difference between the final settlement price and the settlement price of the previous exchange trading day. The final settlement price is established from the average of the associated EPEX spot market prices.

European-style options for German electricity, where the financially settled futures serve as the underlying, are also traded.

Its subsidiary European Commodity Clearing AG (ECC) offers a well regarded clearing service for OTC trades. OTC transactions corresponding to available products at the EEX or other partner exchanges can be registered by means of a so-called EFP trade (exchange futures for physical) for OTC clearing.

EPEX

EEX was founded as a joint venture of EEX and the French energy exchange Powernext. It is based in Paris and operates the electricity spot market for Germany, Austria, France, Switzerland and Luxembourg. In 2014 the volume of traded electricity on the markets of EPEX SPOT amounted to 382 terawatt hours (TWh), an increase of 10% over the previous year.

EPEX offers an auction based day-ahead and a continuous intra-day market. Products are individual hours, baseload, peakload and other blocks of hours. Market coupling contracts for deliveries between two market areas are also traded in the day-ahead market. The intra-day market is open 24 hours a day, 7 days a week and products can be traded until 45 minutes before delivery. For Germany, short 15-minute periods are offered in addition to hourly periods. EPEX publishes the European Electricity Index PHELIX, which represents a price level that would result in a physically unconstrained market environment on the territory of Germany, Austria and Luxembourg.

APX

The roots of APX go back to 1999, when Amsterdam Power Exchange was founded. In the following years, APX expanded into the UK market under the name APX-UK (now APX Power UK). Then in 2008, APX bought the strongly positioned Dutch and Belgian derivatives exchange for power and gas called ENDEX. The joint company, renamed APXENDEX, split again in 2013 into a power spot exchange

APX covering the Netherlands, Belgium and the UK and a derivatives exchange ICE ENDEX covering gas spot and power/gas derivatives.

APX has comprises several segments. APX Power NL is the day-ahead auction based and continuous intra-day market for Dutch electricity. APX Power UK is the Day-ahead market (hourly), continuous spot market (half-hourly) and prompt market (block products up to 4 weeks out). Finally, Belpex is the day-ahead auction market and continuous intraday market for hourly periods.

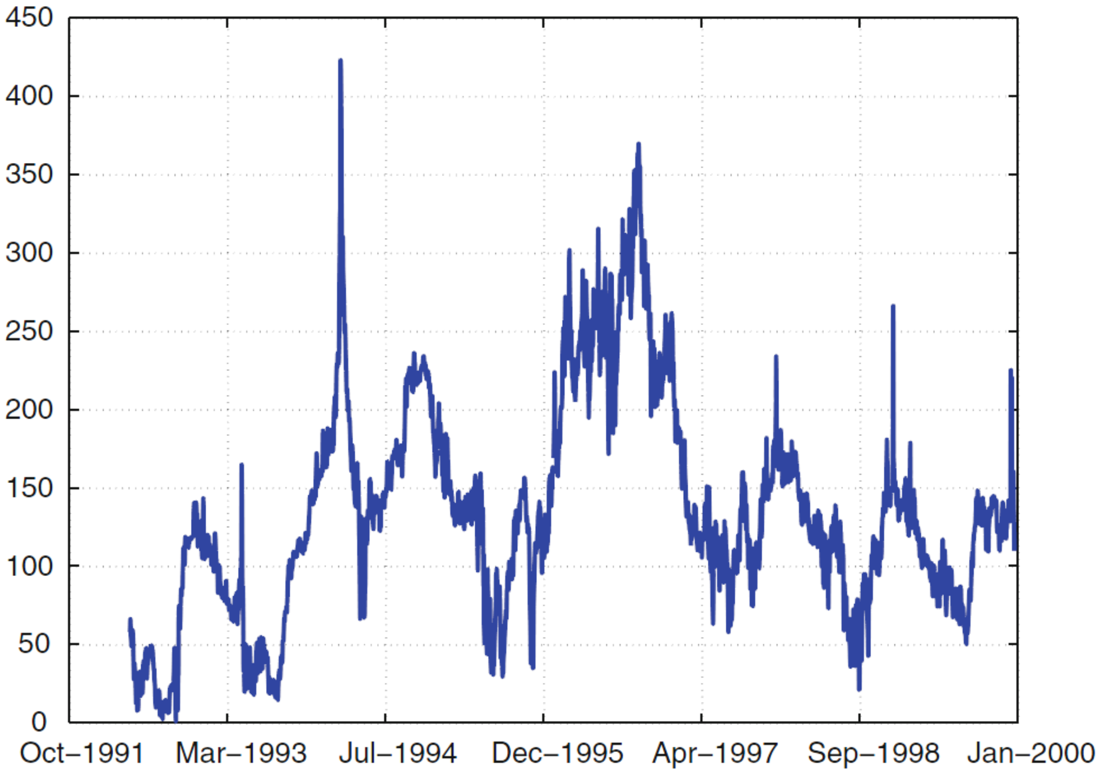


Figure 2.6: Historical daily spot price in NordPool market. Source:NordPool.

Chapter 3

An overview of price models

In this chapter we aim to provide an brief overview of the different approaches to electricity pricing and the models they lead to. The chapter is heavily based on the work of the French electricity expert Réne Aïd [6], whose book is an excellent reference on the current developments in electricity modelling. Many of the methods and approaches used in modelling of other financial markets have found their way into the modelling of electricity.

In the case of electricity, there are three different common approaches to modelling that give rise to different model classes.

1. The first class of models follows the Heath-Jarrow-Morton (HJM) [7] approach and attempts to model the observed forward prices. If needed, the spot price is derived as a future with zero maturity limit. This method is inherited from the modelling of interest rate yield curves. Therefore, many related results are not particular to electricity forward curve modelling and some additional constraints have to be considered since they were initially developed without a delivery period in mind.
2. The second class of models takes the opposite approach and starts by modelling the electricity spot price. The futures prices can then be deduced as an expectation under a convenient probability measure.
3. The third class comprises *structural models*. These models deduce the spot price using an over-simplified version of an economical equilibrium model between the production cost curve and electricity demand. The methodology here is somewhat electricity specific and differs from the usual approaches used in mathematical finance. However, since for electricity, in contrast to other commodities and financial instruments, a lot of helpful data is publicly available, these models are especially interesting.

For each of the above model classes, after discussing some preliminaries in section 3.2, in sections 3.3, 3.4 and 3.6 we will briefly discuss the underlying principles and some examples in order to illustrate their where they succeed and where they fail. A (detailed) comparison of different electricity models is a research topic on its own that exceeds the scope of this thesis, and the aforementioned work by Aïd does a great job of presenting a whole variety of models. Finally, in section 3.7 we discuss the limitations of financial models driven solely by Brownian motion components, and how these disadvantages are accentuated in the case of electricity. But first of all, in section 3.1, we start by recalling the Black 76 model, mainly due to its importance as an option quoting tool.

Note that on the modelling level and in the following, we do not distinguish between forwards and futures, and often use the two terms synonymously and interchangeably. Also, in all of the model examples, we try to stick to the notation used in the original papers, since it allows for easier cross-comparison in case of further reading.

3.1 Black 76 model

Ever since the publication of the original Black-Scholes model that focused on stock options, simple variations on this concept have found their way into other applications, and pricing futures is no exception.

The risk-neutral dynamic of the price of a future at time t with delivery at some time $T' > t$ in Black's model is given by a simple diffusion

$$dF_t = \sigma dW_t, \tag{3.1}$$

where $\sigma > 0$ is a constant volatility parameter and W_t a standard Brownian motion under the risk-neutral measure.

In Black's model, the value of a European call option with maturity T at time t on a future F_t with delivery date $T' > T$ equals:

$$C(F, K, \tau, \sigma) = e^{-r\tau}[FN(d_1) - KN(d_2)] \tag{3.2}$$

where F is the price of the future at time t, N the cdf of the standard normal distribution, $\tau = T - t$ and

$$d_1 = \frac{\ln(\frac{F}{K}) + \frac{\sigma^2}{2}\tau}{\sigma\sqrt{\tau}}, \quad d_2 = d_1 - \sigma\sqrt{\tau} = \frac{\ln(\frac{F}{K}) - \frac{\sigma^2}{2}\tau}{\sigma\sqrt{\tau}}. \tag{3.3}$$

The value of a corresponding European put is

$$P(F, K, \tau, \sigma) = e^{-r\tau}[KN(-d_2) - FN(-d_1)]. \tag{3.4}$$

Since the value of a call option is a strictly increasing function of the volatility parameter

$$C : (0, \infty) \to \left((e^{-r\tau}[F - K])^+, e^{-r\tau}F\right), \tag{3.5}$$

for any observed market price within this range, one can invert the function and find the volatility parameter s.t. the observed price of the option matches the market price, i.e.

$$\exists! \ \Sigma_t(T, K) > 0, \ \ C(F_t, K, \tau, \Sigma_t(T, K)) = C_t^*(T, K). \tag{3.6}$$

This number is called the *implied volatility.* It is sometimes referred to as "the wrong number which, plugged into the wrong formula, gives the right answer".

For a fixed date t, the function

$$\Sigma_t : (T, K) \to \Sigma_t(T, K) \tag{3.7}$$

is called the *implied volatility surface* at date t.

Today, options are usually quoted in terms of their implied volatilities and not their actual prices. This method has several advantages, for instance, the quote becomes a dimensionless quantity independent of the type of the underlying plus it allows for simpler comparisons of different models.

Although the Black's model predicts a constant implied volatility surface $\Sigma_t(T, K) = \sigma$, it is a well known empirical fact that $\Sigma_t(T, K)$ is not constant as a function of strike nor time. Of course that none of the market participants actually believe in the hypotheses of the Black's model, they just agree to use it as a tool for translating the market price into a representation in terms of the implied volatility.

3.2 Relation of the spot and forward price

Before we continue with the study of the different model classes, we discuss some important preliminaries. One of the most important relations in modern pricing theory is the relation between the spot and the forward price. There are multiple ways to establish a relation between the two, and we study them in the following subsections.

3.2.1 Cost of carry formula

For freely storable assets, it is well known that the *cost of carry* formula holds:

$$F(t,T) = e^{r(T-t)} S_t, \tag{3.8}$$

where S_t is the spot price at time t, $F(t,T)$ is the price of a forward at time t with delivery at time T, and r a constant interest rate. This relation is easily derived using a no-arbitrage argument. However, the cost of storage for commodities is not negligible (e.g. cereals, copper, oil), and the formula has been modified to account for storage cost:

$$F(t,T) = e^{r(T-t)+c(T-t)} S_t, \tag{3.9}$$

where $c(T-t)$ is the cost of storage as a function of time to maturity. Even with this correction, the relation still did not hold in some cases. Sometimes the forward price could be lower than the spot. An attempt at a further improvement of the formula was made with the introduction of the *convenience yield*, a function representing the extra return that the actors would gain from holding the commodity instead of a forward contract:

$$F(t,T) = e^{r(T-t)+c(T-t)-y(t,T)} S_t, \tag{3.10}$$

with $y(t,T)$ as the convenience yield. However, since the convenience yield is not observable, the formula is also not an ideal representation.

In the case of electricity, the cost of carry formula breaks. Due to its non-storability, any storage or convenience yield argument in explaining the relation between the current spot and forward price does not hold. All of the relations above are discussed in more detail in [8].

3.2.2 Convergence relation

Another important hypothesis in pricing theory is the *convergence* of forward prices towards spot prices:

$$\lim_{t \to T} F(t,T) = S_T. \tag{3.11}$$

In other words, this relation entails that the basis risk between the forward contract and the spot market is negligible. Provided that this relation holds, a market participant can easily reverse his position on the forward market by buying or selling on the spot market, while remaining sure that the losses and gains from the different markets will even out.

It is not immediately clear whether relation (3.11) holds in case of electricity. Recall from chapter 2 that forward contracts involve a delivery period that leads the operators to implement a cascading mechanism of switching to contracts with shorter delivery periods as soon as they become available. However, it is possible to test if this convergence relation holds for electricity futures, e.g. day-ahead futures contracts. The underlying of day-ahead futures is the average spot price on the day of delivery. The testing of this point was undertaken in [9]. The author shows that even for weekend days the difference between the last quoted day-ahead future and the realised spot is statistically not different from zero. For some particular hours (e.g. peak hours) the difference gets close to 5%. Therefore, one can assume that there is convergence up to a small premium.

3.2.3 Spot-forward risk neutral relation

A third important spot-forward relation assumes that the forward price at time t is equal to the expectation w.r.t. the risk neutral measure of the spot price given available information at time t:

$$F(t,T) = \mathbb{E}^{\mathbb{Q}} \left[S_t \mid \mathcal{F}_t \right], \tag{3.12}$$

where $\mathbb{Q}$ is the risk neutral measure and $\mathcal{F}_t$ the information at time t. For pricing purposes, it helps if the risk-neutral measure is unique. If not, the main criterion for the choice of $\mathbb{Q}$ is the ability to fit the observed market prices.

3.2.4 Risk premium

Finally, an often discussed quantity in the literature is the *risk premium*, defined as:

$$R(t,T) = F(t,T) - \mathbb{E}^{\mathbb{P}}\left[S_t \mid \mathcal{F}_t\right], \tag{3.13}$$

where $\mathbb{P}$ is the physical measure. The risk premium is then given an economical interpretation as an indicator of relative market power or the relative risk aversions of the electricity producers and consumers. However, despite being an interesting economic indicator, it is not of great relevance in the scope of derivatives pricing.

3.3 HJM-style models

Within the HJM framework the dynamic of the forward price $F(t,T)$ at time t with delivery date $T > t$ in SDE form is given by:

$$\frac{dF}{F}(t,T) = \sum_{i=1}^{n} \sigma_i(t,T) dW_t^i, \tag{3.14}$$

which admits the following solution:

$$F(t,T) = F(0,T) \exp\left(-\frac{1}{2} \int_0^t \sum_{i=1}^{n} \sigma_i^2(s,T) ds + \int_0^t \sum_{i=1}^{n} \sigma_i^2(s,T) dW_s^i \right), \tag{3.15}$$

where $\sigma_i(t,T)$ are n volatility functions, and the W^i are n Brownian motions, possibly correlated. The initial condition $F(0,T)$ is given by the observed forward prices.

First of all, one needs to pin down what the delivery date T actually represents. Since electricity is delivered during a period of time, in this form one can only model one set of futures, e.g. monthly, quarterly, yearly etc.. T would then denote the starting day of delivery. For instance, when modelling monthly futures, T would represent the first day of the delivery month and $F(t,T)$ the forward price of a monthly future with delivery over the month beginning at date T.

We are free to choose the number n of Brownian motions to our liking. The optimal choice of n in order to capture the dynamics of electricity forwards has been a subject of multiple studies based on the principal component analysis (PCA). The analyses by Frestad et al. [10, 11, 12] and Koekebakker et al. [13] suggest that the electricity forward curve is much more volatile than any other market. In [13], the authors use a modelling approach close to the standard HJM, PCA is performed on NordPool electricity data (September 1995 - March 2001). The results suggest that more than seven factors are needed in order to get capture more than 90% of the variance, which is much higher compared to other markets. One factor accounts for 68% of the variance, two factors for 75%, three factors for 80% and four factors for 83%. For the sake of comparison, the PCA done in [10] shows that three factors were enough to capture over 90% of variance in the copper, crude oil and bond market.

There are a few drivers resulting in the increased complexity of the electricity forward curve compared to other markets. First of all, electricity forward prices exhibit a strong seasonality pattern. However, this seasonality does not pose a big modelling issue, since it is caused by the well known seasonality in consumption. Another argument trying to explain the need for an increased number of factors are the unrelated dynamics of the short term (e.g. weekly) contracts with the longer maturity contracts (e.g. yearly). For example, next week's electricity prices have little to do with the next year's electricity prices, and therefore require separate factors. Finally, note that it has been shown that returns of contracts with even very long maturities may not follow a normal distribution.

Exploiting the relation 3.12 and setting $S_t = F(t,t)$ for the spot price at time t and applying the Ito formula reveals the following dynamics of the spot price:

$$\frac{dS_t}{S_t} = \left(\partial_2 \ln F(0,T) - \sum_{i=1}^{n} \left[\int_0^t \sigma_i(s,t)\partial_2\sigma_i(s,t)ds + \int_0^t \partial_2\sigma_i(s,t)dW_s^i \right] \right) dt + \sum_{i=1}^{n} \sigma_i(t,t)dW_t^i. \quad (3.16)$$

Proof. We quickly outline the steps that lead to the (3.16).

Let

$$X_t = \int_0^t \sigma(s,t)dW_s$$

$$S_t = g(t, X_t) = \exp\left[\ln f(0,t) - \frac{1}{2}\int_0^t \sigma^2(s,t)ds + X_t\right].$$

Thus

$$dS = \partial_1 g \cdot dt + \partial_2 g \cdot dX + \frac{1}{2}\partial_2^2 g \cdot \langle dX_t, dX_t \rangle .$$

Moreover,

$$\partial_1 g = g \times \left[\partial_2 \ln f(0,t) - \int_0^t \sigma(s,t)\partial_2\sigma(s,t)dt - \frac{1}{2}\sigma^2(t,t)\right]$$

It holds that $\partial_2 g = \partial_2^2 = g$. And, one has

$$dX_t = \left[\int_0^t \partial_2\sigma(s,t)dW_s\right] dt + \sigma(t,t)dW_t$$

and

$$\langle dX_t, dX_t \rangle = \sigma^2(t,t)dt.$$

Thus, the term $\frac{1}{2}g\sigma^2(t,t)$ from $\partial_1 g$ cancels out with the term $+\frac{1}{2}g\sigma^2(t,t)$ coming from $\frac{1}{2}\partial_2^2 g \cdot \langle dX_t, dX_t \rangle$. ☐

3.3.1 Single-factor model

The simplest example of a HJM-style model is the single factor model with $\sigma(t,T) = \sigma_0 e^{-a(T-t)}$. By using the fact that:

$$\int_0^t \sigma(s,t)dW_s = \ln F(t,t) - \ln F(t,0) + \frac{1}{2}\int_0^t \sigma^2(s,t)ds \quad (3.17)$$

in combination with (3.16) we can see that this particular choice of the volatility function results in a mean-reverting dynamic of the spot price:

$$\frac{dS_t}{S_t} = (\mu(t) - a\ln(S_t))\, dt + \sigma_0 dW_t, \quad (3.18)$$

with

$$\mu(t) = \partial_2 \ln F(0,t) + a \ln f(0,t) + \frac{\sigma_0^2}{4}(1 - e^{-2at}). \quad (3.19)$$

Note that, in case we wish to preserve the Markov property of the spot price, not all volatility functions are admissible. In particular, if we assume that the volatility is only dependent of the time to maturity $T-t$, the only possible volatility functions preserving the Markov property are of the form $\sigma_0 e^{-a(T-t)}$ with σ_0 and a constant. These are also the most commonly used in literature and in practice. Without the Markov property, computation of price derivatives becomes significantly more complicated since the whole past of the Brownian motions has to be remembered.

3.3.2 Modeling multiple granularities

If we wish to simultaneously model forward prices for any delivery period $[T_1, T_2]$, we can specify the dynamics of $F(t, T_1, T_2)$ by:

$$dF(t, T_1, T_2) = \Sigma(t, T_1, T_2) F(t, T_1, T_2) dW_t. \tag{3.20}$$

A slight variation on the HJM approach that allows for simultaneous modelling of forwards with different lengths of the delivery period starts with the introduction of the unobserved instantaneous forward rate, that is, the price at time t for delivery at the instant T, denoted by $f(t, T)$. Then the (observable) price of a futures contract $F(t, T_1, T_2)$ at time t involving the delivery continuously between T_1 and T_2 equals:

$$F(t, T_1, T_2) = \frac{1}{T_2 - T_1} \int_{T_1}^{T_2} f(t, s) ds. \tag{3.21}$$

Now, the idea is to model the non-observable instantaneous delivery forward price $f(t, T)$, analogously to the instantaneous forward rate in the HJM framework [7], whereas the initial conditions are given by $f(0, T)$:

$$df(t, T) = \sigma_f(t, T) dW_t. \tag{3.22}$$

However, in both approaches the choice of the volatility function is limited if we wish to preserve the Markov property of the spot. This is studied in [14]. The authors show that, for (3.20), the only suitable volatility functions depend only on time t, $\Sigma(t, T_1, T_2) = \Sigma(t)$. Consequently, the only log-normal volatility structure for the quoted forward contracts that satisfies the no-arbitrage condition that all overlapping contracts must satisfy is Black's model. On the other hand, for (3.22), the authors show that the volatility functions has to be independent of the maturity T, $\sigma(t, T) = \sigma(t)$, amongst which is the most popular choice is constant volatility, once again resulting in Black's model.

In essence, one is faced with the choice between the realism of the volatility term structure and the relation with the spot price.

3.3.3 Two-factor model

In [15], Kiesel at al. develop an HJM two-factor model that is often used in practice. The authors model the price of the monthly futures $F(t, T)$, where t is the current date and T is starting date of the delivery period (first day of the month). They focus on monthly contracts and treat all other contracts as a portfolio of monthly contracts. The forward prices are then modelled directly under the risk-neutral measure, and the initial conditions are given by the observed forward curve. For the sake of simplicity, they use the matching moments approximation to approximate the density of the yearly contract by a log-normal distribution. Finally, the following two-factor model gets calibrated:

$$dF(t, T) = e^{-\kappa(T-t)} \sigma_1 F(t, T) dW_t^1 + \sigma_2 F(t, T) dW_t^2, \tag{3.23}$$

where W^1, W^2 are uncorrelated Brownian mottions. The variance for a given month then equals:

$$\mathrm{Var}\left[\ln F(t, T)\right] = \frac{\sigma_1}{2\kappa} \left(e^{-2\kappa(T-t)} - e^{-2\kappa T} \right) + \sigma_2^2 t. \tag{3.24}$$

The parameters of the model are then estimated by finding a least squares fit to the observed at-the money (ATM) option prices.

3.3.4 Joint modelling

Electricity markets exhibit significant correlation with other fuel markets. Therefore, it is natural to consider a joint model of electricity forward prices and fuel prices. For example, a basic joint coal-power model of this kind takes the following form:

$$\frac{dF^e}{F^e}(t,T) = \sigma^e(t,T)dW_t^e \tag{3.25}$$

$$\frac{dF^c}{F^c}(t,T) = \sigma^c(t,T)dW_t^c \tag{3.26}$$

where $F^e(t,T)$, $F^c(t,T)$ are forward prices of power and coal. The dependence is then introduced by correlating the Brownian motions W^e and W^c.

However, since it is highly unlikely for a shock in the electricity price to transfer to the forward prices of coal (it is usually the other way around), the symmetric model is a little unrealistic. To alleviate this problem, models relying on co-integration of variables have been developed, that attempt to capture the phenomenon of the levels of prices of different commodities moving together. More on this topic can be found in [16, 17, 18].

3.3.5 Conclusion

The main advantage of HJM-style forward models is their tractability and the fact that many results carry over from interest rate theory. Furthermore, options prices, for vanilla and other instruments, admit closed form solutions and can be computed analytically, which is very efficient. We've also seen that cross-market correlations can also be introduced in relatively simple way, although not without its drawbacks. However, HJM-style models model the forward price as a continuous process with log-normal returns, which is unrealistic. The log-normality assumption also results in the inability to fit the observed volatility curve. Also worth noting is that the HJM approach usually remains within the complete market setting.

3.4 Spot models

An ideal spot model captures the dynamic of the spot price while simultaneously preserving the Markov property of the spot which simplifies the calculation of the forward curve as an expectation of the spot under a suitable risk-neutral measure, according to (3.12). In this section we present a few spot-models and how they fare in representing the true behaviour of spot and forward prices.

3.4.1 Mean-reverting diffusion model

The one-factor model by Lucia and Schwartz, presented in [19], models the logarithm of the daily average spot price P_t under the physical measure $\mathbb{P}$ as:

$$\ln P_t = f(t) + Y_t, \tag{3.27}$$

by decomposing it into a deterministic function representing the seasonal part and a stochastic process Y_t with the following dynamics:

$$dY_t = -\kappa Y_t dt + \sigma dW_t^{\mathbb{P}} \text{ and } Y_0 = y_0, \tag{3.28}$$

where $W^{\mathbb{P}}$ is a standard Brownian motion under $\mathbb{P}$. So, Y_t is a mean-reverting stochastic process with a null long-run mean, constant reverting speed κ and starting value y_0. The distribution of P_t under $\mathbb{P}$ is well known to be normal with the following parameters:

$$\mathbb{E}^{\mathbb{P}}\left[P_t\right] = \exp\left(f(t) + (\ln P_0 - f(0))e^{-\kappa t} + \frac{\sigma^2}{4\kappa}(1 - e^{-2\kappa t})\right), \tag{3.29}$$

$$\mathrm{Var}^{\mathbb{P}}\left[P_t\right] = \mathbb{E}^{\mathbb{P}}\left[P_t\right]^2\left[\exp\left(\frac{\sigma^2}{2\kappa}(1 - e^{-2\kappa t})\right) - 1\right]. \tag{3.30}$$

Moreover, in order to establish a relation between the spot and forward prices, the authors assume that there is a unique risk-neutral measure $\mathbb{Q}$ under which the property (3.12) holds, i.e. we can express the futures price as the expectation of the spot:

$$F(t,T) = \mathbb{E}^{\mathbb{Q}}\left[P_T \mid \mathcal{F}_t\right]. \tag{3.31}$$

They further assume that the mean-reverting dynamics of Y are preserved under the risk-neutral measure $\mathbb{Q}$, and the difference is captured by the difference in the long-run mean:

$$dY_t = \kappa(\alpha^* - Y_t)dt + \sigma dW_t^{\mathbb{Q}} \text{ with } \alpha^* = -\frac{\lambda\sigma}{\kappa}. \tag{3.32}$$

The constant λ is known as the market price of risk.

In this setting, the price $F(0,T)$ of the future quoted at time 0 for delivery at the time T admits an analytical solution:

$$F(t,T) = \exp\left(f(T) + e^{-\kappa(T-t)}(\ln P_t - f(t)) + \alpha^*(1 - e^{-\kappa(T-t)}) + \frac{\sigma^2}{4\kappa}(1 - e^{-2\kappa(T-t)})\right). \tag{3.33}$$

Relying on these result the authors estimate and calibrate the model. The deterministic component of the spot, $f(t)$, is modelled as:

$$\begin{aligned} f(t) &= a + bD_t + c\cos\left((t+\tau)\frac{2\pi}{365}\right), \\ \text{with } D_t &= \begin{cases} 1, & \text{if } t \text{ is a holiday or a weekend,} \\ 0, & \text{else.} \end{cases} \end{aligned} \tag{3.34}$$

The estimation of the parameters $\kappa, \sigma, a, b, c, \tau$ is performed in a single non-linear least squares procedure. The authors perform an estimation of the model on the NordPool data ranging from January 1993 to December 1999 (seven years of daily data). This estimation leads to a significant non-zero mean-reversion coefficient $\kappa = 0.016 \times 365 = 5.84$ per year with a time step equal to one day $\Delta t = \frac{1}{365}$, and volatility $\sigma = 1.64$. The absolute percentage error, that measures the goodness of fit, is lower than 1%. However, although the model estimation matches the observations, the dynamic of simulated prices, illustrated in fig. 3.1 lacks the spicky behaviour of the observed NordPool daily spot price, depicted in fig. 2.6.

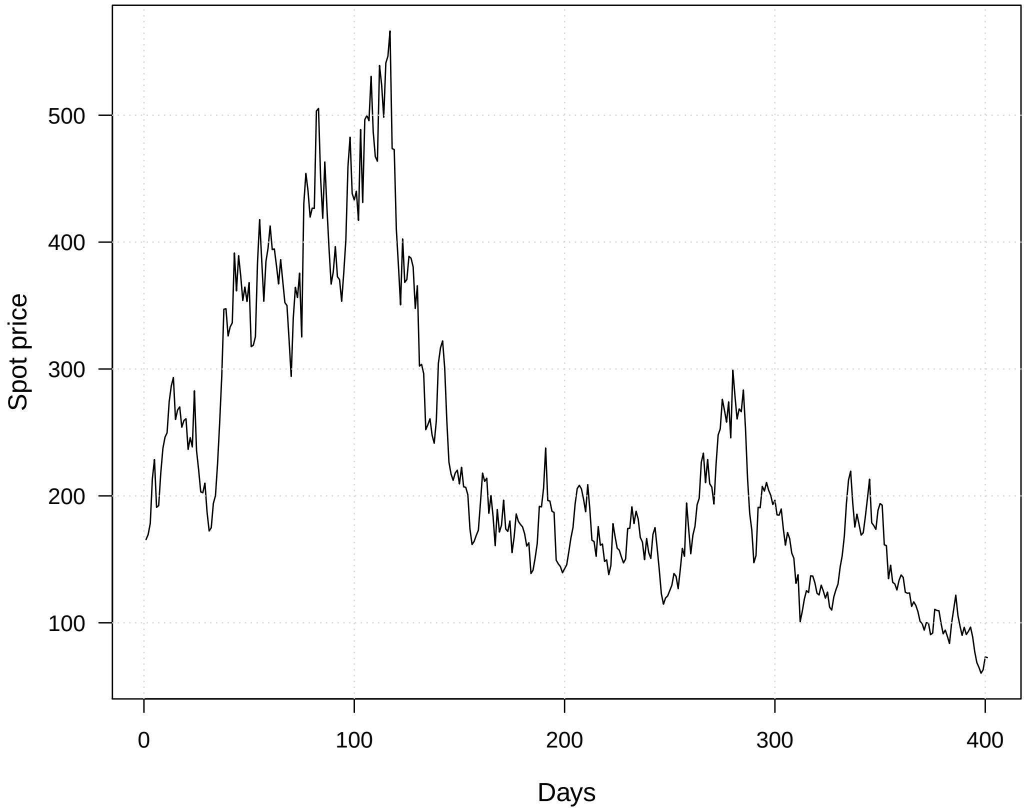


Figure 3.1: Sample daily spot path using Lucia and Schwartz's model. Realistic parameters according to the original paper were used: $\kappa = 5.84, \sigma = 1.64, a = 4.86, b = -0.09, c = 0.306, \tau = 0.836$.

Therefore, the calibration is performed using the observed forward prices instead. The theoretical forward price of a contract with delivery period $[T_1, T_2]$ is calculated by:

$$F(0, T_1, T_2) = \frac{1}{T_2 - T_1} \sum_{T_1 \leq T \leq T_2} F(0, T). \tag{3.35}$$

The authors perform the calibration on the observed forward prices over a year (NordPool data, end 1998 to end 1999). Using the above model with an out-of-sample parameter estimation and a null market price of risk ($\lambda = 0$), the authors arrive at a root mean square error (RMSE) over all of the sample contracts of around 9%, which is equivalent to about 11 NOK. So, the model does not allow for a perfect fit of the forward curve, possibly due to an insufficient number of factors.

The mean-reverting dynamic of the spot results in a dampening effect of the current-time dependent factors in the forward price dynamic, so that the prices of forwards (3.33) with long time to maturity quickly converge towards the deterministic seasonal component of the forward price, as visible in fig. 3.2.

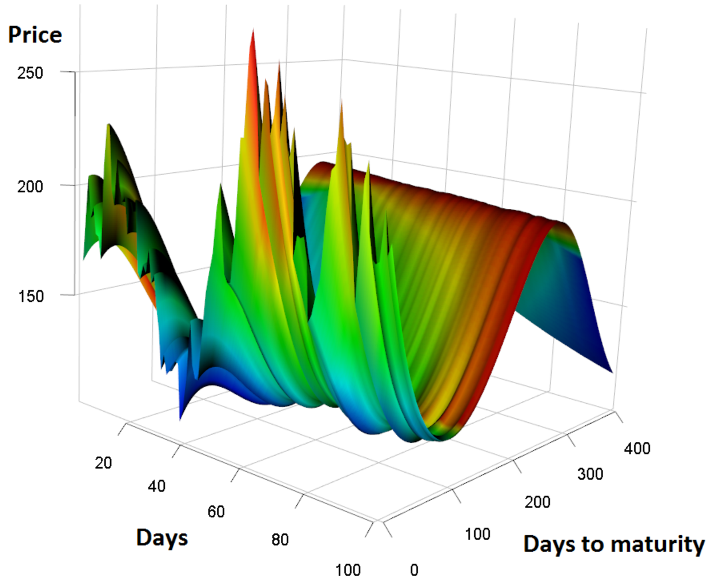


Figure 3.2: Simulation of the forward curve using Lucia and Schwartz”s model and same parameters as in fig. 3.1.

In conclusion, although it is possible to achieve a good fit of the historical spot prices using the model by Lucia and Schwartz [19], this model is limited in its ability to capture real price dynamics. The simulated spot prices lack some of the features of the observed spot, such as spikes. A perfect fit of the observed forward curve is also not achievable. This is partially due to the assumption of the constant price of risk, which can be improved with introduction of a seasonal market price. Finally, A one-factor model is simply not enough to efficiently model the forward curve, as already noted in section 3.3.

3.4.2 Mean-reverting jump-diffusion model

One of the simplest ways to improve the spot price dynamic of the previous model described in 3.4.1 is through addition of a jump component. In this subsection will discuss the model presented by Cartea and Figureoa in [20] that follows this approach.

The log spot price S_t dynamic under the physical measure is modelled as:

$$\ln S_t = g(t) + Y_t \tag{3.36}$$

where $g(t) \in C^1$ is a deterministic differentiable function modelling the seasonality of the log-price, and Y_t a stochastic process with the following dynamic:

$$dY_t = -\alpha Y_t \, dt + \sigma(t) \, dW_t + J \cdot dq_t \tag{3.37}$$

Here, α is the mean-reversion coefficient, $\sigma(t)$ the time dependent volatility, q_t a Poisson process with intensity l and J a random variable representing the size of the random jumps with $\ln(J) \sim \mathcal{N}(\mu_J, \sigma_J^2)$.

The dynamic of the spot price under the physical measure $\mathbb{P}$ is then given by:

$$dS_t = \alpha(\rho(t) - \ln\, S_t) S_t \, dt + \sigma(t) S_t \, dW_t + S_t(J-1) \, dq_t \tag{3.38}$$

with $\rho(t)$ equal to:

$$\rho(t) = \frac{1}{\alpha}\left(g'(t) + \frac{1}{2}\sigma^2(t)\right) + g(t). \tag{3.39}$$

The new jump-related term $S_t(J-1)dq_t$ in (3.38) comes from the fact that after a price jump, S_{t-} jumps to JS_{t-}, which makes $\Delta S_t = (J-1)S_{t-}$. Arguing that jumps should not lead to an extra return, the authors further assume that the expectation of J is equal to 1, i.e. $\mathbb{E}[J] = 1$, so the additional constraint $\mu_J = -\frac{\sigma_J^2}{2}$ follows.

Figure 3.3 illustrates a simulation of the daily average spot price based on this model. The spiky behaviour is now clearly present due to the jumps and the strong per annum mean reversion value $\alpha = 102$, equivalent to a half-life of two days. Although quick, this rate of return to normal price level is still too slow to capture an price change within an hour.

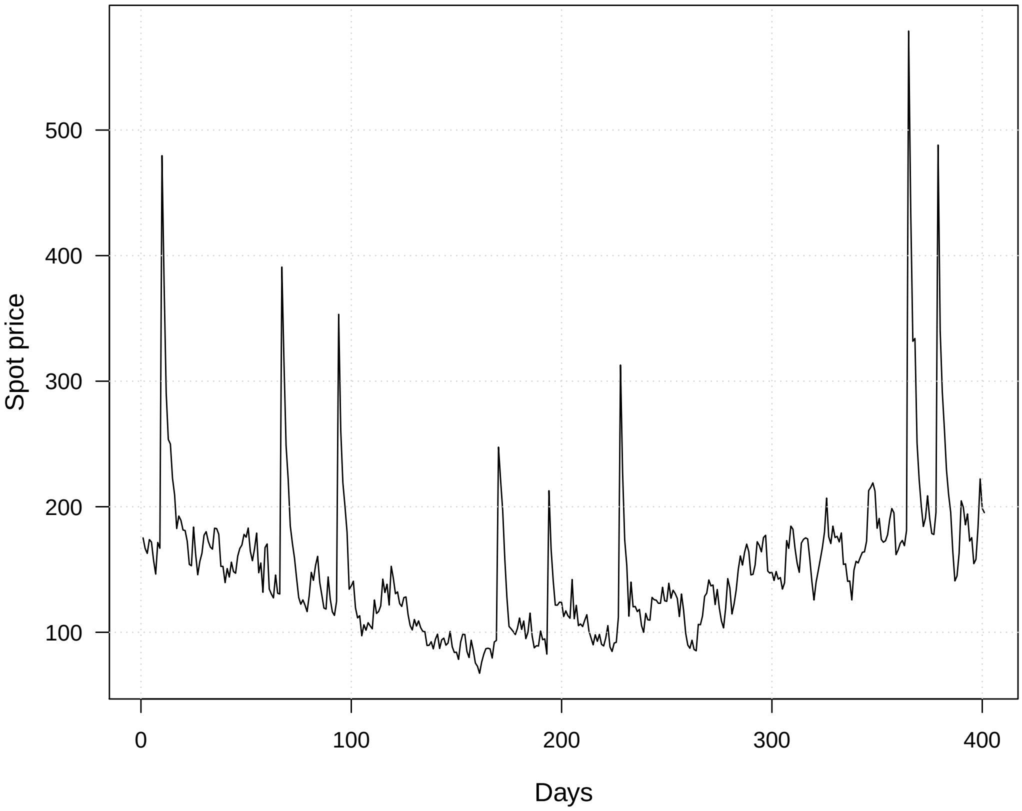


Figure 3.3: Sample daily spot path using Cartea and Figueroa's model. Realistic parameters according to the original paper were used: $\alpha = 102, \sigma_J = 0.67, l = 8.5, \sigma = 1.64$. For the seasonal part, (3.34) was used.

Once again, in order to establish relation (3.12) which would enable the calculation of the futures prices, the authors turn to the assumption of the constant market price of risk λ, analogous to Lucia and Schwartz in [19]. The log spot dynamic $x_t = \ln(S_t)$ under the risk-neutral measure $\mathbb{Q}$ is then given by:

$$dx_t = \alpha\left(\mu^*(t) - x_t\right)dt + \sigma_t dW_t^{\mathbb{Q}} + \ln\ J\,dq_t, \tag{3.40}$$

with $\mu^*(t) = \frac{1}{\alpha}g'(t) + g(t) - \lambda\frac{\sigma(t)}{\alpha}$ This setting allows for a closed form analytical solution of the futures prices. The price of a future at time t for delivery at the instant T is equals:

$$F(t,T) = G(T)\left(\frac{S_t}{G(t)}\right)^{e^{-\alpha(T-t)}} \mathscr{D}(t,T)\mathscr{J}(t,T) \tag{3.41}$$

where $G(t) = e^{g(t)}$ represents the deterministic seasonal part, $\mathscr{D}(t,T)$ is a term coming from the diffusion part

$$\mathscr{D}(t,T) = \exp\left(\int_t^T \left[\frac{1}{2}\sigma^2(s)e^{-2\alpha(T-s)} - \lambda\sigma(s)e^{-\alpha(T-s)}\right] ds\right), \tag{3.42}$$

and $\mathscr{J}(t,T)$ is the term coming from the jump part

$$\mathscr{J}(t,T) = \exp\left(l\int_t^T (\xi(T,s) - 1)ds\right), \tag{3.43}$$

where

$$\xi(T,s) = \exp\left(-\frac{\sigma_J^2}{2}\left(e^{-\alpha(T-s)} - e^{-2\alpha(T-s)}\right)\right). \tag{3.44}$$

Let us comment shortly on the forward prices. First of all, a perfect fit of the observed forward curve is not very likely due to the constant market price of risk. As already mentioned in the previous section, a possible improvement would include a seasonal market price of risk. Secondly, both the jump frequency represented by the jump intensity l and the jump size driven by σ_J have a negative impact on the forward price. Since those two parameters are intertwined, estimation of one of them may have an impact on the estimation of the other, therefore it is hard to assess a change in only one of them. A blind comparative static leads to a decrease in $F(t,T)$ that causes an increase in l because the term $\int_t^T (\xi(T,s) - 1)ds$ is negative. Furthermore, a jump of S at time t impacts the whole term structure. However, this is dampened extremely quickly due to the strong mean-reversion value of α. This same strong mean-reversion, although useful for erasing jump effects in the price, is also a disadvantage since it results in a flat dynamic of the futures prices with long maturities, so the forward curve basically equals the seasonal component. The authors therefore suggest far-lower values of the mean-reversion coefficient in order to achieve a realistic dynamic of long-term futures.

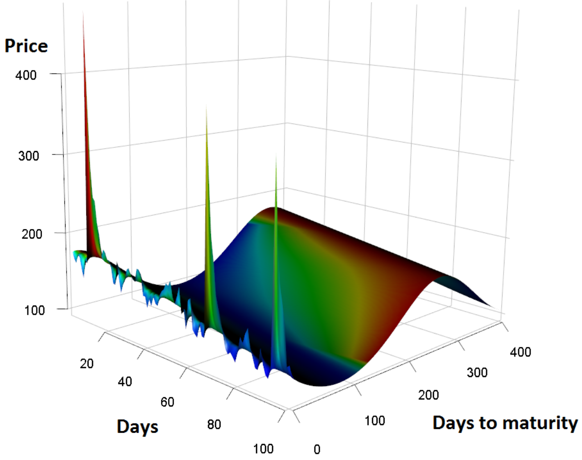


Figure 3.4: Simulation of the forward curve using Cartea and Figueroa's model and same parameters as in fig. 3.3

.

3.4.3 Multi-factor models

Although one-factor spot price models that we discussed succeed in reproducing the main features of the spot price such as seasonality, mean-reversion and spikes, they fail to reproduce the proper dynamic of the forward curve. We could try to improve the dynamic of those models by introducing volatility functions that depend not only on time but on the current spot price as well, leading to local volatility spot models. However, since the changes in the forward curve are, amongst others, driven by the information that has little to do with the spot price (e.g. planned outages of power plants, prices of fuel futures), it is unlikely that this would result in an improvement.

To overcome this, multi-factor spot models offer virtually limitless modelling possibilities. However as a trade-off, by having more than one factor we give up the Markov property of the spot. In this subsection we present as illustration the toy example discussed in [6].

The author models the spot price S_t at time t as:

$$S_t = \Lambda(t) e^{X_t + Y_t}, \tag{3.45}$$

where $\Lambda(t)$ is a deterministic function modelling the seasonal component of the price, X_t a mean-reverting process

$$dX_t = \lambda_X(\mu_X - X_t)dt + \sigma_X dW_t, \tag{3.46}$$

and Y_t a mean-reverting jump process

$$dY_t = -\lambda_Y Y_t dt + h dN_t. \tag{3.47}$$

with $\ln(h) \sim N(\mu_J, \sigma_J)$ and N_t a Poisson process with intensity l.

Having two price components in 3.45 allows for a separation of short term spikes given by Y and the trend given by X. The different mean-reversion speeds allow a separate quick dampening of the spikes caused by Y, without affecting the long end of the forward curve. This way, we get both the spikes in the spot price as well as the fluctuations in the forward curve dynamic due to the diffusion factor, as illustrated in [6].

Note that this toy example does not discuss the estimation of the different price components, i.e. the parameters of X_t and Y_t. The only observable quantity is the spot price process S_t whereas the estimation procedure should reconstruct the two processes X_t and Y_t that act as building blocks. The parameter estimation of a multi-factor spot model usually involves using some sort of filtering methods.

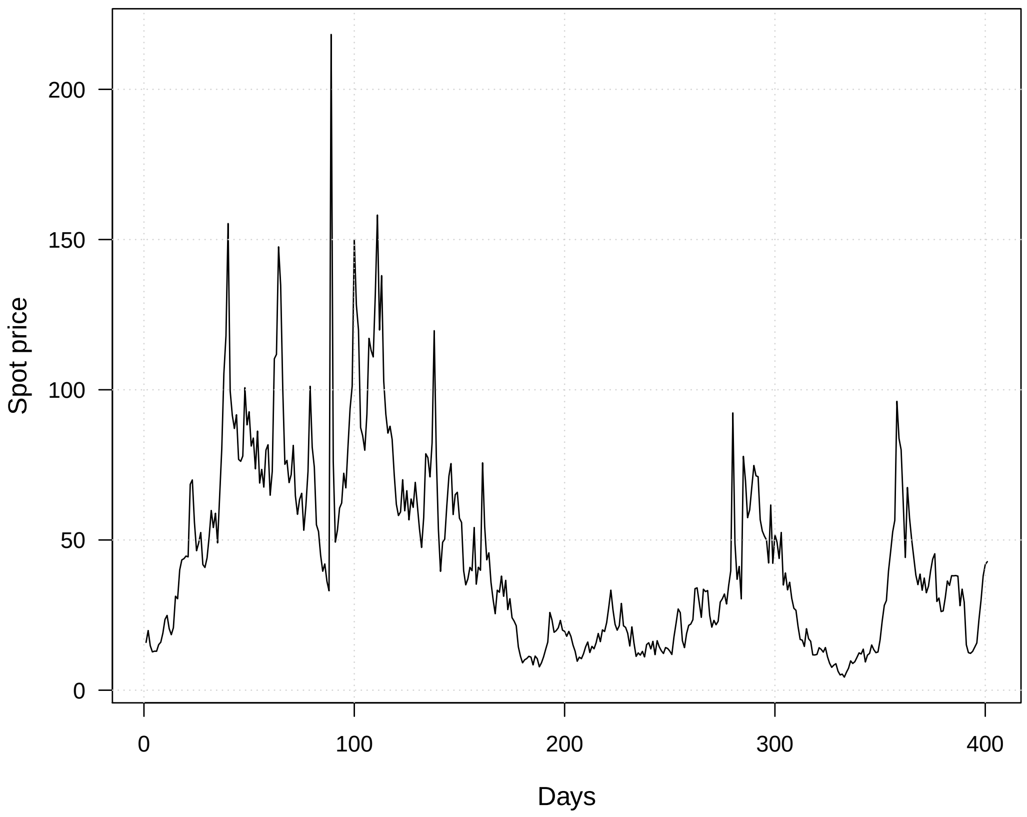


Figure 3.5: A sample path of the daily spot price using the model (3.45). Spikes are clearly observable.
.

3.5 Conclusion

Spot price models do a decent job at their purpose, that is, reproducing the spot price dynamic. For this, even single-factor jump models suffice. Generally, models with jumps are superior to continuous models for electricity spot modelling, due to its spiky nature.

We have seen that in the case of the single-factor mean-reverting model, even if we manage to capture the (spiky) spot dynamic, we do it at the expense of a flat forward curve due to high values of the mean-reversion factor. More than one factor is needed in order to reproduce the spot and forward dynamic correctly, possibly with the presence of jumps in the driving factors. The estimation of these multi-factor spot models usually involves filtering procedures.

Furthermore, for pricing forwards, one always first has to find a way to establish a relation between the physical and the risk-neutral measure. This is often done by assuming dynamics of the stochastic driving factors directly under the risk-neutral measure, adjusted by a market price of risk, which adds an additional layer of complexity. Spot models also lack the natural ability of forward models to perfectly fit the observed forward prices. These are all disadvantages making pricing options on forwards more difficult.

3.6 Structural models

We've already established that more than one factor is needed in order to properly capture spot and forward price dynamics. *Structural models* get their factors from the information available on the electric system. Large amounts of publicly accessible data are characteristic for electricity markets. For instance, a quick look at the NordPool website offers the access to: hourly historical consumption per geographic area, detailed production and planned outages, exchanges, levels of reservoirs and much more.

Structural models focus on the price formulation mechanism and dependencies between the observable factors. They attempt to deduce the spot price using a mathematical model of an economic equilibrium with stochastic driving factors. In this section we take a look at two models, the model from Barlow [21] as an example of a basic structural model, and the model by Cartea and Villaplana [22] as an example of a more advanced multi-factor structural model.

3.6.1 Barlow's model as a basic structural model

In the model presented in Barlow [21] the spot price S_t is determined by the equilibrium between an increasing supply function $u_t(x)$ and a decreasing demand function $d_t(x)$. More precisely, the spot price S_t at time t verifies

$$u_t(S_t) = d_t(S_t). \tag{3.48}$$

The author than assumes a constant supply function $u_t = g$ and an inelastic demand $d_t(x) = D_t$, where D_t denotes a stochastic process modelling representing the current electricity consumption. The spot price is then given by $S_t = g^{-1}(D_t)$. He further uses a basic non-linear form for $g(x) = a_0 - b_0 x^\alpha$ with $\alpha < 0$. If the demand exceeds the maximum capacity a_0, the spot price gets capped at A_0. Therefore, the spot price equals:

$$S_t = \begin{cases} \left(\frac{a_0 - D_t}{b_0}\right)^{\frac{1}{\alpha}} & D_t \leq a_0 - \epsilon_0 b_0 \\ \epsilon_0^{\frac{1}{\alpha}} & D_t \geq a_0 - \epsilon_0 b_0 \end{cases} \tag{3.49}$$

where ϵ_0 is determined by the level of the price cap $A_0 = \epsilon_0^{\frac{1}{\alpha}}$.

Note that the inverse demand function can be rewritten as $g^{-1}(D_t) \approx \frac{C}{a_0 - D_t}$ by considering that $\alpha \approx 1$ (check the paper [21]). From this form it follows that the price is inversely proportional to a quantity that can be interpreted as a residual capacity if a_0 is the maximum available capacity in the system.

The author then uses an Ornstein-Uhlenbeck process to model the demand D_t:

$$dD_t = -\lambda(D_t - a_1)dt + \sigma_1 dW_t, \tag{3.50}$$

which is the only source of randomness in the model.

After a row of transformation and algebraic simplifications the author expresses the spot price in the following form:

$$S_t = \begin{cases} (1 + \alpha X_t)^{\frac{1}{\alpha}} & 1 + \alpha X_t \leq \epsilon_0 \\ \epsilon_0^{\frac{1}{\alpha}} & 1 + \alpha X_t \geq \epsilon_0 \end{cases} \tag{3.51}$$

where $\epsilon_0^{\frac{1}{\alpha}} = A_0$ and X_t is another Ornstein-Uhlenbeck process

$$dX_t = -\lambda(X_t - a)dt + \sigma dW_t. \tag{3.52}$$

The parameters $\lambda, \alpha, a, \sigma$ then have to be estimated based on market data.

Figure 3.6 shows a sample path of the model. What instantly catches the eye is the striking spiky behaviour of the price, in spite of the continuity of the single driving stochastic process.

However, there are still problems left in this model. As pointed out in the litearure [23], the estimation leads to a negative value of $\alpha = -1.08$, which makes the supply curve a steep function near the breaking point given by $a_0 - \epsilon_0 b_0$. Because of this, the model hits the price cap fairly often. When a spike occurs, it is more likely to reach the cap. Improvements of the model were suggested in [24, 25], by addressing the shape of the supply curve.

Finally, this model, X_t being the only stochastic driver, is still just a one-factor model and therefore limited in its ability to capture real price dynamics effectively. We can easily predict that the forward curve would suffer from the typical drawbacks discussed earlier. This is confirmed by the forward curve analysis done in Aid [6], the result is expectedly flat.

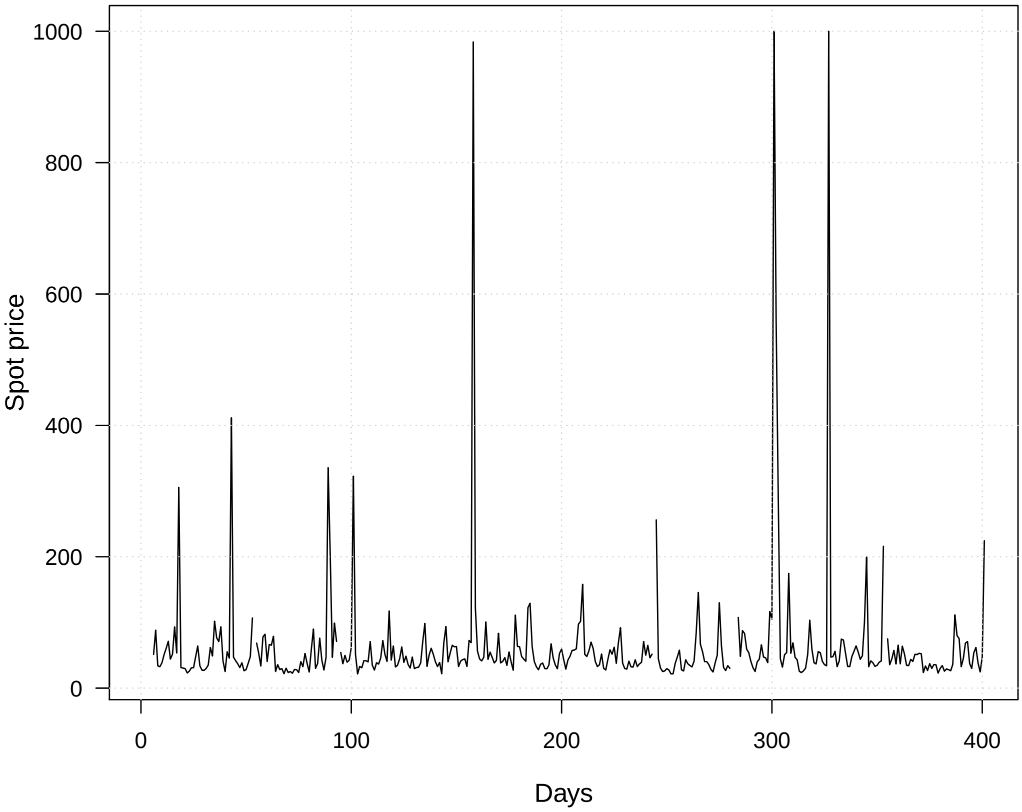


Figure 3.6: A sample daily spot path using Barlow's model. Realistic parameters according to the original paper were used: $\alpha = -1.08, \lambda = 172, a = 0.91, \sigma = 0.12$. The price is capped at 1000.

3.6.2 Cartea and Villaplana's two factor structural model

A more advanced multi-factor structural model is presented in Cartea and Villaplana [22]. In this model, an equilibrium function ϕ defines the spot price P_t at time t:

$$P_t = \phi(D_t, C_t), \tag{3.53}$$

where D_t are C_t stochastic processes modelling the electricity demand and the available capacity, respectively. The equilibrium function ϕ is increasing in the demand argument and decreasing in the capacity argument.

The authors model the demand and capacity dynamics are using a deterministic seasonal component and an Ornstein-Uhlenbeck process for the noise component under the physical measure $\mathbb{P}$:

$$D_t = g_t^D + X_t^D \qquad dX_t^D = -\kappa^D X_t^D dt + \sigma_t^D dW_t^{\mathbb{P},D} \tag{3.54}$$

$$C_t = g_t^C + X_t^C \qquad dX_t^C = -\kappa^C X_t^C Ct + \sigma_t^C dW_t^{\mathbb{P},C} \tag{3.55}$$

where $W_t^{\mathbb{P},D}$ and $W_t^{\mathbb{P},C}$ are independent standard $\mathbb{P}$-Brownian motions.

The authors choose the function ϕ to be of the form:

$$\phi(D_t, C_t) = \beta \exp(\gamma C_t + \alpha D_t) = P_t, \tag{3.56}$$

with $\alpha, \beta > 0$ but $\gamma < 0$.

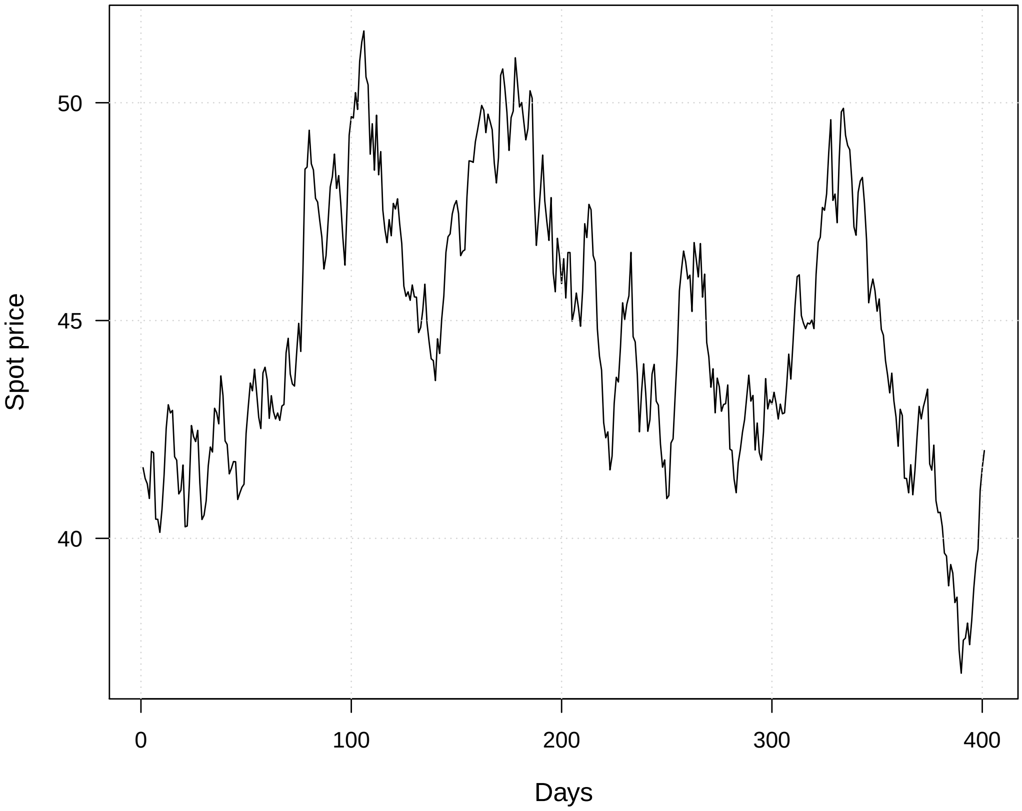


Figure 3.7: A sample path of the daily spot price using Cartea and Villaplana's model. Realistic parameters according to the original paper were used: $\kappa_D = 0.33, \sigma_D = 1580, \kappa_C = 0.37, \sigma_C = 2056, g_t^D = 3.5 \cdot 10^4 + 10^3 \cos\left(\frac{2\pi(t+7)}{364}\right), g_t^C = 3.7 \cdot 10^4 + 10^3 \cos\left(\frac{2\pi(t+7)}{364}\right)$.

In order to compute prices of futures contracts using relation (3.12), it is necessary to know the dynamics of spot price under the risk-neutral measure $\mathbb{Q}$. The authors incorporate the transition to the risk-neutral dynamic of X_t^D and X_t^C through the introduction of two time-dependent market prices of risk $\phi^D(t)$ and $\phi^C(t)$ for each unhedgeable factor D and C. The dynamics of X_t^D and X_t^C under $\mathbb{Q}$ then verify:

$$dX_t^D = -\kappa^D(X_t^D + \theta^D(t))dt + \sigma^D(t)dW_t^{\mathbb{Q},D}, \tag{3.57}$$

$$dX_t^C = -\kappa^C(X_t^C + \theta^C(t))dt + \sigma^C(t)dW_t^{\mathbb{Q},C}, \tag{3.58}$$

with

$$\theta^D(t) = \frac{\sigma^D(t)\phi^D(t)}{\kappa^D}, \qquad \theta^C(t) = \frac{\sigma^C(t)\phi^C(t)}{\kappa^C}.$$

Here $W_t^{\mathbb{Q},D}$ and $W_t^{\mathbb{Q},C}$ are independent standard $\mathbb{Q}$-Brownian motions. This setting allows for analytical computation of the conditional expectations of the spot price $\mathbb{E}^{\mathbb{Q}}\left[P_T \mid \mathcal{F}_t\right] = F(t,T)$, i.e. the futures price:

$$\begin{aligned} F(t,T) = \beta \exp \Bigg[& \gamma g^C(t) + \alpha g^D(t) + \\ & + \gamma \left(e^{-\kappa^C(T-t)} X_t^C + \kappa^C \int_t^T e^{-\kappa^C(T-s)} \theta^C(s) ds + \frac{1}{2} \int_t^T e^{-2\kappa^C(T-s)} \sigma_C^2(s) ds \right) \\ & + \alpha \left(e^{-\kappa^D(T-t)} X_t^D + \kappa^D \int_t^T e^{-\kappa^D(T-s)} \theta^D(s) ds + \frac{1}{2} \int_t^T e^{-2\kappa^D(T-s)} \sigma_D^2(s) ds \right) \Bigg]. \end{aligned} \tag{3.59}$$

Figures 3.7 and 3.8 illustrate the performance of this model. This time, due to the smoothness properties of the equilibrium function, there are no spikes present in the spot price. However, the forward curve dynamic does exhibit a floating behaviour for longer maturities.

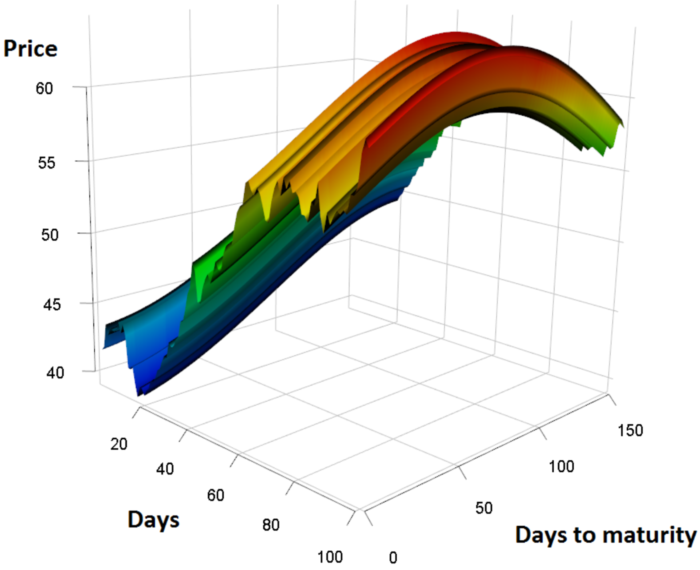


Figure 3.8: Simulation of the forward curve using Cartea and Villaplana's model and same parameters as in fig. 3.7

.

3.6.3 Conclusion

Structural models are still considered to be in their prime. The idea of modelling the economical equilibrium between the production cost curve and electricity demand leads to a very interesting class of models.

One-factor structural models suffer from same problems as one-factor models due to the insufficient number of factors: correct reproduction of both spot and forward dynamics. A statistical explanation of this phenomenon is partially offered by PCAs performed on electricity data. However, in case Barlow's model described in section 3.6.1, we are also able to offer a structural explanation: the assumption of constant demand is unrealistic. Another interesting result is the ability of Barlow's model to obtain spikes in spite of the continuity of the only underlying stochastic driver. On the other hand, multi-factor structural models are a powerful tools capable of reproducing both spot and forward behaviour fairly well. Forward pricing again requires an assumption on the relationship between the physical and the risk-neutral measure.

The main advantage of structural models is their ability to capture information and dependencies that spot and forward models may simply be oblivious of, in their hope of modelling it implicitly through one of their parameters.

3.7 Limitations of Brownian motion driven financial modelling

Since the beginning of mathematical finance, when Louis Bachelier attempted to model the prices of assets at the Paris Bourse in [26], Brownian motion has played a central role in financial modelling. It is the most studied and best understood stochastic processes in mathematics. Brownian motion was also the main stochastic driver in the Black-Scholes model that revolutionized option trading. It is also the building block of more general *diffusion models*. In this section, we take the opportunity to point out some disadvantages of pricing models that are driven solely by Brownian motions. We then argue how all of these disadvantages are easily addressed by using simple jump processes instead.

Two important properties of Brownian motion are its *continuity* and *scale invariance*, i.e., the sample paths of Brownian motion are continuous functions of time and the statistical properties of Brownian motion are the same at all time resolutions. In contrast, prices in financial markets often involve jumps (or even spikes in case of electricity) and exhibit significantly different behaviour across time scales (again especially true for electricity).

Geometric Brownian motion, given by

$$\frac{dX_t}{X_t} = \sigma\, dW_t + \left(\mu + \frac{\sigma^2}{2}\right) dt, \tag{3.60}$$

and used in the Black-Scholes framework is not the only continuous time model built on Brownian motion.

Local volatility models proposed by Dupire, Derman and Kani are non-linear Markov diffusions where instantaneous volatility is dependent on both time and price:

$$\frac{dX_t}{X_t} = \sigma(t, X_t)\, dW_t + \mu\, dt. \tag{3.61}$$

Stochastic volatility models model the price X_t as a component of a bivariate diffusion (X_t, σ_t) driven by a two-dimensional Brownian motion (W_t^1, W_t^2).

$$\begin{aligned} \frac{dX_t}{X_t} &= \sigma_t\, dW_t^1 + \mu\, dt \\ \sigma_t = f(Y_t) \quad dY_t &= \alpha_t\, dt + \gamma_t\, dW_t^2. \end{aligned} \tag{3.62}$$

These models have much more flexible statistical properties, but still share with Brownian motion the continuity property.

It is a well known empirical fact that log returns of financial time series (especially electricity) follow a heavy-tailed distribution. This is mainly due to presence of jumps in the price. Although it is impossible to capture this behaviour with a geometric Brownian motion, local volatility and stochastic volatility models are perfectly capable of reproducing heavy tails. However, in case of local volatility models, heavy tails are obtained at the price of highly varying (nonstationary) diffusion coefficients, whereas in the case of diffusion-based stochastic volatility models, unrealistically high values of "volatility of volatility" are needed.

Further issues with diffusion models arise in an attempt to fit the model to the observed market option prices, or rather their implied volatilities. Geometric Brownian motion's inability to do so was one of the main drivers in development of more general diffusion models.

Local volatility models, on the other hand, are able to fit practically any cross section of prices. However, they give rise to non-intuitive profiles of local volatility which, to this day, have not received an interpretation in terms of market dynamics. So, although local volatility models manage to provide a solution to the "calibration" problem, they fail to explain the structure of the volatility surface.

Diffusion-based stochastic volatility models are able to reproduce the implied volatility curve for a given maturity fairly well. But, they do not perform as well across different maturities, that is, they cannot yield a realistic term-structure of implied volatilities. Furthermore, in order to reproduce a skew in the volatility curve, a negative correlation between movements in price and movements in volatility is required. The is sometimes attributed to the so-called "leverage" effect, but that explanation is not exactly structural either.

A final piece of evidence highlighting the shortcomings discussed above comes from the short-term option market. The very *existence* of a market for short-term options proves that the market participants acknowledge the presence of jumps in the price. How else could the price move 10% up or down in a matter of days? Moreover, short-term options are traded at significant prices and may exhibit a significant skew. This feature is unattainable in diffusion-based stochastic volatility models, one would require ridiculously high values of "volatility of volatility". Local volatility models are able reproduce this behaviour, but only with a very high variability in the local volatility surface, which is difficult to use and interpret.

Regarding hedging, one-dimensional diffusion models (local volatility models) give rise to complete markets, whereas in diffusion-based stochastic models completeness can restored by adding a single option to the set of the available hedging instruments.

On the contrary, simple jump processes such as Lévy processes generically lead to highly variable returns with realistic tail behaviour without the need for extreme parameter values or introduction of additional unobservable random factors. They are also able to reproduce and explain a variety of volatility curve patterns. For instance, fear of negative jumps leads to a negative skew, fear of positive jumps to a positive skew, and fear of symmetric jumps to a smile in the volatility curve. Short-term option prices are fitted just as well, since jumps are an integral part of the model. The strongest argument for using discontinuous models is not a statistical one: it is the presence of jumps in the price! This qualitative difference has a huge positive impact. The continuity property simply neglects abrupt market movements and one has to think of various "workarounds" in order to make it work (e.g., high values of "volatility of volatility"). In case of jump models, the above addressed properties are generic to the model.

Chapter 4

COS pricing method

In this chapter we present the cosine-series expansion method that we use for efficient option pricing within our model. The method was originally developed by Fang and Oosterlee in [1].

The starting point for pricing European options using numerical integration techniques is the risk-neutral valuation formula:

$$v(x,t) = e^{-r\tau}\mathbb{E}^{\mathbb{Q}}\left[v(y,T) \mid x\right] = e^{-r\tau}\int_{\mathbb{R}} v(y,T)f(y \mid x)dy, \tag{4.1}$$

where v denotes the option price, t_0 the initial date, T the maturity, $\tau = T - t_0$ the time to maturity and x the underlying price at time t_0.

For a random variable X, its density f and its characteristic function ϕ are Fourier transforms of each other:

$$\phi(\omega) = \int_{\mathbb{R}} e^{ix\omega}f(x)dx, \tag{4.2}$$

$$f(x) = \frac{1}{2\pi}\int_{\mathbb{R}} e^{-i\omega x}\phi(\omega)d\omega. \tag{4.3}$$

4.1 Inverse Fourier Integral via Cosine Expansion

To start with, we look at a different method of solving the inverse Fourier integral in (4.3). Instead of reconstructing the integrand using its Fourier-cosine expansion, we approach the problem differently by reconstructing the whole integral from its Fourier-cosine expansion. For functions with finite support, it is known that the cosine series expansion usually results in an optimal approximation [27].

The cosine expansion of a function f supported on $[0,\pi]$ equals to:

$$f(\theta) = {\sum_{k=0}^{\infty}}' A_k \cdot cos(k\theta) \text{ with } A_k = \frac{2}{\pi}\int_0^{\pi} f(\theta)cos(k\theta)d\theta,$$

where $\sum'$ stands for a sum with the first term weighted by $\frac{1}{2}$. For a function with finite support on any other interval $[a,b] \in \mathbb{R}$, one can use a simple transformation of variables to bring it in above form:

$$\theta := \frac{x-a}{b-a}\pi, \quad x := \frac{b-a}{\pi}\theta + a.$$

The Fourier-cosine expansion of a function f supported on an arbitrary interval $[a, b] \in \mathbb{R}$ then equals:

$$f(x) = {\sum_{k=0}^{\infty}}' A_k \cdot \cos\left(k\pi \frac{x-a}{b-a}\right), \tag{4.4}$$

with

$$A_k = \frac{2}{b-a} \int_a^b f(x) \cos\left(k\pi \frac{x-a}{b-a}\right) dx. \tag{4.5}$$

Any $\mathbb{R}$-valued finitely supported functions admits a cosine expansion. We start our derivation by truncating the infinite range of integration in (4.3), making the function finitely supported. This truncation of the integration range does not lead to a large loss of accuracy, since in order for the Fourier transform to exist, the integrands in (4.3) have to decay to zero at $\pm\infty$.

Suppose $[a, b] \subset \mathbb{R}$ is such that the truncated integral approximates its infinite counterpart very well:

$$\phi_1(\omega) := \int_a^b e^{i\omega x} f(x) dx \approx \int_{\mathbb{R}} e^{i\omega x} f(x) dx = \phi(\omega). \tag{4.6}$$

Note that, throughout the derivation, subsequent numerical approximations are denoted by a subscript, like i in ϕ_i. This is not to be confused with the subscripted series coefficients, e.g., A_k and F_k.

Comparing the definition of ϕ_1 in (4.6) with the cosine series coefficients of $f(x)$ on $[a, b]$ in (4.5) we establish the relation:

$$A_k = \frac{2}{b-a} \operatorname{Re}\left[\phi_1\left(\frac{k\pi}{b-a}\right) \cdot \exp\left(-i\frac{ka\pi}{b-a}\right)\right]. \tag{4.7}$$

It follows that $A_k \approx F_k$ for an F_k defined as

$$F_k = \frac{2}{b-a} \operatorname{Re}\left[\phi\left(\frac{k\pi}{b-a}\right) \cdot \exp\left(-i\frac{ka\pi}{b-a}\right)\right].$$

Replacing A_k by F_k in the cosine series expansion of $f(x)$ on $[a, b]$ yields an infinite series

$$f_1(x) = {\sum_{k=0}^{\infty}}' F_k \cos\left(k\pi \frac{x-a}{b-a}\right), \tag{4.8}$$

which we truncate to the first N summands and arrive at the final approximation of f on $[a, b]$

$$f_2(x) = {\sum_{k=0}^{N-1}}' F_k \cos\left(k\pi \frac{x-a}{b-a}\right). \tag{4.9}$$

The resulting error in $f_2(x)$ is composed of two parts: an error from approximating A_k by F_k and a series truncation error in the step from (4.8) to (4.9).

Note that because the cosine series expansion of *entire functions* (functions without any singularities anywhere in the complex plane except at ∞) converges exponentially, we can expect highly accurate approximations of functions that have no singularities on $[a, b]$, with a small N.

4.2 Pricing European options

We now study the COS formula for pricing European-style options whose value is given by (4.1). We derive the approximation in three steps: v_1, v_2 and v_3 will denote the subsequent approximations.

In the first step, we truncate the infinite integration range of (4.1) to $[a,b] \subset \mathbb{R}$. Since the density function rapidly decays to zero with $y \to \pm\infty$ in (4.1), the truncation does not lead to a significant loss in accuracy.

$$v_1(x,t_0) := e^{-r\tau} \int_a^b v(y,T) f(y \mid x) dy \tag{4.10}$$

In the second step, since the density function $f(y \mid x)$ is rarely known in contrast to the characteristic function, we replace the density in (4.10) by its cosine series expansion in y

$$f(y \mid x) = {\sum_{k=0}^{\infty}}' A_k(x) \cos\left(k\pi \frac{y-a}{b-a}\right),$$

with

$$A_k(x) = \frac{2}{b-a} \int_a^b f(y \mid x) \cos\left(k\pi \frac{y-a}{b-a}\right) dy,$$

Then v_1 has the following representation:

$$v_1(x,t_0) = e^{-r\tau} \int_a^b v(y,T) {\sum_{k=0}^{\infty}}' A_k(x) \cos\left(k\pi \frac{y-a}{b-a}\right) dy. \tag{4.11}$$

Interchanging the summation and integration in (4.11) and introducing the definition

$$V_k := \frac{2}{b-a} \int_a^b v(y,T) \cos\left(k\pi \frac{y-a}{b-a}\right) dy, \tag{4.12}$$

gives us

$$v_1(x,t_0) = \frac{1}{2}(b-a) e^{-r\tau} \cdot {\sum_{k=0}^{\infty}}' A_k(x) V_k.$$

Here V_k are the cosine series coefficients of $v(y,T)$ in y. Hence, the product of two real functions, $f(y \mid x)$ and $v(y,T)$ has been transformed to a product of their Fourier-cosine series coefficients.

Since these coefficients decay rapidly, we further truncate the series and arrive at the approximation v_2:

$$v_2(x,t_0) = \frac{1}{2}(b-a) e^{-r\tau} \cdot {\sum_{k=0}^{N-1}}' A_k(x) V_k.$$

As in previous section, we approximate $A_k(x)$ by $F_k(x)$ and obtain:

$$\boxed{v(x,t_0) \approx v_3(x,t_0) = e^{-r\tau} {\sum_{k=0}^{N-1}}' \operatorname{Re}\left[\phi\left(\frac{k\pi}{b-a}; x\right) e^{-ik\pi\frac{a}{b-a}}\right] V_k} \tag{4.13}$$

which is known as the COS formula for a general underlying process. In the next section we will show that V_k can be obtained analytically for plain vanilla options and that (4.13) can be simplified for certain forms of the characteristic function ϕ.

4.3 Coefficients V_k for plain vanilla options

In order to use the COS formula (4.13) for pricing options, one first has to recover the payoff series coefficients V_k. In this section we derive V_k for plain vanilla calls and puts.

Assume that the characteristic function of the log-asset price is known and represent the payoff as a function of the log-asset price. The log-asset prices are denoted by:

$$x := \ln\left(\frac{S_0}{K}\right) \text{ and } y := \ln\left(\frac{S_T}{K}\right)$$

where S_t is the price of the underlying at time t and K the strike price.

The payoff of a European vanilla option, in log-asset price, equals:

$$v(y,T) = [\alpha \cdot K(e^y - 1)]^+ \text{ with } \alpha = \begin{cases} +1 & \text{for a call,} \\ -1 & \text{for a put.} \end{cases}$$

In order to derive V_k for vanilla call and put options, we need two mathematical results.

The Fourier-cosine series coefficients χ_k of $g(y) = e^y$ on $[c,d] \subset [a,b]$, and the cosine series coefficients ψ_k of $g(y) = 1$ on $[c,d] \subset [a,b]$, are known analytically and given by:

$$\begin{aligned} \chi_k(c,d) &= \int_c^d e^y \cos\left(k\pi\frac{y-a}{b-a}\right) dy \\ &= \frac{1}{1+\left(\frac{k\pi}{b-a}\right)^2}\left[\cos\left(k\pi\frac{d-a}{b-a}\right)e^d - \cos\left(k\pi\frac{c-a}{b-a}\right)e^c\right. \\ &\quad \left.+\frac{k\pi}{b-a}\cos\left(k\pi\frac{d-a}{b-a}\right)e^d - \frac{k\pi}{b-a}\cos\left(k\pi\frac{c-a}{b-a}\right)e^c\right], \end{aligned} \tag{4.14}$$

$$\begin{aligned} \psi_k(c,d) &= \int_c^d \cos\left(k\pi\frac{y-a}{b-a}\right) dy \\ &= \begin{cases} \left[\sin\left(k\pi\frac{d-a}{b-a}\right) - \sin\left(k\pi\frac{c-a}{b-a}\right)\right]\frac{b-a}{k\pi} & k \neq 0, \\ (d-c) & k = 0. \end{cases} \end{aligned} \tag{4.15}$$

Now, for a vanilla call option we get:

$$V_k^{call} = \frac{2}{b-a}\int_a^b K(e^y-1)^+ \cos\left(k\pi\frac{y-a}{b-a}\right) dy \tag{4.16}$$

$$= \frac{2}{b-a}\int_0^b K(e^y-1)s \cos\left(k\pi\frac{y-a}{b-a}\right) dy = \frac{2}{b-a}K(\chi_k(0,b) - \psi_k(0,b)), \tag{4.17}$$

with χ_k and ψ_k given by (4.14) and (4.15), respectively. Similarly, for a vanilla put, we obtain:

$$V_k^{put} = \frac{2}{b-a}K\left(-\chi_k(a,0) + \psi_k(a,0)\right). \tag{4.18}$$

4.4 Simplified formula

Formula (4.13) can be greatly simplified for models whose characteristic function admits the following decomposition:

$$\phi(\omega; \boldsymbol{x}) = \varphi \cdot e^{i\omega\boldsymbol{x}} \text{ with } \varphi := \phi(\omega, 0).$$

For instance, this holds for additive processes, Lévy processes and the Heston model. This property enables many option prices for different strikes to be computed simultaneously. Note that boldfaced values represent vectors.

The pricing formula from (4.13) now simplifies to:

$$v(\boldsymbol{x}, t_0) \approx e^{-r\tau} {\sum_{k=0}^{N-1}}' \operatorname{Re}\left[\varphi\left(\frac{k\pi}{b-a}\right) e^{ik\pi \frac{\boldsymbol{x}-a}{b-a}}\right] \boldsymbol{V}_k$$

The coefficients of vanilla options V_k in (4.16) and (4.18) can now be represented as a vector multiplied by a scalar:

$$\boldsymbol{V}_k = U_k \boldsymbol{K}$$

with

$$U_k = \begin{cases} \frac{2}{b-a}(\chi_k(0,b) - \psi_k(0,b)) & \text{for a call,} \\ \frac{2}{b-a}(-\chi_k(0,b) + \psi_k(0,b)) & \text{for a put.} \end{cases}$$

Finally, the simplified pricing formula equals:

$$\boxed{v(\boldsymbol{x}, t_0) \approx \boldsymbol{K} e^{-r\tau} \operatorname{Re}\left[{\sum_{k=0}^{N-1}}' \varphi\left(\frac{k\pi}{b-a}\right) U_k \cdot e^{ik\pi \frac{\boldsymbol{x}-a}{b-a}}\right],} \tag{4.19}$$

where the sum can be written as a matrix-vector product if $\boldsymbol{K}$ and $\boldsymbol{x}$ are vectors.

4.5 Error analysis

The overall error in the derivation of the COS formula consists of three parts:

1. The truncation of the integration range in the risk-neutral valuation formula (4.1):

$$\epsilon_1 = v(x, t_0) - v_1(x, t_0) = \int\limits_{\mathbb{R}\backslash[a,b]} v(y, T) f(y \mid x) dy \tag{4.20}$$

2. The substition of the density $f(x)$ by its Fourier-cosine series expansion on the truncated range:

$$\epsilon_2 = v_1(x, t_0) - v_2(x, t_0) = \frac{1}{2}(b-a)e^{-r\tau} \sum_{k=N}^{\infty} A_k(x) \cdot V_k \tag{4.21}$$

3. The substitution of the Fourier-cosine series coefficients by the approximation relying on the characteristic function:

$$\epsilon_3 = v_2(x, t_0) - v_3(x, t_0) \tag{4.22}$$

$$= e^{-r\tau} {\sum_{k=0}^{N-1}}' \operatorname{Re}\left[\int\limits_{\mathbb{R}\backslash[a,b]} e^{ik\pi \frac{y-a}{b-a}} f(y \mid x) dy\right] V_k \tag{4.23}$$

Since the coefficients V_k of vanilla options are known V_k are known analytically, we do not need to account for any error in the coefficients V_k.

The decay rate of the Fourier-cosine series coefficients is the key to finding an upper bound of the errors. The convergence rate of the Fourier-cosine series is dependent on the properties of the function on the expansion interval. The authors then use the following mathematical definitions and results to obtain an upper bound for the error.

Definition 4.1 (Algebraic index of convergence)**.** The algebraic index of convergence $n \geq 0$ is the largest number for which

$$\lim_{k \to \infty} |A_k| \, k_n < \infty, \quad k >> 1,$$

where A_k are the coefficients of the series. An alternative definition is that if the coefficients A_k of a series decay asymptotically as

$$A_k \sim O(\frac{1}{k^n}), \quad k >> 1,$$

then n is the algebraic index of convergence.

Definition 4.2 (Exponential index of convergence)**.** In case when the algebraic index of convergence $n \geq 0$ is unbounded, meaning the A_k decrease faster than $\frac{1}{k^n}$ for any finite n, the series is said to exhibit exponential convergence.

Alternatively, if for some constant γ, called the asymptotic rate of convergence, and some $r > 0$, called the index of convergence,

$$A_k \sim O(\exp(-\gamma k^r)), \quad k >> 1,$$

then the series shows exponential convergence. For $r < 1$, the convergence is called subgeometric. For $r = 1$, the convergence is either called supergeometric (for some $j > 0$) with

$$A_k \sim O(k^{-n} \exp(-\frac{k}{j} \ln(k))).$$

or geometric with

$$A_k \sim O(k^{-n} \exp(-\gamma k)). \tag{4.24}$$

Result 4.3 (Convergence of Fourier-cosine series [27] p.70-71)**.** If $g(x) \in C^\infty([a, b] \subset \mathbb{R})$, then its Fourier-cosine series expansion on $[a, b]$ has geometric convergence. The constant γ in (4.24) is determined by the location in the complex plane of the singularities nearest to the expansion interval. Exponent n is determined by the type and strength of the singularity.

If a function $g(x)$, or any of its derivatives, is discontinuous, its Fourier-cosine series coefficients show algebraic convergence. Integration-by-parts show that the algebraic index of convergence n, is at least as large as n', with the $n' - th$ derivative of $g(x)$ integrable.

Result 4.4 (Series truncation of algebraically converging series)**.** It can be shown that the series truncation error of an algebraically converging series behaves like

$$\sum_{k=N+1}^{\infty} \frac{1}{k^n} \sim \frac{1}{(n-1)N^{n-1}}. \tag{4.25}$$

The proof can be found in [28].

We are now able to prove the following.

Result 4.5. Error ϵ_3 given by (4.22) can be bounded by:

$$|\epsilon_3| < |\epsilon_1| + Q \, |\epsilon_4| \tag{4.26}$$

where Q is a constant independent of N from (4.22) and

$$\epsilon_4 := \int_{\mathbb{R} \backslash [a,b]} f(y \mid x) dy. \tag{4.27}$$

Proof. Assuming $f(y \mid x)$ is a real function, we can rewrite (4.22) as

$$\epsilon_3 = e^{-r\tau} {\sum_{k=0}^{N-1}}' V_k \int_{\mathbb{R}\backslash[a,b]} \cos\left(k\pi\frac{y-a}{b-a}\right) f(y \mid x)dy. \tag{4.28}$$

Interchanging the summation and integration, rewriting ${\sum_{k=0}^{N-1}}'$ as $({\sum_{k=0}^{\infty}}' - {\sum_{k=N}^{\infty}}')$ and replacing the Fourier-cosine expansion of $v(y,T)$ in y by $v(y,T)$:

$$\epsilon_3 = e^{-r\tau} \int_{\mathbb{R}\backslash[a,b]} \left[v(y,T) - \sum_{k=N}^{\infty} \cos\left(k\pi\frac{y-a}{b-a}\right) \cdot V_k\right] f(y \mid x)dy \tag{4.29}$$

$$= \epsilon_1 - e^{-r\tau} \int_{\mathbb{R}\backslash[a,b]} \left[\sum_{k=0}^{N-1} \cos\left(k\pi\frac{y-a}{b-a}\right) \cdot V_k\right] f(y \mid x)dy. \tag{4.30}$$

According to results 4.3 and 4.4 the V_k exhibit at least algebraic convergence and therefore admit an upper bound as follows,

$$\left|\sum_{k=N}^{\infty} \cos\left(k\pi\frac{y-a}{b-a} \cdot V_k\right)\right| \leq \sum_{k=N}^{\infty} |V_k| \leq \frac{Q*}{(N-1)^{n-1}} \leq Q*, \text{ for } N >> 1, n \geq 1.$$

where $Q^* > 0$ is a positive constant. Combining this with (4.29) we get:

$$|\epsilon_3| < |\epsilon_1| + Q\,|\epsilon_4|$$

with

$$Q := e^{-r\tau}, \text{ and } \epsilon_4 := \int_{\mathbb{R}\backslash[a,b]} f(x \mid y)dy.$$

□

As it turns out, two of the three error components, ϵ_1 and ϵ_3, are related to the choice of the truncation range. When the truncation range is sufficiently large, the overall error is dominated by ϵ_2.

It is clear from (4.21) that ϵ_2 depends on both the Fourier-cosine series coefficients of the density, A_k, and those of the payoff functions, V_k. We now introduce the reasonable assumption that the density function f is usually smoother than the typical payoff functions in finance and therefore A_k decay faster than V_k. Accordingly, the product of A_k and V_k converges faster than either A_k or V_k and admits the following bound:

$$\left|\sum_{k=N}^{\infty} A_k(x) \cdot V_k\right| \leq \sum_{k=N}^{\infty} |A_k(x)|\,. \tag{4.31}$$

Hence, error ϵ_2 is dominated by the series truncation error of the density function.

Result 4.6 (Series truncation error of geometrically converging series [27] p.48)**.** If a series exhibits geometrical convergence, then the error after truncation of the expansion after $(N+1)$ terms, $E_T(N)$, equals

$$E_T(N) \sim P^* e^{-N\nu}. \tag{4.32}$$

The constant $\nu > 0$ is called the asymptotic rate of convergence of the series, and it satisfies

$$\nu = \lim_{n\to\infty}\left(-\frac{\log|E_T(n)|}{n}\right), \tag{4.33}$$

and P^* is a factor which varies less than exponentially with N.

Result 4.7. Error ϵ_2 converges exponentially in the case of density functions $g(x) \in C^\infty([a,b])$.

$$|\epsilon_2| < \exp(-(N-1)\nu) \tag{4.34}$$

Proof. Simply apply Proposition 4.6 to (4.31). ☐

Result 4.8. Error ϵ_2 for densities having discontinuous derivatives admits the following bound:

$$|\epsilon_2| < \frac{\bar{P}}{(N-1)^{\beta-1}}, \tag{4.35}$$

where $\bar{P}$ is a constant, $\beta \geq n \geq 1$ and n is the algebraic index of convergence of V_k.

Proof. Given a proper choice of the truncation of the infinite integration range, it follows from (4.20), (4.26), (4.34) and (4.35) that the overall error converges either exponentially for density functions from $C^\infty([a,b] \subset \mathbb{R})$, i.e.

$$|\epsilon| < 2\,|\epsilon_1| + Q\,|\epsilon_4| + Pe^{-(N-1)\nu}, \tag{4.36}$$

or algebraically for density functions with a discontinuity in one of its derivatives, i.e.

$$|\epsilon| < 2\,|\epsilon_1| + Q\,|\epsilon_4| + \frac{\bar{P}}{(N-1)^{\beta-1}}. \tag{4.37}$$

☐

4.6 Choice of the truncation range

The authors in [1] suggest the following truncation interval for maturities ranging from $T = 0.1$ to $T = 10$:

$$[a,b] := \left[c_1 - L\sqrt{c_2 + \sqrt{c_4}}, c_1 + L\sqrt{c_2 + \sqrt{c_4}}\right] \text{ with } L = 10,$$

where c_n denotes the n-th cumulant of $\ln\left(\frac{S_T}{K}\right)$. The cumulant c_4 captures fat tails and sharp peaks that are characteristic of densities of many Lévy processes. Note that larger values of L require larger N for the same level of accuracy.

For extremely short maturities, like $T = 0.001$, a truncation range taking cumulant c_6 into account is more accurate:

$$[a,b] := \left[c_1 - L\sqrt{c_2 + \sqrt{c_4 + \sqrt{c_6}}}, c_1 + L\sqrt{c_2 + \sqrt{c_4 + \sqrt{c_6}}}\right]$$

However, c_6 is usually relatively difficult to derive.

Another important point concerns the numerical stability of call option value calculations. Since the call payoff grows exponentially with the log-price, this introduces a significant cancellation error for larger values of L in (4.16). On the other hand, the payoff of a put option is bounded by K and therefore they do not suffer from this. For pricing call options it is thus recommended to compute the put option value and apply the well known put-call parity.

$$v^{call}(x,t_0) = v^{put}(x,t_0) + S_0 - Ke^{-rT} \tag{4.38}$$

Chapter 5

Parallel computing with CUDA

In this chapter we take the opportunity to briefly describe the CUDA technology and provide the reader with a crash course in CUDA parallel programming, just enough for them to understand the implementation of the underlying algorithm that we use for pricing. Sections 5.1 to 5.3 provide an introduction and cover the difference between serial and parallel computing, some historical developments and the high-level architectural differences between standard and graphics processors. Then, in section 5.4 we cover the basic CUDA syntax in order to understand the implementation behind the algorithm described in section 5.5. Finally, in section 5.6 we explain how parallel computing speeds up pricing in our model, together with some results. This chapter is based on [29, 30, 31].

5.1 Serial and parallel computing

Traditionally, software has been written for serial computation, where a problem is broken into a discrete series of instructions, which get executed sequentially one after another on a single processor unit, and only one instruction may execute at any moment in time.

Parallel computing, on the other hand, makes use of multiple compute resources *simultaneously* in order to solve a computational problem. A problem gets broken into discrete parts that can be solved concurrently and independently, each part is then further broken down to a series of instructions that execute simultaneously on different processors, and an overall control mechanism takes care of coordinating them. A prerequisite for parallel computing, apart from the obvious requirement for hardware that enables simultaneous execution of multiple program instructions, is the underlying problem's aptitude to being broken down into discrete pieces that can be solved simultaneously. Not all problems can be decomposed in such a way, but for those that do, it might result in a significant speed up.

In recent years, a lot has been done towards the computing industry's widespread shift to parallel computing. Pretty much every consumer computer today ships with a multicore central processor. Starting with the introduction of dual-core, low-end netbook machines to 8- and 16-core workstation computers, parallel computing is no longer reserved exclusively for exotic supercomputers or mainframes. Moreover, smartphones and other electronic devices have begun incorporating parallel computing capabilities in an effort to provide functionality well beyond those of their predecessors. Parallel computing has become mainstream and offers great potential to those willing to try it.

For many years, one of the key ways of improving performance of processors has been to increase the speed at which the processor's clock operated. The first personal computers of the early 1980s, consumer central processing units (CPUs) ran with internal clocks operating around 1MHz. Fast forward 30-35 years, most desktop processors have clock speeds ranging from 1GHz to 4GHz, nearly a thousand times faster than the clock of the original personal computer. Although increasing the CPU clock speed is far from the only method of improving computing performance has been improved, it has always been a reliable source of performance gains.

More recently, because of fundamental and physical limitations in the fabrication of integrated circuits, arose the need for an alternative to the upward-spiraling clock speeds as a means of extracting increased computational power.

In recent years, however, consumer CPU manufacturers have been forced to look for alternatives to this traditional source of increased computational power. Because of various fundamental limitations in the fabrication of integrated circuits, it is no longer feasible to rely on upward-spiraling processor clock speeds as a means for extracting additional power from existing architectures. Because of power and heat restrictions as well as a rapidly approaching physical limit to transistor size, researchers and manufacturers have begun to look elsewhere. Outside the world of consumer computing, supercomputers have for decades extracted massive performance gains by relying on the same principles, resulting in an astronomical performance gain of supercomputers. However, in addition to dramatic improvements in the performance of a single processor, supercomputer manufacturers have also made great leaps in performance by steadily increasing the number of processors. It is not uncommon for the fastest supercomputers to have tens or hundreds of thousands of processor cores working in tandem.

Following this approach, in 2005, faced with an increasingly competitive marketplace and few alternatives, leading consumer CPU manufacturers began offering processors with two computing cores in place of one. Over the following years, they followed this development with the release of three-, four-, six-, and eight-core central processor units. Sometimes referred to as the multicore revolution, this trend has marked a huge shift in the evolution of the consumer computing market. Today, it is virtually impossible to buy a desktop CPU containing a single computing core. Even low-end, low-power CPUs ship with two or more cores. Releases of CPUs with an even larger number of cores have already been announced, further confirming that parallel computing has arrived for good.

There was a time not so long ago when parallel computing was looked upon as an “exotic” pursuit and even regarded as a specialty within the computer science field. With the increased presence of multi-core hardware and parallel computing in recent years, this perception has changed profoundly. The computing world has shifted to the point where parallel programming and the ability to solve problems using it have become standard practice. Parallel computing plays a large role in the computing world today and is only becoming more important as the time goes by.

5.2 The move towards general purpose GPU computing

Compared to the CPU’s traditional data processing pipeline, performing general-purpose computations on a graphics processing unit (GPU) is a new concept. In fact, the GPU itself is relatively new compared to the computing field at large. However, the idea of computing on graphics processors is older than it seems. Let us quickly review the historical development of graphic cards.

In the late 1980s and early 1990s, the popularity of graphically driven operating systems such as Microsoft Windows helped create the need for a new type of processor, one that would handle the calculations associated with the graphical elements of the operating system. This resulted in a dramatic revolution of the state of graphics processing. In the early 1990s, some of the first 2D display accelerators for personal computers were sold. These display accelerators provided hardware-assisted bitmap operations to assist in the display and usability of graphical operating systems.

Meanwhile, in the world of professional computing, a company called Silicon Graphics spent the 1980s popularizing the use of three-dimensional graphics. Initially the focus was on government and defense applications, scientific and technical visualization, as well as tools for creating grand cinematic effects. Then in 1992, Silicon Graphics opened the programming interface to its hardware by releasing the OpenGL library, which was meant to be used as a standardized, platform-independent method for writing 3D graphics applications. As it is always the case with technology, it was only a matter of time before OpenGL found its way into consumer applications.

By the mid-1990s, the demand for consumer applications employing 3D graphics had escalated rapidly. Interestingly enough, one of the main drivers of 3D development were computer games, which accelerated the adoption od 3D graphics in consumer computing. At the same time, companies such as NVIDIA, ATI

Technologies, and 3dfx Interactive began releasing graphics accelerators that were affordable enough to the average consumer. These developments set a foundation for 3D graphics as an upcoming technology. With the release of NVIDIA's GeForce 256, for the first time, transform and lighting computations could be performed directly on the graphics processor, thereby enabling creation of even more visually interesting applications. Since transform and lighting were already integral parts of the OpenGL graphics pipeline, the GeForce 256 marked the beginning of a natural progression where more and more of the graphics pipeline would be implemented directly on the graphics processor.

Arguably the most important breakthrough in GPU technology from a parallel-computing point of view is the release of the GeForce 3 series graphics card in 2001. The GeForce 3 series was the first graphics chip to implement Microsoft's then-new DirectX 8.0 standard. What was so special about this standard is the requirement for the hardware to contain both programmable vertex and programmable pixel shading stages. For the first time in history, developers had some control over the exact computations that would be performed on their GPUs.

The release of GPUs that possessed programmable pipelines attracted many researchers to the possibility of using graphics hardware for more than simply OpenGL- or DirectX-based rendering. However, performing arbitrary calculations on the GPU was an extremely convoluted procedure. Because the only means of interaction with the GPU were still the standard graphics APIs such as OpenGL and DirectX, one had to be creative and use a lot of workarounds in order to present a general-purpose computation as a rendering procedure to the GPU.

Simply put, the GPUs of the early 2000s were designed to produce a color for every pixel on the screen using programmable arithmetic units known as pixel shaders. A pixel shader uses its (x, y) position on the screen as well as some additional information to combine various inputs in computing a final color. The additional information includes input colors, texture coordinates, or other attributes that would be passed to the shader when it ran. Since the arithmetic performed on the input colors and textures was completely controlled by the programmer, researchers observed that these input "colors" could actually represent any data.

By treating the inputs that were actually numerical data as colors, programmers could then program the pixel shaders to perform arbitrary computations on this data. The results would be handed back to the GPU as the final pixel "color", although the colors would simply be the result of whatever computations the programmer had instructed the GPU to perform on their inputs. This result would then be read by the programmer, and the GPU would never be the wiser. In essence, the whole procedure was a big workaround "tricking" the GPU into performing nonrendering tasks by making those tasks appear as if they were a standard rendering. Although very clever, this approach was also too restrictive in resource management and complicated to achieve a critical mass of developers. Plus, it meant that anyone interested in using a GPU to perform general-purpose computations still had to master a graphics API and special shading languages, either OpenGL or DirectX, and the corresponding workarounds. Understandably, this was too much of a hurdle for the programming model to gain wide acceptance. But, it also indicated a demand for general-purpose GPU computing.

In November 2006, NVIDIA unveiled the industry's first DirectX 10 GPU, the GeForce 8800 GTX, which marks the beginning of general-purpose GPU computing of today. The GeForce 8800 GTX was the first GPU to be built with NVIDIA's Compute Unified Device Architecture (CUDA). This architecture introduced new components designed strictly for GPU computing and aimed to get rid of limitations of previous graphics processors concerning general-purpose computation.

In contrast to previous generations of GPUs that partitioned computing resources into vertex and pixel shaders, the CUDA architecture introduced a unified shader pipeline, which allowed each and every arithmetic logic unit (ALU) on the chip to be mobilized for performing general-purpose computations. These ALUs were built to comply with IEEE requirements for single-precision floating-point arithmetic and were designed to use an instruction set tailored for general computation rather than specifically for graphics. Moreover, the execution units on the GPU had arbitrary read and write access to memory as well as access to a software-managed cache known as shared memory. All of these features were added in order to create a GPU that would excel at computation as well as perform well at traditional graphics tasks.

Along the innovation on the hardware level in form of CUDA, NVIDIA also developed a programming language for interacting with the new capabilities. The time of complicated workarounds was over. NVIDIA took industry standard C and added a relatively small number of keywords in order to harness some of the special features of the CUDA Architecture, giving birth to CUDA C. A public compiler for this language was released shortly after the GeForce 8800 GTX. And with that, CUDA C became the first language specifically designed by a GPU company to facilitate general-purpose computing on GPUs.

5.3 CPU and GPU comparison

The insatiability of the market demand for real-time, high-definition 3D graphics, has driven the evolution of the programmable graphic processor unit (GPU) into a highly parallel, multithreaded, many core processor with massive horsepower and very high memory bandwidth.

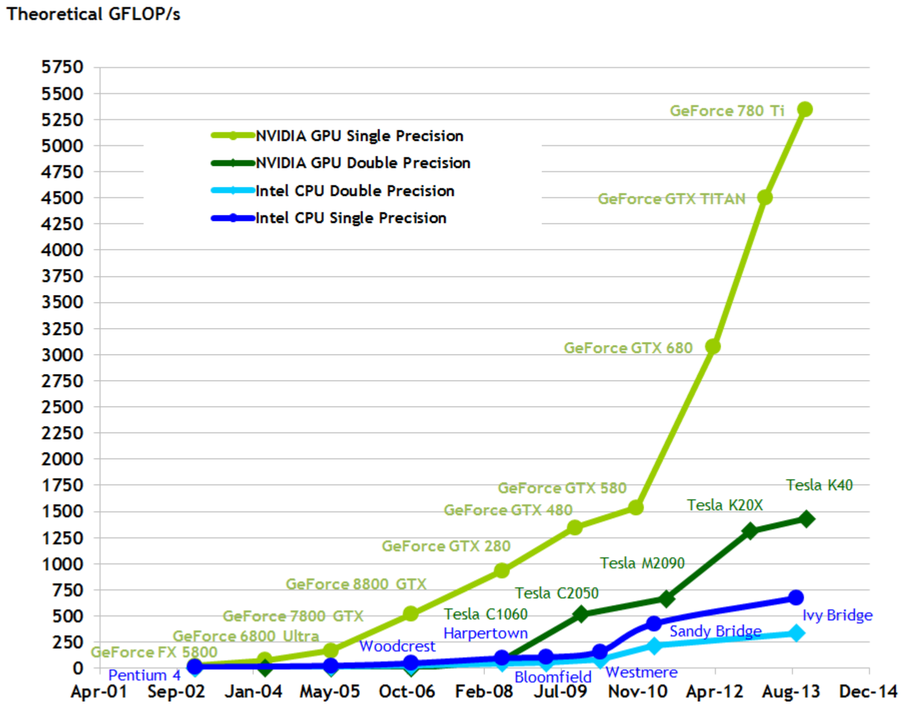


Figure 5.1: Comparison of the floating-point operations per second and memory bandwidth for the CPU and GPU

The GPU is specialized for computations characteristic of graphics rendering, meaning it excels in compute-intensive, highly parallel computations. It is designed in a way such that more transistors are devoted to data processing rather than data caching and flow control, as illustrated in fig. 5.2. This is the reason behind the superior floating-point capability between of the GPU over the CPU.

The GPU is especially well-suited for programs including many data-parallel computations, where the same program is executed on many data elements in parallel with high arithmetic intensity (the ratio of arithmetic operations to memory operations). The fact that the same program is executed for each data element entails that there is no need for sophisticated flow control. In addition, since the execution of the program includes many data elements and has a high arithmetic intensity, the memory access latency can be hidden with calculations instead of big data caches. But, problems other than 3D rendering can make use of this hardware. For example, image processing, pattern recognition, signal processing, physics simulation, computational biology and, in our case, computational finance.

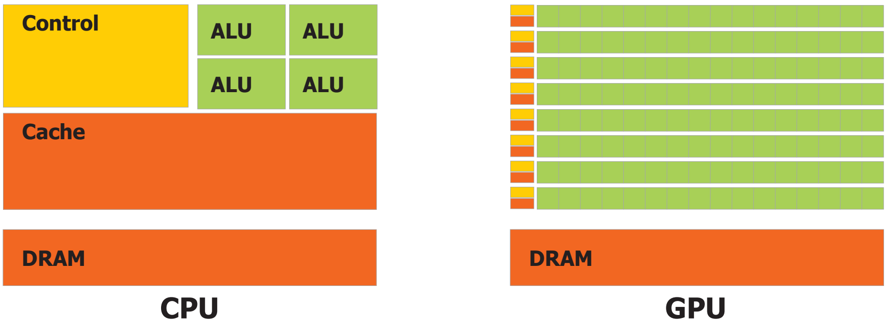


Figure 5.2: Comparison of CPU and GPU architecture: the GPU devotes more transistors to data processing.

5.4 Parallel programming with CUDA

In this section we take a look at just how simple parallel computing with CUDA really is. We explain the basic methodology, the syntax and provide sample codes. Basic C knowledge is an asset in understanding the sample codes and some of the terms.

5.4.1 A scalable programming model

At the core of the CUDA parallel programming model are three key abstractions, which are exposed to the programmer as a minimal set of C language extensions:

1. A hierarchy of thread groups.
2. A hierarchy of shared memories.
3. Barrier synchronization.

These abstractions allow for fine-grained data parallelism and thread parallelism, nested within coarse-grained data parallelism and task parallelism. This allows partitioning of a given problem into coarse sub-problems that can be solved independently in parallel by blocks of threads, whereas each sub-problem gets partitioned further into finer pieces that can be solved cooperatively in parallel by all threads within the block.

This way of decomposing problems enables automatic scalability. Each block of threads can be scheduled on any of the available streaming multiprocessors (SM) within a GPU, in any order, concurrently or sequentially. Therefore, a compiled CUDA program can execute on any number of multiprocessors, whose exact count is handled by the runtime system. This scalable programming model allows the CUDA architecture to span a wide market range by simply scaling the number of multiprocessors and memory partitions.

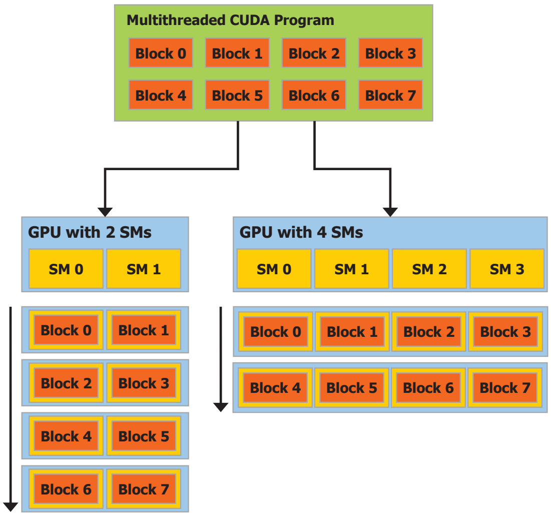


Figure 5.3: Automatic scaling: a CUDA GPU is built around an array of streaming multiprocessors (SMs). A multithreaded program is then partitioned into blocks of threads that execute independently from each other. A GPU with more multiprocessors will automatically execute the program in less time than a GPU with fewer multiprocessors.

5.4.2 Kernels, thread blocks and memory hierarchy

The CUDA programming model enables efficient and relatively simple interaction of the (serial) code executing on the CPU and the (parallel) code executing on the GPU. The approach that uses a system with more than one kind of processor to complete a task at hand is known as *heterogenous computing*. We will use the terms *host* to refer to the CPU and its memory, and *device* to refer to the GPU and its memory.

A *kernel* is a function that is executed in parallel on the CUDA device. In CUDA C, kernels are defined as a C `void` function using the additional global declaration specifier `__global__`. A kernel invocation, however, differs from a standard function call. Since kernels get executed in blocks of threads, which we will discuss promptly, an execution configuration with the number of blocks and threads, as well as the size of shared memory per block, is a crucial ingredient of a kernel call.

Let us look at the following code snippet as an example of a kernel definition and invocation:

```
// Kernel definition
__global__ void my_kernel()
{
        ...
}

int main()
{
        ...
        // Kernel invocation with M blocks of N threads
        my_kernel<<<M, N>>>();
        ...
}
```

This launches M blocks of N threads totalling to $M \cdot N$ instances of the kernel `my_kernel` on the GPU running in parallel. So, what exactly is the difference between thread blocks and threads within a block? Whereas thread blocks are required to execute independently, threads within a block are aware of each other. They can cooperate by sharing data through *shared memory* and by synchronizing their execution.

Each thread executing the kernel is given a unique thread and block ID that are accessible within the kernel through the built-in `threadIdx` and `blockIdx` variables, respectively. Consider the following sample code that adds two vectors A and B of size N and stores the result into vector C:

```
// Kernel definition
__global__ void VecAdd(float* A, float* B, float* C)
{
        int i = threadIdx.x;
        C[i] = A[i] + B[i];
}

int main()
{
        ...
        // Kernel invocation with N threads
        VecAdd<<<1, N>>>(A, B, C);
        ...
}
```

Here, the kernel is launched in one block of N threads. Each of the N threads of the block that execute `VecAdd()` performs one pair-wise addition and stores the result in the array `C`.

For convenience, `threadIdx` is a 3-component vector, so that threads can be identified using a 1-dimensional, 2-dimensional, or 3-dimensional thread index, forming a 1-dimensional, 2-dimensional, or 3-dimensional thread block. This was done in order to simplify the code and avoid common index transformations. It also provides a natural way to invoke computation across the elements in a domain such as a vector, matrix, or volume.

The relationship between the thread index and its thread ID is straightforward. For a 1-dimensional block, they are the same; for a 2-dimensional block of size (D_x, D_y), the thread ID of a thread of index (x, y) is $(x + yD_x)$; for a 3-dimensional block of size (D_x, D_y, D_z), the thread ID of a thread of index (x, y, z) is $(x + yD_x + zD_xD_y)$.

As an illustration of multidimensional thread indices, the following code adds two matrices A and B of size $N \times N$ and stores the result into matrix C:

```
// Kernel definition
__global__ void MatAdd(float A[N][N], float B[N][N], float C[N][N])
{
        int i = threadIdx.x;
        int j = threadIdx.y;

        C[i][j] = A[i][j] + B[i][j];
}

int main() {
        ...
        // Kernel invocation with one block of N * N * 1 threads
        int numBlocks = 1;
        dim3 threadsPerBlock(N, N);
        MatAdd<<<numBlocks, threadsPerBlock>>>(A, B, C);
        ...
}
```

The number of threads per block is limited, since all threads of a block are expected to reside on the same processor core and must share the limited memory resources of that core. On current GPUs, a thread block may contain up to 1024 threads.

However, to work around this limitation, given that all thread executions are independent of each other, we can execute the kernel in multiple equally-shaped thread blocks, so that the total number of threads is equal to the number of threads per block times the number of blocks.

Just like threads, blocks can be organized into a one-dimensional, two-dimensional, or three-dimensional grid of thread blocks. In applications, the number of thread blocks in a grid is usually dictated by the size of the data being processed or the number of processors in the system, which it can greatly exceed.

The number of threads per block and the number of blocks per grid specified in the execution configuration syntax `<<< , >>>` syntax can be of type `int` or `dim3`. Two-dimensional blocks or grids can be specified as in the above code snippet. Like threads, each block within the grid can be identified by a one-dimensional, two-dimensional, or three-dimensional index accessible within the kernel through the built-in `blockIdx` variable. The dimension of the thread block is accessible within the kernel through the built-in `blockDim` variable.

Modifying the previous `MatAdd()` example to handle multiple blocks, the code now reads:

```
// Kernel definition
__global__ void MatAdd(float A[N][N], float B[N][N], float C[N][N])
{
        int i = blockIdx.x * blockDim.x + threadIdx.x;
        int j = blockIdx.y * blockDim.y + threadIdx.y;

        if (i < N && j < N) C[i][j] = A[i][j] + B[i][j];
}

int main()
{
        ...
        // Kernel invocation
        dim3 threadsPerBlock(16, 16);
        dim3 numBlocks(N / threadsPerBlock.x, N / threadsPerBlock.y);
        MatAdd<<<numBlocks, threadsPerBlock>>>(A, B, C);
        ...
}
```

Note that this example, for the sake of simplicity, assumes even divisibility of the number of threads per grid in each dimension by the number of threads per block in that dimension. The grid is then created with enough blocks to have one thread per matrix element as before. A common (arbitrary) choice of a thread block size is 16x16 (256 threads).

Thread blocks are required to execute independently, so that they can be executed in any order, no matter if serially or in parallel. This independence requirement allows thread blocks to be scheduled in any order across any number of cores, as in fig. 5.3, enabling programmers to write code that scales with the number of cores.

As already mentioned, threads within a block can cooperate by sharing data through shared memory whose size has to be determined at the time of the kernel invocation and by synchronizing their execution to coordinate memory accesses. More precisely, one can specify synchronization points in the kernel by calling the `__syncthreads()` intrinsic function; `__syncthreads()` acts as a barrier at which all threads in the block must wait before any is allowed to proceed. For efficient cooperation, the shared memory is expected to be a low-latency memory near each processor core and `syncthreads()` is expected to be lightweight.

CUDA threads are able to access data from multiple memory spaces during their execution. Each thread has private local memory. Each thread block has its own shared memory with the same lifetime as the block that is accessible to all threads of the block. All threads have access to the same global memory. On top of that, all threads can access two additional read-only memory spaces: the constant and texture memory spaces. These two memory spaces are irrelevant for our purposes.

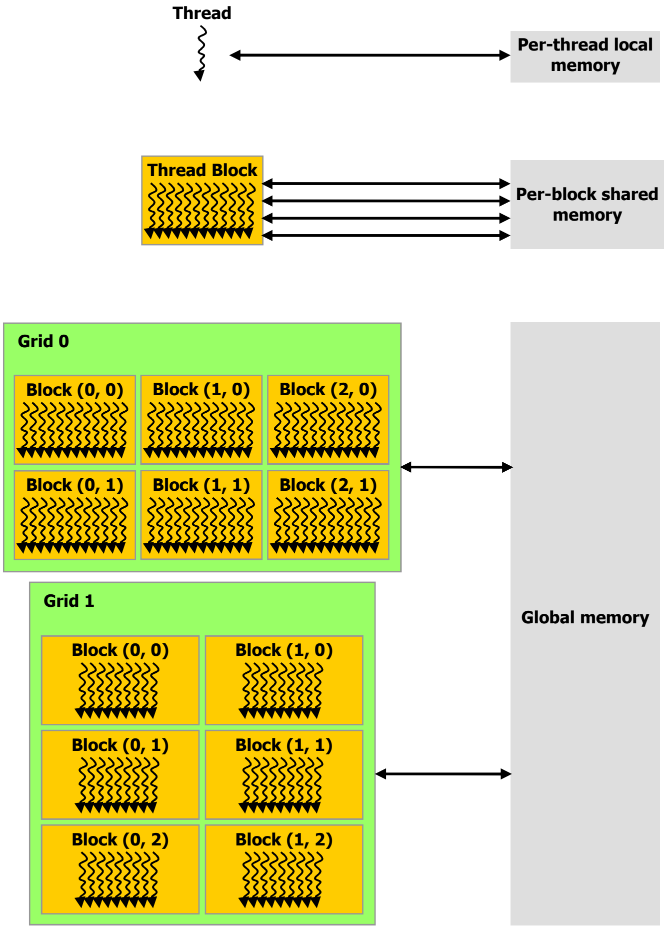


Figure 5.4: Memory access hierarchy

5.5 Parallel reduction sum algorithm

We now discuss how a calculation of a sum of an array can be done in parallel on the GPU, resulting in a significant speed up, using the *parallel reduction* approach.

Summing up an arrray of n elements using a single thread is pretty straightforward:

```
int sum = 0;

for( int i=0; i < N; ++i )
        sum += x[i];
```

At this stage, assume that $n = 2^k$ is smaller than the maximum number of threads per block. So how do we go about calculating the sum using multiple threads? Intuitively, one might assume that in a parallel calculation of the sum each thread of a thread block would then just add its corresponding element to some sum variable that is shared between the threads. However, this is not possible, since it would just introduce a data race in which different threads would try updating the same variable in shared memory simultaneously and end up rewriting it.

To avoid this, we introduce the parallel reduction sum algorithm. The idea is to compute the sum using multiple threads in such a way that two different threads never attempt to modify the same variable simultaneously.

We introduce an array S of size n shared between the threads that is going to store the partial sums. We set $S_i = x_i$ and thread with ID i handles the i-th partial sum S_i. We compute the stepwise partial sums the following way. In step m, a thread with thread ID $i \in \{0, \ldots, \frac{n}{2^m} = 2^{k-m}\}$ adds the value of $S_{i+2^{k-m}}$ to S_i, the others simply idle. Then the threads synchronize, ensuring the memory write is over, and go over to the next step $m+1$ until $m > k$. In the end, S_0 contains the total sum. For an array of length $n = 2^k$ this takes $\log_2(n) = k$ steps in each thread. Since all n threads are running in parallel, we managed to reduce the complexity from $O(n)$ to $O(\log_2(n))$. Because a picture is worth a thousand words, fig. 5.5 illustrates the reduction procedure.

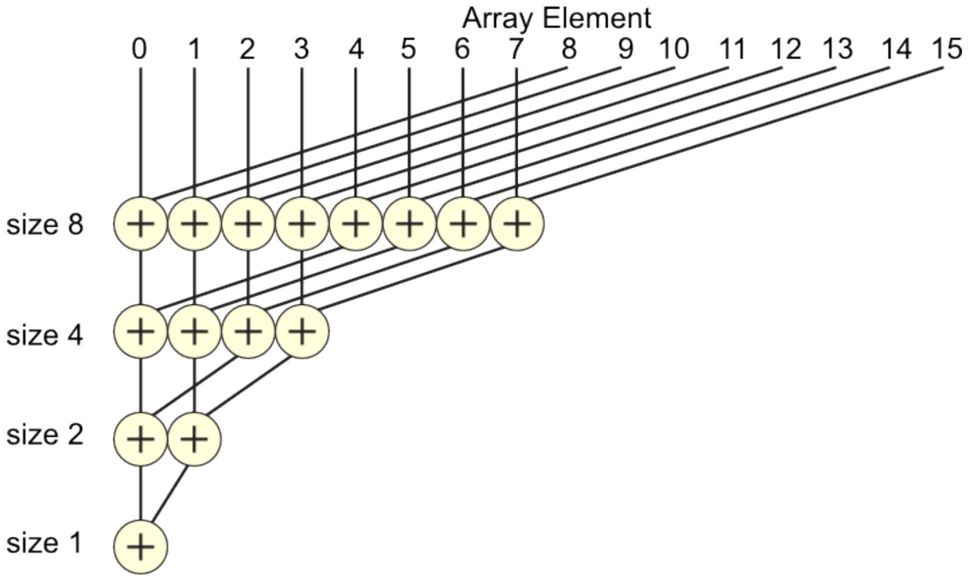


Figure 5.5: Parallel reduction summation tree

The following code snippet implements the parallel reduction sum algorithm:

```
__global__ void ArraySum( float *x, double* sum )
{
        // shared data of the thread block, contains the partial sums
        extern __shared__ double S[];

        int i = threadIdx.x;
        S[i] = x[i];
        __syncthreads();

        for (int offset = blockDim.x / 2; offset > 0; offset /= 2)
        {
                if (i < offset) sdata[i] += sdata[i + offset];
                __syncthreads();
        }

        *sum = S[0];
}

int main()
{
        ...
        // Kernel invocation with n threads and the size of shared memory
        // needed for the partial sum array
        VecAdd<<<1, n, n*sizeof(float)>>>(x, sum);
        ...
}
```

If the size of the array is not a power of two, we just fill it up with zeroes until it is. For longer arrays we can use multiple thread blocks. There are two ways to go about this: either we divide it in m thread blocks and sum the results up sequentially at the end, or we use the approach of recursively invoking the kernel on the GPU. For the latter, devices with Compute Capability 3.5 are required.

5.6 Speeding up pricing and calibration

The COS formula (4.19) basically reduces to a sum where the summands are independent of each other, making it ideal for parallel processing.

Here is the basic outline of our implementation of option pricing:

1. Copy the option data of n options from the host to the device memory.
2. Copy the model parameter data from the host to the device memory.
3. Run n blocks of k threads with enough shared memory which equals to an array of size k per block. Each block handles one option. Each of the threads in the block handles one summand of the COS formula:
    - 3.1. Compute the k-th summand of the COS formula.
    - 3.2. Synchronize all threads within the block.
    - 3.3. Run their portion of the parallel reduction sum algorithm, as described in section 5.5. In the end, the total sum has been calculated in thread with ID 0 and stored in position 0 of the array.
4. Copy the results from device memory to host memory.
5. Deallocate the model parameter data from the device memory.
6. Deallocate the option data from the device memory.

For the purpose of calibration, when many prices have to be computed using the same option data but different parameters in order to calculate the target function that gets minimised, we only copy the option data once at the beginning of the calibration:

1. Copy the option data of n options from the host to the device memory.
2. The parameter estimation loop:
    - 2.1. "Guess" the model parameters.
    - 2.2. Copy these model parameter data from the host to the device memory.
    - 2.3. Calculate option prices using these parameters.
    - 2.4. Calculate the target function.
    - 2.5. Deallocate the model parameter data from the device memory.
3. Deallocate the option data from the device memory.

This is much more efficient, since it avoids redundant repeated data copying of option data from the host to the device.

The actual implementation can be found in appendix A. A similar setup to ours but for different models was used in [32].

Chapter 6

Our model

The main driving force behind the introduction of continuous-time stochastic models in finance has been the development of *option pricing models*. An option pricing model serves as an arbitrage-free interpolation and extrapolation tool, it captures the features of option prices quoted on the market and is able to extrapolate the concept of value to instruments not priced on the market. Stochastic models in mathematical finance are also used for hedging and quantifying risk associated with a given position. That is, a stochastic model is calibrated to match the observed market prices, and can then be used to price more complex derivative products not quoted on the market or, compute hedging strategies and assess risks.

So, what exactly constitutes a good stochastic model? Some of the wanted properties of a good model include:

⇒ A realistic reproduction of statistical properties of the time series of the price. For instance, diffusion models, although they are able to capture heavy tail behaviour of the price time series, still model the price as a continuous process, although it is a well-known fact that prices exhibit jumps. This is especially true in the case of electricity, which is known for its highly volatile jumpy dynamic. Neglecting jumps in this case is unrealistic to say the least. Models driven by jump processes, e.g., Lévy processes, do a much better job.

⇒ Ability to fit the market data. A good model should be able to fit the observed market data fairly well at the time of calibration. It does not make much sense to base our future decisions on a model that is unable to even reproduce the current dynamics.

⇒ Efficiency. A good model should allow for computations and calibration within a reasonable time-frame.

⇒ Robustness. Neither a small modification in the data used for calibration nor a small modification of the model parameters should lead to a significant change in behaviour of the model.

For our model, we choose to take an approach similar to the HJM framework and directly model the risk-neutral dynamic of futures prices. Futures prices are the basic brick for hedging and pricing more complex products. A model that is able to perfectly fit the observed forward curve and replicate the market prices of options is a good starting point for valuation of more complex derivatives.

6.1 Motivation

Recall from (3.21) how the instantaneous forward enables us to model futures prices of any granularity simultaneously. If $f(t,T)$ denotes the instantaneous forward rate, i.e., the price at time t for delivery at the instant T, the price of a future at time t with delivery period $[T_1, T_2]$ is given by

$$F_t^{T_1,T_2} = \frac{1}{T_2 - T_1} \int_{T_1}^{T_2} f(t,s)\,ds. \tag{6.1}$$

Consider now a model with the following dynamic of the instantaneous forward rate

$$\frac{df(t,T)}{f(t,T)} = \sigma_1^f(t)\,dW_t^1 + \sigma_2^f(t)\,dW_t^2, \tag{6.2}$$

where (W_1, W_2) is a two-dimensional Brownian motion with correlation coefficient ρ^f. Setting

$$\sigma_1^f(t) = e^{-bt}, \qquad \sigma_2^f(t) = c, \tag{6.3}$$

gives an interpretation to the underlying Brownian motion: W_t^1 models the short-term fluctuations in the instantaneous forward rate that fade away with time, whereas W_t^2 models the long-term fluctuations.

Following the approach of Bjerksund et al. [2008][XXX] for their one factor model, we approximate the futures price dynamic by

$$\frac{dF_t^{T_1,T_2}}{F_t^{T_1,T_2}} \approx \frac{1}{T_2 - T_1} \int_{T_1}^{T_2} \frac{df(t,s)}{f(t,s)}\,ds = \left(\frac{1}{T_2 - T_1} \int_{T_1}^{T_2} \sigma_1^f(t)\,dt \right) dW_t^1 + \left(\frac{1}{T_2 - T_1} \int_{T_1}^{T_2} \sigma_2^f(t)\,dt \right) dW_t^2. \tag{6.4}$$

This motivates a model where the price of a future at time t with delivery period $[T_1, T_2]$ is modelled as

$$\frac{dF_t^{T_1,T_2}}{F_t^{T_1,T_2}} = \sigma_1^{T_1,T_2}(t)\,dW_t^1 + \sigma_2^{T_1,T_2}(t)\,dW_t^2 \tag{6.5}$$

with

$$\begin{aligned} \sigma_1^{T_1,T_2}(t) &= \frac{1}{T_2 - T_1} \int_{T_1}^{T_2} \sigma_1^f(t)\,dt = \frac{a}{T_2 - T_1} \left(e^{-b(T_1 - t)} - e^{-b(T_2 - t)} \right) \\ \sigma_2^{T_1,T_2}(t) &= \frac{1}{T_2 - T_1} \int_{T_1}^{T_2} \sigma_2^f(t)\,dt = c, \end{aligned} \tag{6.6}$$

and where (W_1, W_2) is a two-dimensional Brownian motion with correlation coefficient ρ. The SDE then admits the following solution:

$$\begin{aligned} F_t^{T_1,T_2} = F_0^{T_1,T_2} \exp \Bigg(& \int_0^t \sigma_1^{T_1,T_2}(s)\,dW_s^1 + \int_0^t \sigma_2^{T_1,T_2}(s)\,dW_s^2 \\ & - \frac{1}{2} \int_0^t \left(\sigma_1^{T_1,T_2}(s)^2 + 2\rho \sigma_1^{T_1,T_2}(s) \sigma_2^{T_1,T_2}(s) + \sigma_2^{T_1,T_2}(s)^2 \right) ds \Bigg). \end{aligned} \tag{6.7}$$

For some point in time t and a delivery period $[T_1, T_2]$, this model can be reduced to a case of the Black's model, by setting

$$\begin{aligned}
\tilde{F}_t &= \tilde{F}_0 \exp\left(\tilde{\sigma}\tilde{W}_t - \frac{1}{2}\tilde{\sigma}^2\right), \text{ which verifies } \frac{d\tilde{F}_t}{\tilde{F}_t} = \tilde{\sigma} d\tilde{W}_t \\
\tilde{\sigma}^2 &= \int\limits_0^t \left(\sigma_1^{T_1,T_2}(s)^2 + 2\rho\sigma_1^{T_1,T_2}(s)\sigma_2^{T_1,T_2}(s) + \sigma_2^{T_1,T_2}(s)^2\right) ds \\
\tilde{F}_0 &= F_0^{T_1,T_2} \exp\left(-\frac{1}{2}\int\limits_0^t \left(\sigma_1^{T_1,T_2}(s)^2 + 2\rho\sigma_1^{T_1,T_2}(s)\sigma_2^{T_1,T_2}(s) + \sigma_2^{T_1,T_2}(s)^2\right) ds\right),
\end{aligned} \tag{6.8}$$

with a Brownian motion $\tilde{W}$ independent of anything else. Therefore, vanilla option prices, just like any other whose closed form is known for Black's model, can be computed analytically. This also implies that the implied volatility is constant across strikes for a combination of time t and a delivery period $[T_1, T_2]$.

The goal of our model is to generalize this expression by replacing the two-dimensional Brownian motion by a suitable two-dimensional Lévy martingale.

6.2 The Lévy driven model

We start by replacing the two-dimensional Brownian motion in (6.5) by a two-dimensional Lévy process and keeping the volatility functions from (6.6):

$$\frac{dF_t^{T_1,T_2}}{F_t^{T_1,T_2}} = \sigma_1^{T_1,T_2}(t)\, dL_t^1 + \sigma_2^{T_1,T_2}(t)\, dL_t^2. \tag{6.9}$$

with a Lévy process L^t of the following form

$$\mathrm{L}_t = \begin{pmatrix} L_t^1 \\ L_t^2 \end{pmatrix} = \begin{pmatrix} b_1 t + W_t^1 + \sum\limits_{n=0}^{N_t^1} Y_t^1 \\ b_2 t + W_t^2 + \sum\limits_{n=0}^{N_t^2} Y_t^2 \end{pmatrix}, \tag{6.10}$$

where $b_1, b_2 \in \mathbb{R}$, (W_t^1, W_t^2) is a two-dimensional standard Brownian motion with correlation coefficient ρ, and $\sum\limits_{n=0}^{N_t^i} Y_t^i$ for $i = 1, 2$ are independent compound Poisson processes with jump intensities λ_i, and the cumulative distribution functions of jump sizes Y^i are given by F_i, respectively.

Note that L_t is a martingale with respect to its natural filtration $\mathcal{F}_t = \sigma(L_s : s \leq t)$ if and only if $b_i = -\lambda_i \mathbb{E}^{\mathbb{Q}}\left[Y^i\right]$ for $i \in \{1, 2\}$. Note also that for the price process in our model (6.17) the choice of b_i is irrelevant since it cancels out anyway.

Proof. L_t is integrable by definition. By looking at:

$$\mathbb{E}\left[L_t \mid \mathcal{F}_t\right] = \mathbb{E}\left[\begin{pmatrix} L_t^1 \\ L_t^2 \end{pmatrix} \mid \mathcal{F}_t\right] = \mathbb{E}\left[\begin{pmatrix} b_1 t + W_t^1 + \sum\limits_{n=0}^{N_t^1} Y_t^1 \\ b_2 t + W_t^2 + \sum\limits_{n=0}^{N_t^2} Y_t^2 \end{pmatrix} \mid \mathcal{F}_t\right] = \begin{pmatrix} b_1 t + \lambda_1 \mathbb{E}\left[Y^1\right] \\ b_2 t + \lambda_2 \mathbb{E}\left[Y^2\right] \end{pmatrix}, \tag{6.11}$$

we see that equality holds if and only if $b_i = -\lambda_i \mathbb{E}^{\mathbb{Q}}\left[Y^i\right]$. ☐

Therefore, in our model, we set $b_i = -\lambda_i \mathbb{E}^{\mathbb{Q}}\left[Y^i\right]$ for $i \in \{1, 2\}$.

Denote by X_t the stochastic process

$$X_t^{T_1,T_2} = \int_0^t \sigma_1^{T_1,T_2}(s)dL_s^1 + \int_0^t \sigma_2^{T_1,T_2}(s)dL_s^2, \tag{6.12}$$

which is not a Lévy process but a member of the more general class of additive processes, as noted in result 1.49.

It follows from result 1.43 and result 1.50 that the characteristic function of X_t is given by:

$$\Phi_{X_t^{T_1,T_2}}(w) = \mathbb{E}^{\mathbb{Q}}\left[e^{iwX_t}\right] = \exp\left(\int_0^t \psi_s^{T_1,T_2}(w)\, ds\right) \tag{6.13}$$

with $\psi_s(w)$ equal to:

$$\begin{aligned}
\psi_s^{T_1,T_2}(w) = \exp\Bigg(&- \frac{w^2}{2}\left(\sigma_1^{T_1,T_2}(s)^2 + 2\rho\sigma_1^{T_1,T_2}(s)\sigma_2^{T_1,T_2}(s) + \sigma_2^{T_1,T_2}(s)^2\right) \\
&+ \lambda_1 \int_{\mathbb{R}} \left(e^{iw\sigma_1^{T_1,T_2}(s)y_1} - 1 - iw\sigma_1^{T_1,T_2}(s)y_1\right) F_1(dy_1) \\
&+ \lambda_2 \int_{\mathbb{R}} \left(e^{iw\sigma_2^{T_1,T_2}(s)y_2} - 1 - iw\sigma_2^{T_1,T_2}(s)y_2\right) F_1(dy_2)\Bigg) \\
= \exp\Bigg(&- \frac{w^2}{2}\left(\sigma_1^{T_1,T_2}(s)^2 + 2\rho\sigma_1^{T_1,T_2}(s)\sigma_2^{T_1,T_2}(s) + \sigma_2^{T_1,T_2}(s)^2\right) \\
&+ \lambda_1 \left(\mathbb{E}\left[e^{iw\sigma_1^{T_1,T_2}(s)Y^1}\right] - 1 - iw\sigma_1^{T_1,T_2}(s)\mathbb{E}\left[Y^1\right]\right) \\
&+ \lambda_2 \left(\mathbb{E}\left[e^{iw\sigma_2^{T_1,T_2}(s)Y^2}\right] - 1 - iw\sigma_2^{T_1,T_2}(s)\mathbb{E}\left[Y^2\right]\right)\Bigg) \\
= \exp\Bigg(&- \frac{w^2}{2}\left(\sigma_1^{T_1,T_2}(s)^2 + 2\rho\sigma_1^{T_1,T_2}(s)\sigma_2^{T_1,T_2}(s) + \sigma_2^{T_1,T_2}(s)^2\right) \\
&+ \lambda_1 \left(\Phi_{Y^1}\left(w\sigma_1^{T_1,T_2}(s)\right) - 1 - iw\sigma_1^{T_1,T_2}(s)\mathbb{E}\left[Y^1\right]\right) \\
&+ \lambda_2 \left(\Phi_{Y^2}\left(w\sigma_2^{T_1,T_2}(s)\right) - 1 - iw\sigma_2^{T_1,T_2}(s)\mathbb{E}\left[Y^2\right]\right)\Bigg)
\end{aligned} \tag{6.14}$$

where Φ_{Y^1} and Φ_{Y^2} are characteristic functions of Y^1 and Y^2, respectively.

In case the moment generating function of $X_t^{T_1,T_2}$ exists for some $u \in \mathbb{R}$, it is given by the following relation:

$$M_{X_t^{T_1,T_2}}(u) = \mathbb{E}\left[e^{uX_t^{T_1,T_2}}\right] = \Phi_{X_t^{T_1,T_2}}(-iu). \tag{6.15}$$

Provided that $\mathbb{E}\left[e^{X_t^{T_1,T_2}}\right] = M_{X_t^{T_1,T_2}}(1) < \infty$ for all $t > 0$, and since $X_t^{T_1,T_2}$ is a process with independent increments, it follows from result 1.46 that:

$$\frac{e^{X_t^{T_1,T_2}}}{\mathbb{E}\left[e^{X_t^{T_1,T_2}}\right]} = \exp\left(X_t^{T_1,T_2} - \int_0^t \psi_s^{T_1,T_2}(-iw)\, ds\right) \tag{6.16}$$

is a true martingale.

Moreover, a simple application of the Ito-lemma shows that the process

$$F_t^{T_1,T_2} = F_0^{T_1,T_2} \exp\left(X_t^{T_1,T_2} - \int_0^t \psi_s^{T_1,T_2}(-iw)\, ds \right) \tag{6.17}$$

solves the initial SDE (6.9) of our model with starting value $F_0^{T_1,T_2}$.

Finally, the characteristic function of the log price process $\ln(F_t)$ is given by:

$$\Phi_{\ln(F_t^{T_1,T_2})}(w) = \exp\left[iw \left(X_0 - \int_0^t \psi_s^{T_1,T_2}(-i)\, ds \right) + \int_0^t \psi_s^{T_1,T_2}(w)\, ds \right], \tag{6.18}$$

with $X_0 = \ln(F_0)$.

For the distribution of the jump sizes, we choose:

$$\begin{aligned} Y^1 &\sim \mathcal{N}(\mu, \sigma) \\ -Y^2 &\sim \operatorname{Exp}(\alpha), \end{aligned} \tag{6.19}$$

that is, in (6.14) we use

$$\begin{aligned} \Phi_{Y^1}(w) &= \exp\left(i\mu w - \frac{1}{2}\sigma^2 w^2 \right), \\ \Phi_{Y^2}(w) &= \frac{\alpha}{\alpha + iw}. \end{aligned} \tag{6.20}$$

Similary to the case from (6.5), in our model L_t^1 is responsible for the short-term and L_t^2 for the long-term fluctuations in the forward curve. It is a little peculiar that we choose L_t^2 to only exhibit negative jumps, but, as we will see in the section with results, this configuration seems to work really well.

The new model is a significant improvement over the model (6.5) from the introductory section, and as a matter of fact an improvement over any other HJM-style model. First, from a qualitative point of view, the forward price is righteously modelled as a discontinuous process which is much more realistic. Secondly, we get rid of the false log-normality property of returns. Thirdly, we expect a better fit of the implied volatility curve, since it is no longer constant across a range of strikes. Our setting also allows for relatively simple option pricing, as discussed in the following section. Finally, the fact that we are using an additive process as the stochastic our log-price gets rid of the time-homogeneity we would encounter had we simply only used an exponential Lévy process, therefore limiting our ability to fit the volatility curve of different maturities simultaneously.

6.3 Option pricing

For option pricing within our model, we rely on the COS method described in chapter 4, which we further speed up by doing the computations on the graphics processing card with the help of CUDA technology, as described in chapter 5.

Prerequisites for using the COS pricing method is the knowledge of the characteristic function of the log-price and the payoff series coefficients of the derivative one is trying to price, which are given in (4.12). For vanilla calls and puts, the payoff series coefficients are known analytically. The characteristic function of the log-price in our setting is also known and given by (6.18).

For pricing of more complex derivatives, one can always compute the payoff series coefficients numerically. However for some analytical expressions exist, e.g., the payoff series coefficients for gap and digital options are discussed and closed form solutions are stated in [1].

6.4 Calibration

For a given date, we calibrate our model to the set of all European vanilla options quoted on that date. These include options on forwards with different maturities and delivery periods, so that our model captures the dynamic across different maturities and delivery periods.

We attempt to minimize the sum of the weighted quadratic differences between the implied volatilities predicted by the model and those observed in the market:

$$\sum_i \left(w_i \left(\sigma_i^{observed} - \sigma_i^{model}\right)\right)^2 \tag{6.21}$$

In order to find this least-squares fit, we rely on the differential evolution algorithm described below.

6.4.1 Differential evolution optimization

Differential evolution (DE), originally developed by Storn and Price in [33], is an optimization method that attempts to find an optimal solution to a problem by iteratively trying to improve a candidate solution with respect to a given measure of fit. It allows for optimization of multidimensional real-valued functions. In contrast to classic optimization methods such as gradient descent and quasi-newton methods, it does not require the gradient of the function to be known. Thus, the differential evolution method can be used to optimize problems that are not even continuous, are noisy, etc.

A basic variant of the differential evolution algorithm works by having a population of candidate solutions (called agents) in the search-space. New candidates agents are then created by a combination of the positions of existing agents from the population using simple mathematical formulae. If the new position of an agent is an improvement it is accepted and kept as a part of the population, otherwise it is simply discarded. This process is repeated for a number of steps in the hope, but without any guarantees, that a satisfactory solution will eventually be found. Essentially, the function that is getting optimized is treated as a black box that measures the quality of a candidate solution.

Let $f : \mathbb{R}^d \to \mathbb{R}$ be the function that we are trying to minimize (or maximize). The function takes a candidate solution as argument in the form of a vector of real numbers and outputs a real number that measures the fit of the given candidate solution. The gradient of f is unknown. The goal is to find a solution m for which $f(m) \leq f(p)$ for all p in the search-space, meaning that m is the global minimum. Maximization is performed by considering the function $-f$ instead.

Denote by $x \in \mathbb{R}^d$ a candidate solution (agent) in the population. The basic DE algorithm has the following outline:

1. Initialize all agents $\mathbf{x}$ with random positions in the search-space.
2. Repeat the following until a until a termination criterion is met, for instance, maximum number of iterations or wanted level of fit have been reached. For each agent $\mathbf{x}$ in the population:
    - 2.1. Select three agents $\mathbf{a}$, $\mathbf{b}$ and $\mathbf{c}$ that are distinct from each other and $\mathbf{x}$ from the population at random.
    - 2.2. Select a random index $R \in \{1, \ldots, d\}$.
    - 2.3. Compute the new potential position of the agent $\mathbf{y} = (y_1, \ldots, y_d)$:
        - i. For each $i \in \{1, \ldots, d\}$, generate a uniformly distributed number $r_i \sim U(0, 1)$.
        - ii. If $r_i < CR$ or $i = R$ then set $y_i = a_i + F \times (b_i - c_i)$, else set $y_i = x_i$
        - iii. If $f(y) < f(x)$ then replace $\mathbf{x}$ with $\mathbf{y}$ in the population, i.e., replace the existing agent in the population with the improved candidate solution.
3. Select agent $\mathbf{x}$ with lowest $f(\mathbf{x})$ from the population, i.e., the best candidate solution.

The parameter $F \in [0,2]$ is called the *differential weight* and $CR \in [0,1]$ is called the *crossover probability*. The choice of these parameters is a research topic on its own. For instance, some rules of thumb can be found in [34].

In the implementation, we used the R package `DEoptim` (`https://cran.r-project.org/web/packages/DEoptim/index.html`).

6.5 Results

To illustrate what the model is capable of, we present the calibration results for German Power for two different dates: 2016-04-01 and 2015-09-25. Whereas the forward prices are available on EEX, the option data was provided by Axpo Trading AG. Since the exact option data is confidential, only plots of the volatility curve fit are provided instead of the full data used for the calibration.

	2016-04-01	**2015-09-25**
$\boldsymbol{a}$	0,3314650	0.49956363
$\boldsymbol{b}$	0,1112192	0.32277525
$\boldsymbol{c}$	0,2231404	0.36629823
$\boldsymbol{\rho}$	-0,8871685	-0.97815862
$\boldsymbol{\lambda_1}$	0,4680769	0.35046356
$\boldsymbol{\mu}$	0,2157817	0.04918823
$\boldsymbol{\sigma}$	0,5132891	0.42787168
$\boldsymbol{\lambda_2}$	2,3448109	1.78733243
$\boldsymbol{\alpha}$	154.18	184.55
RMSE	0,01518	0,01073
Avg. rel. error	5,02 %	3,98 %

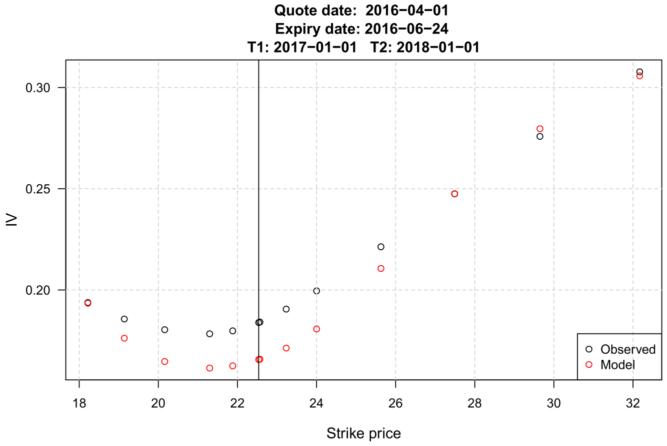
Table 6.1: Calibrated parameters and error measures for the two calibration dates 2016-04-01 and 2015-09-25.

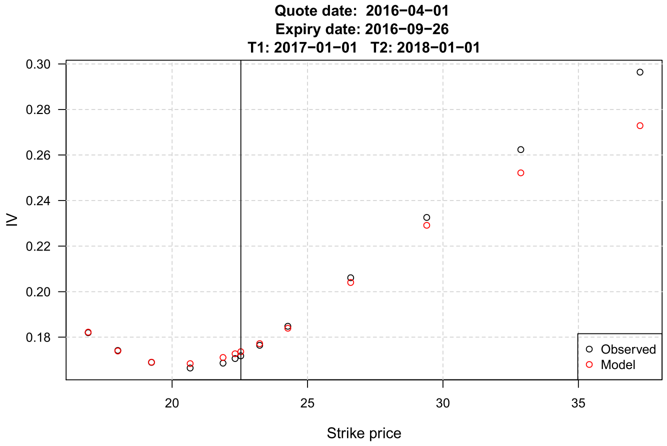
As we can see from the volatility curves below, the model is able to fit the observed implied volatilities fairly well. In both cases, it reaches an average relative error of under 5.1%. Estimated parameters can be found in table 6.1.

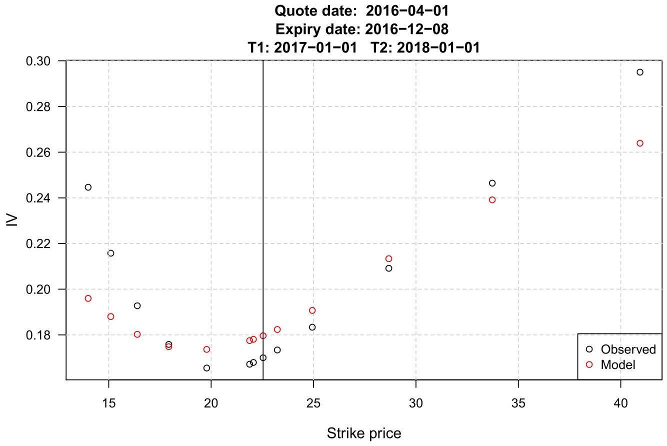
We also examine the performance of our pricing approach. We compare our pricing approach, a serial implementation of the COS formula in R, and a closed-formula that calculates the option price in the setting given by (6.5) (without any jumps). The third one is used merely as a reference value for an "instant" result that one would have if a closed form solution was available. We measure how long it takes the different approaches to calculate the prices for around 70 options (data from 2016-04-01), five times in a row. As we can see in table 6.2, our model is around 24 times faster than its serial counterpart, and around 27 times slower than the theoretical analytical reference value. This is a significant improvement! Note however, that we ran this on a relatively old GPU, the Nvidia GT540M, that has 96 CUDA cores. The difference would have been much larger had we used a newer GPU. For the sake of comparison, current-generation GPUs commonly have over a thousand CUDA cores.

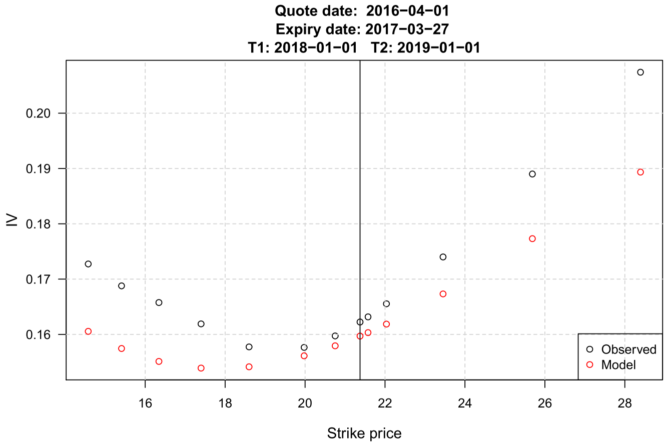
	Replications	**Elapsed**	**Average time**	**Relative time**	
Analytical formula (BS style)	5	0,028	0,0056	1,000	0,0362
COS formula in R (serial)	5	18,760	3,752	670,000	1,000
COS formula using the GPU	5	0,773	0,1546	27,607	24,269

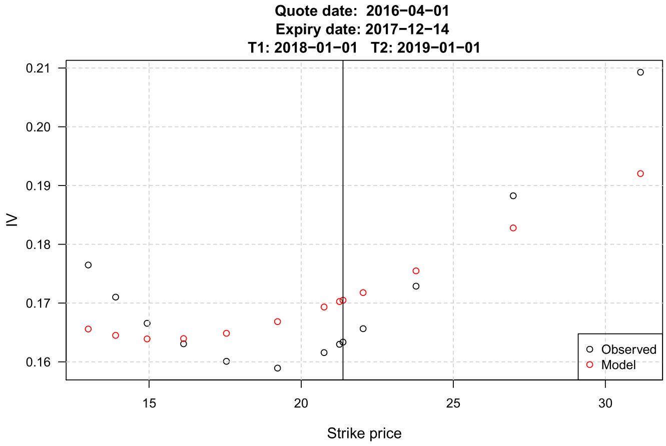
Table 6.2: Speed comparison of different pricing methods. Configuration used: i7 2630QM 2.0 GHz with an Nvidia GT540M. Option pricing of options on 2016-04-01 was performed.

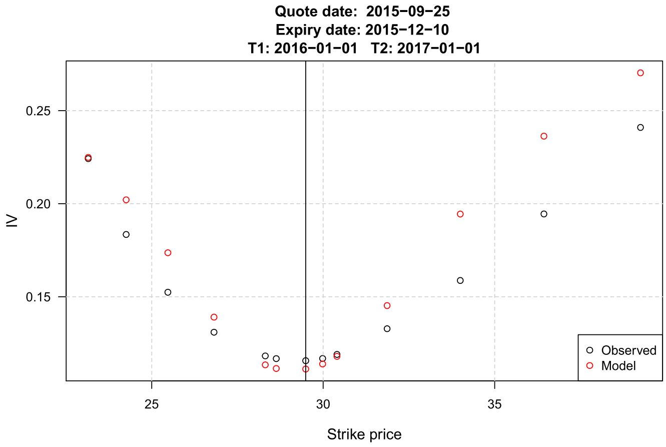

Quote date: 2016−04−01
Expiry date: 2016−06−24
T1: 2017−01−01 T2: 2018−01−01
IV
0.30
0.25
0.20
18
20
22
24
26
28
30
32
Strike price
Observed
Model

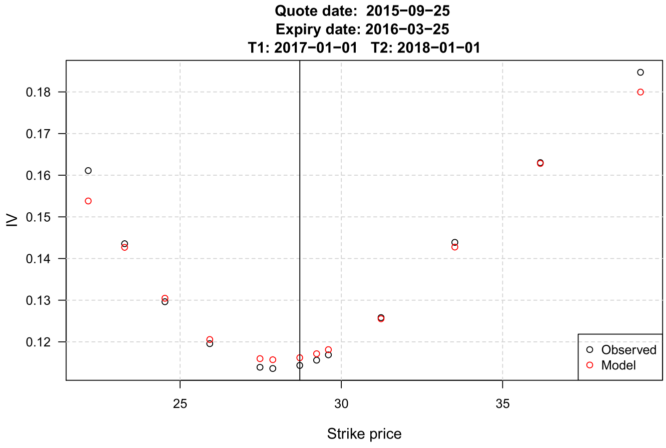

Quote date: 2016−04−01
Expiry date: 2016−09−26
T1: 2017−01−01 T2: 2018−01−01
IV
0.30
0.28
0.26
0.24
0.22
0.20
0.18
20
25
30
35
Strike price
Observed
Model

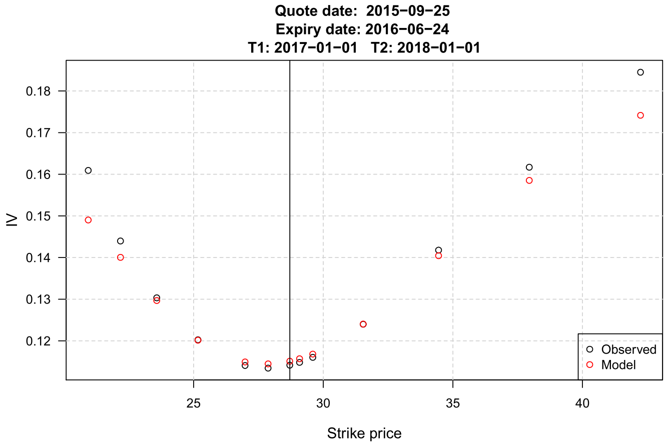

Quote date: 2016−04−01
Expiry date: 2016−12−08
T1: 2017−01−01 T2: 2018−01−01
IV
0.30
0.28
0.26
0.24
0.22
0.20
0.18
15
20
25
30
35
40
Strike price
Observed
Model

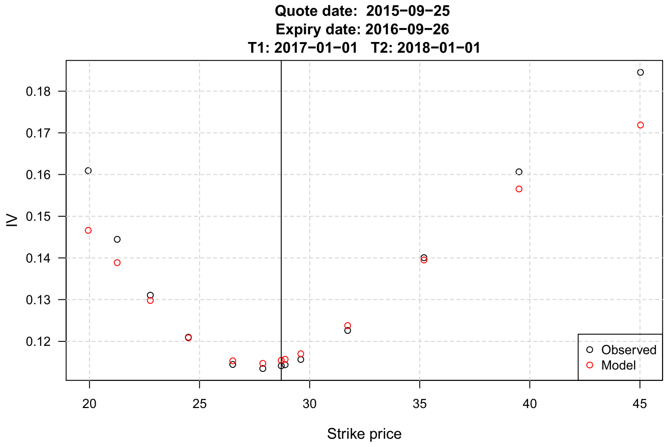

Quote date: 2016−04−01
Expiry date: 2017−03−27
T1: 2018−01−01 T2: 2019−01−01
IV
0.20
0.19
0.18
0.17
0.16
16
18
20
22
24
26
28
Strike price
Observed
Model

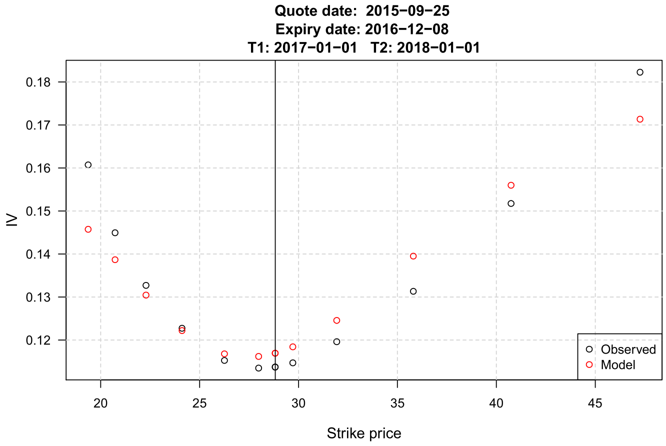

Quote date: 2016−04−01
Expiry date: 2017−12−14
T1: 2018−01−01 T2: 2019−01−01
IV
0.21
0.20
0.19
0.18
0.17
0.16
15
20
25
30
Strike price
Observed
Model


Quote date: 2015−09−25
Expiry date: 2015−12−10
T1: 2016−01−01 T2: 2017−01−01
IV
0.25
0.20
0.15
25
30
35
Strike price
Observed
Model


Quote date: 2015−09−25
Expiry date: 2016−03−25
T1: 2017−01−01 T2: 2018−01−01


Quote date: 2015−09−25
Expiry date: 2016−06−24
T1: 2017−01−01 T2: 2018−01−01


Quote date: 2015−09−25
Expiry date: 2016−09−26
T1: 2017−01−01 T2: 2018−01−01



6.6 Conclusion

We have shown that a generalization of (6.5) that uses a two-dimensional Lévy martingale instead of a two-dimensional Brownian motion leads to an interesting forward price model that inherits some of the good properties from its "predecessor" and improves in other regards. The model allows for simultaneous modelling of arbitrary delivery period granularities. As an HJM-style model, it is able to perfectly fit the observed forward prices. Unfortunately, due to the introduction of the jump component we lose the closed form solutions for vanilla and other option prices. However, the COS method offers a great alternative, and in combination with parallel computing using the GPU we significantly reduce the performance hit that one would take from employing a numerical procedure the traditional way. Qualitatively the presence of jumps increases the credibility of the model and enables easier interpretation of the parameters that lead to heavy tails of log-returns in the forward price. Perhaps most importantly, the model is also able to fit the observed implied volatilities fairly well, meaning that it reflects the expectations in the market, which encourages us in using it as an option valuation and risk quantification tool.

We have also seen how parallel computing using GPUs can lead to a significant performance boost and improvement of the usability of these kinds of models. For situations where efficiency is of special importance, an investment into a CUDA capable GPU may very well be worth it.

Finally, in this setting even more complex derivatives could be priced relatively efficiently, provided that we derive the needed payoff series coefficients (4.12).

6.7 Further steps

In this last section of the thesis, we briefly mention further topics that naturally arise from our described setting. The foundation for some of them has already been laid in the implementation, but unfortunately due to time constraints they have not had the chance to become a part of this thesis.

One could reconsider the choice of the volatility functions, perhaps some other choice would do a better job? Maybe one should also consider a three- or an even four-dimensional underlying Lévy process instead? Would this lead to better results? Regarding calibration, maybe it would make sense to fit the model to correlations between log-returns of the different delivery periods to better capture the ρ parameter dynamic? Another thing one could do is vary the distributions of the jump sizes. Possibly there is a jump size configuration that works even better.

Another interesting point would be to find other fields in electricity pricing where parallel processing might result in a significant increase in performance. Pricing of path-dependent options might be a good candidate.

Finally, motivated by the joint-model described in section 3.3.4, in the next subsection we briefly discuss how one could extend our model to jointly model two markets.

6.7.1 Two-market model proposal

Extending our framework to cover two different markets (commodities) A and B, the forward prices would then admit the following dynamics:

$$
\begin{aligned}
F_t^{A;T_1,T_2} &= F_0^{A;T1,T2} \exp\left(X_t^{A;T_1,T_2} - \int_0^t \psi^{A;T1,T2}(s)ds \right) \\
F_t^{B;T_1,T_2} &= F_0^{B;T1,T2} \exp\left(X_t^{B;T_1,T_2} - \int_0^t \psi^{B;T1,T2}(s)ds \right),
\end{aligned}
\tag{6.22}
$$

with

$$
\begin{aligned}
X_t^{A;T_1,T_2} &= \int_0^t \sigma_1^{A;T_1,T_2}(s)dL_s^{A;2} + \int_0^t \sigma_2^{A;T_1,T_2}(s)dL_s^{A;2} \\
X_t^{B;T_1,T_2} &= \int_0^t \sigma_1^{B;T_1,T_2}(s)dL_s^{B;2} + \int_0^t \sigma_2^{B;T_1,T_2}(s)dL_s^{B;2}
\end{aligned}
\tag{6.23}
$$

and

$$
\begin{aligned}
\text{Ł}_t^A &= \begin{pmatrix} L_t^{A,1} \\ L_t^{A,2} \end{pmatrix} = \begin{pmatrix} b_1^A t + W_t^{A,1} + \sum\limits_{n=0}^{N_t^{A,1}} Y_t^{A,1} \\ b_2^A t + W_t^{A,2} + \sum\limits_{n=0}^{N_t^{A,2}} Y_t^{A,2} \end{pmatrix} \\
\text{Ł}_t^B &= \begin{pmatrix} L_t^{B,1} \\ L_t^{B,2} \end{pmatrix} = \begin{pmatrix} b_1^B t + W_t^{B,1} + \sum\limits_{n=0}^{N_t^{B,1}} Y_t^{B,1} \\ b_2^B t + W_t^{B,2} + \sum\limits_{n=0}^{N_t^{B,2}} Y_t^{B,2} \end{pmatrix}.
\end{aligned}
\tag{6.24}
$$

Now, the simplest way to introduce dependence between the two is by correlating the Brownian motions. This results in a 4 × 4 correlation matrix.

$$
\begin{array}{c} \\ W^{A,1} \\ W^{A,2} \\ W^{B,1} \\ W^{B,2} \end{array}
\begin{array}{c}
\begin{array}{cccc} W^{A,1} & W^{A,2} & W^{B,1} & W^{B,2} \end{array} \\
\begin{pmatrix}
1 & \rho_A & \rho_{A1,B1} & \rho_{A1,B2} \\
\rho_A & 1 & \rho_{A2,B1} & \rho_{A2,B2} \\
\rho_{A1,B1} & \rho_{A2,B1} & 1 & \rho_B \\
\rho_{A1,B2} & \rho_{A2,B2} & \rho_B & 1
\end{pmatrix}
\end{array}.
\tag{6.25}
$$

However, since commodities commonly jump at around the same time, we would like to extend the model by making it possible for both commodities to jump at the same time. For instance, it is easy to imagine that a jump in gas prices would have an effect on electricity prices due to increased production costs. One possibility is to extend the underlying Lévy processes by adding a compound Poisson process that jumps at the same time:

$$\begin{aligned}
Ł_t^A &= \begin{pmatrix} L_t^{A,1} \\ L_t^{A,2} \end{pmatrix} = \begin{pmatrix} b_1^A t + W_t^{A,1} + \sum_{n=0}^{N_t^{A,1}} Y_t^{A,1} + \sum_{n=0}^{N_t^{C,3}} Y_t^{A,3} \\ b_2^A t + W_t^{A,2} + \sum_{n=0}^{N_t^{A,2}} Y_t^{A,2} + \sum_{n=0}^{N_t^{C,4}} Y_t^{A,4} \end{pmatrix} \\
Ł_t^B &= \begin{pmatrix} L_t^{B,1} \\ L_t^{B,2} \end{pmatrix} = \begin{pmatrix} b_1^B t + W_t^{B,1} + \sum_{n=0}^{N_t^{B,1}} Y_t^{B,1} + \sum_{n=0}^{N_t^{C,3}} Y_t^{B,3} \\ b_2^B t + W_t^{B,2} + \sum_{n=0}^{N_t^{B,2}} Y_t^{B,2} + \sum_{n=0}^{N_t^{C,4}} Y_t^{B,4} \end{pmatrix},
\end{aligned} \tag{6.26}$$

where Poisson processes $N_t^{C,3}$, $N_t^{C,4}$ model the occurrence of common price shocks that affect both commodities. Then $Y_t^{A,3}$, $Y_t^{A,4}$, $Y_t^{B,3}$ and $Y_t^{B,4}$ are the sizes of these shocks. In other words, these compound pairs of Poisson processes

$$\left(\sum_{n=0}^{N_t^{C,3}} Y_t^{A,3}, \sum_{n=0}^{N_t^{C,4}} Y_t^{A,4} \right) \tag{6.27}$$

$$\left(\sum_{n=0}^{N_t^{C,3}} Y_t^{B,3}, \sum_{n=0}^{N_t^{C,4}} Y_t^{A,4} \right) \tag{6.28}$$

jump together at the same time but have different jump sizes.

The next thing one could discuss is the dependence between the sizes of the simultaneous common jumps. Some of the possibilities include:

1. Independence. Although the jumps happen at the same time, assume no dependence between their sizes. However, this seems a little naive. It is easy to imagine a scenario where, because of the assumption of independence one gets positive jumps in commodity A and negative jumps in commodity B, whereas they both should have moved the same way.
2. Same jump size for both commodities, $Y_t^{A,3} = Y_t^{B,3}$, $Y_t^{B,4} = Y_t^{B,4}$. Is this realistic? Although the X_t process is "normalized", it may be unrealistic to expect the exact same amount of relative shock size in both commodities.
3. A deterministic transformation of the jump size, preferably linear. The simplest example would be $Y_t^{A,3} = kY_t^{B,3}$, $Y_t^{B,4} = kY_t^{B,4}$. Then a price shock in commodity A would transfer to commodity B in some percentage.
4. Multivariate normal distribution. Then the margins are normal and one can easily tweak the linear correlation between the two. This way we could of our choice and different magnitudes of jumps across commodities.
5. Copulas. Then jump sizes of commodities A and B can have their respective marginal distributions and we are free to choose a copula for the sake of modelling their dependence. However, this makes the matter unnecessarily much more complicated and we suggest against it. Using a copula means dealing with rank correlations, and this would lead to various filtering procedures.

In contrast to the single market model, it no longer suffices to calibrate each market simply to the quoted option data. We need a product or a way that captures the dependence dynamic. Perhaps a spread or the correlation of the log-returns of the forward prices should be used? A lot of interesting new questions arise.

Appendix A

Implementation and source code

A development of an efficient pricing library for our model has been a central goal of this thesis. Here we provide the source code and a short description of the implementation. The core functions are developed in (CUDA) C and then compiled into a library meant for use with the programming language R. A starting point that gave us a package skeleton was [32].

The implementation supports pricing of processes with the following dynamic:

$$\tilde{F}_t^{T_1,T_2} = \tilde{F}_0^{T_1,T_2} \exp\left(\tilde{X}_t^{T_1,T_2} - \int\limits_0^t \tilde{\psi}_s^{T_1,T_2}(-iw)\, ds \right)$$

with

$$\tilde{X}_t^{T_1,T_2} = \int\limits_0^t \sigma_1^{T_1,T_2}(s) d\tilde{L}_s^1 + \int\limits_0^t \sigma_2^{T_1,T_2}(s) d\tilde{L}_s^2,$$

and

$$\tilde{L}_t = \begin{pmatrix} \tilde{L}_t^1 \\ \tilde{L}_t^2 \end{pmatrix} = \begin{pmatrix} b_1 t + W_t^1 + \sum\limits_{n=0}^{N_t^1} Y_t^1 + \sum\limits_{n=0}^{N_t^3} Y_t^3 \\ b_2 t + W_t^2 + \sum\limits_{n=0}^{N_t^2} Y_t^2 + \sum\limits_{n=0}^{N_t^4} Y_t^4 \end{pmatrix}.$$

where $\tilde{\psi}_s^{T_1,T_2}$ is defined analogously to (6.14) in relation to the characteristic function of $\tilde{X}_t^{T_1,T_2}$. The b_i are once again chosen to make the Lévy process $\tilde{L}_t$ a martingale. Each of the Poisson processes is characterized by its intensity λ_i.

For the jump sizes Y_t^n one is free to choose one of these distributions:

1. $Y_t^n \sim \mathcal{N}(\mu, \sigma)$.
2. $Y_t^n \sim \text{Uniform}(a, b)$.
3. $Y_t^n \sim \text{Exp}(\alpha)$.
4. $-Y_t^n \sim \text{Exp}(\alpha)$.

For pricing, a vector of parameters is passed as an argument to the function. The form of this vector of parameters is shown in table A.1 (in the order from top-left to bottom right). The argument `distr1` selects the distribution of Y^1 as a number from $\{1, 2, 3, 4\}$ according to the list above. The parameters of this distribution are then passed with `distr1_p1` and `distr1_p2`. If the distribution only requires one argument, one should set `distr1_p1=distr1_p2`. Analogously for `distr2`, `distr3`, `distr4`. The rest of the parameters are self-explanatory.

`a`	**`lambda1`**	**`lambda2`**	**`lambda3`**	**`lambda4`**
`b`	**`distr1`**	**`distr2`**	**`distr3`**	**`distr4`**
`c`	**`distr1_p1`**	**`distr2_p1`**	**`distr3_p1`**	**`distr4_p1`**
`rho`	**`distr1_p2`**	**`distr2_p2`**	**`distr3_p2`**	**`distr4_p2`**

Table A.1: Parameter vector members passed to the implementation.

The implementation already lays the foundation for option pricing in the proposed two-market setting described in section 6.7.1.

For pricing within our model, we simply set $\lambda_3 = \lambda_4 = 0$ and use the appropriate parameters for the jump distributions, i.e., `distr1`=1 and `distr2`=4.

After loading the library in R, the functions listed in table A.2 are available.

`Copy_Data`	Copies the option data to the GPU memory.
`Dealloc_Data`	Removes the copied option data from the GPU memory.
`Error_Function`	Calculates the RMSE or average relative error for some parameters.
`Set_Block_Size`	Sets the block size, i.e., the number of summands in the COS formula.
`Compute_Option_Prices`	Computes option prices (IVs) for given option data and parameters.

Table A.2: Functions at user's disposal.

When calibrating the model, `Error_Function` is meant to be used in combination with `Copy_Data` and `Dealloc_Data`. Since the option data does not change during the calibration, we load it at the beginning and then unload it after we are done. This avoids redundant memory copying that we would have had we used the more general function `Compute_Option_Prices`.

The following external libraries are used in the implementation:

- ⇒ **Let's be rational**: a library for efficient calculations of the Black-Scholes implied volatility, based on the paper by Peter Jäckel [35]. Available at: `https://pypi.python.org/pypi/lets_be_rational/`.
- ⇒ **cuda_complex**: an implementation of `std::complex` for use on CUDA devices. Allows us to write cleaner code using overloaded operators. Available at: `https://github.com/jtravs/cuda_complex/blob/master/cuda_complex.hpp`.

Table A.3 contains the source code structure description.

`cuda_pricing.cu`	the core file where all the core functions and GPU kernels are defined
`cuda_pricing.h`	simply the header file of `cuda_pricing.cu`
`gpuCalib.c`	a higher-level file that defines R-friendly functions
`gpuCalib.r`	the .R file providing an interface with the compiled library

Table A.3: Source code structure

Listing A.1: cuda_pricing.cu

```
#define _USE_MATH_DEFINES
#include <stdio.h>
#include <stdlib.h>

#include <cuda_runtime_api.h>
#include <cuda.h>

#include "cuda_pricing.h"
#include "lets_be_rational.h"
#include "cuda_complex.hpp"
```

```
// number of blocks = number of options
int num_blocks = 0;
// number of threads per block, i.e., number of terms in the COS formula
int block_size = 128;

int num_bytes    = 0;

// bool variable keep track of whether the data has been loaded
bool bLoad              = false;

// variables stored in the host memory
int  *h_option_type;
double *h_F0, *h_K, *h_tau, *h_T1, *h_T2, *h_W, *h_observed;

// variables stored on the device memory
int  *d_option_type;
double *d_F0, *d_K, *d_tau, *d_T1, *d_T2, *d_results;
// parameter array
double *d_p0;

// pi
#define pi 3.14159265358979323846264338327950

// positions of parameters in the paramater array
#define N_PARAMETERS  20
#define a_             0
#define b_             1
#define c_             2
#define rho_           3
#define lambda1_       4
#define distr1_        5
#define distr1_p1_     6
#define distr1_p2_     7
#define lambda2_       8
#define distr2_        9
#define distr2_p1_    10
#define distr2_p2_    11
#define lambda3_      12
#define distr3_       13
#define distr3_p1_    14
#define distr3_p2_    15
#define lambda4_      16
#define distr4_       17
#define distr4_p1_    18
#define distr4_p2_    19

/********************************** COS method related fu
    ***********************************/

// ************ Characteristic functions of the possible jump size distros ************
     //
__device__ complex<double> CF_Normal(complex<double> w, int D, double mu, double sigma)
{
        complex<double> I(0., 1.);
        complex<double> ret;

        if (D == 0) {
                ret = exp(I*w*mu - 0.5*sigma*sigma*w*w);
        }
        else if (D == 1) {
                ret = I*mu;
        }
        else if (D == 2) {
                ret = -mu*mu - sigma*sigma;
        }
        else if (D == 4) {
                ret = pow(mu, 4.) + 6.*mu*mu*sigma*sigma + 3.*pow(sigma, 4.);
        }
        else if (D == 6) {
                ret = -pow(mu, 6.) - 15.*pow(mu, 4.)*pow(sigma, 2.) - 45.*pow(mu, 2.)*
```

```
                     pow(sigma, 4.) - 15.*pow(sigma, 6.);
        }

        return ( ret );
}

__device__ complex<double> CF_Exp(complex<double> w, int D, double alpha)
{
        complex<double> I(0., 1.);
        complex<double> ret;

        if (D == 0) {
                ret = alpha / (alpha - I*w);
        }
        else if (D == 1) {
                ret = I / alpha;
        }
        else if (D == 2) {
                ret = -2. / (alpha*alpha);
        }
        else if (D == 4) {
                ret = 24. / pow(alpha, 4.);
        }
        else if (D == 6) {
                ret = -720. / pow(alpha, 6.);
        }

        return ( ret );
}

__device__ complex<double> CF_NegExp(complex<double> w, int D, double alpha)
{
        complex<double> I(0., 1.);
        complex<double> ret;

        if (D == 0) {
                ret = alpha / (alpha + I*w);
        }
        else if (D == 1) {
                ret = -I / alpha;
        }
        else if (D == 2) {
                ret = -2. / (alpha*alpha);
        }
        else if (D == 4) {
                ret = 24. / pow(alpha, 4.);
        }
        else if (D == 6) {
                ret = -720. / pow(alpha, 6.);
        }

        return ( ret );
}

__device__ complex<double> CF_Unif(complex<double> w, int D, double a, double b)
{
        complex<double> I(0., 1.);
        complex<double> ret;

        if (D == 0) {
                ret = (exp(-I*w*b) - exp(-I*w*a)) / (I*w*(b - a));
        }
        else if (D == 1) {
                ret = 0.5 * I * (a + b);
        }
        else if (D == 2) {
                ret = 1 / 3. * (a*a + a*b + b*b);
        }
        else if (D == 4) {
                ret = (-pow(a, 5.) + pow(b, 5.)) / (5 * (a - b));
        }
        else if (D == 6) {
                ret = (-pow(a, 7.) + pow(b, 7.)) / (7 * (a - b));
        }

        return ( ret );
}


#define NORMAL_DISTR    1
#define UNIFORM_DISTR   2
#define EXP_DISTR       3
#define NEGEXP_DISTR    4

__device__ complex<double> CF_JumpSize( complex<double> w, int D, double distr_d, double p1, double p2 )
{
        int distr = (int)distr_d;
        complex<double> ret;

        if (distr == NORMAL_DISTR) {
                ret = CF_Normal(w, D, p1, p2);
        }
        else if (distr == UNIFORM_DISTR) {
                ret = CF_Unif(w, D, p1, p2);
        }
        else if (distr == EXP_DISTR) {
                ret = CF_Exp(w, D, p1);
        }
        else if (distr == NEGEXP_DISTR) {
                ret = CF_NegExp(w, D, p1);
        }       else {
                ret = 0.;
                printf("Error: Unknown Jump Size Distribution.\n");
        }

        return ( ret );
}

// ************ Volatility functions ************  //
__device__ double sigma1(double t, double T1, double T2, double *p0)
{
        //return p0[a_] / (p0[b_] * (T2 - T1))*(exp(-p0[b_] * (T1) ) - exp(-p0[b_] * (T2) ));
        return p0[a_] / (p0[b_] * (T2 - T1))*(exp(-p0[b_] * (T1-t)) - exp(-p0[b_] * (T2-t)));
}

__device__ double sigma2(double t, double T1, double T2, double *p0)
{
        return p0[c_];
}

// ************ psi function used in the CF of the log-price  ************  //
__device__ complex<double> CF_psi( complex<double> w, double s, double T1, double T2, double *p0 )
{
        complex<double> I(0., 1.);
        complex<double> ret;

        complex<double> J1, J2, J3, J4;

        double s1 = sigma1(s, T1, T2, p0), s2 = sigma2(s, T1, T2, p0);

        J1 = CF_JumpSize(w*s1, 0, p0[distr1_], p0[distr1_p1_], p0[distr1_p2_]) - 1. - w*s1*CF_JumpSize(0, 1, p0[distr1_], p0[distr1_p1_], p0[distr1_p2_]);
        J2 = CF_JumpSize(w*s2, 0, p0[distr2_], p0[distr2_p1_], p0[distr2_p2_]) - 1. - w*s2*CF_JumpSize(0, 1, p0[distr2_], p0[distr2_p1_], p0[distr2_p2_]);
        J3 = CF_JumpSize(w*s1, 0, p0[distr3_], p0[distr3_p1_], p0[distr3_p2_]) - 1. - w*s1*CF_JumpSize(0, 1, p0[distr3_], p0[distr3_p1_], p0[distr3_p2_]);
        J4 = CF_JumpSize(w*s2, 0, p0[distr4_], p0[distr4_p1_], p0[distr4_p2_]) - 1. - w*
            s2*CF_JumpSize(0, 1, p0[distr4_], p0[distr4_p1_], p0[distr4_p2_]);

        ret = -0.5 * w*w * (s1*s1 + 2.*p0[rho_]*s1*s2 + s2*s2) + p0[lambda1_] * J1 + p0[
            lambda2_] * J2 + p0[lambda3_] * J3 + p0[lambda4_] * J4;

        return ret;
}

// ************ simple numerical integral of psi over [0,t] ************  //
__device__ complex<double> CF_psi_int(complex<double> w, double t, double T1, double T2,
     double *p0)
{
        const int n = 100;
        double step_size = t / n;

        complex<double> integral = 0.0;
        for (double s = 0; s <= t; s += step_size)
                integral += step_size * CF_psi(w, s, T1, T2, p0);

        return integral;
}

// ************ CF of the log-price  ************  //
__device__ complex<double> CF(double w, double X0, double t, double T1, double T2,
    double *p0)
{
        complex<double> I(0., 1.);
        complex<double> ret;

        //Phi_t(w) = exp(I * w * (X0 - integrate_0^t(CF_psi(-I, T1, T2, p0))) +
            integrate_0^t(CF_psi(w, T1, T2, p0)) )
        ret = exp( I * w * (X0 - CF_psi_int(-I, t, T1, T2, p0)) + CF_psi_int(w, t, T1,
            T2, p0) );

        return ret;
}

// ************ Calculation of the integral truncation limits for the COS formula
    ************  //
__device__ void calc_truncation_limits(double *a, double *b, double X0, double t, double
     T1, double T2, double *p0)
{
        complex<double> I(0., 1.);

        complex<double> Cf1d1 = CF_JumpSize(0., 1, p0[distr1_], p0[distr1_p1_], p0[
            distr1_p2_]);
        complex<double> Cf2d1 = CF_JumpSize(0., 1, p0[distr2_], p0[distr2_p1_], p0[
            distr2_p2_]);
        complex<double> Cf3d1 = CF_JumpSize(0., 1, p0[distr3_], p0[distr3_p1_], p0[
            distr3_p2_]);
        complex<double> Cf4d1 = CF_JumpSize(0., 1, p0[distr4_], p0[distr4_p1_], p0[
            distr4_p2_]);

        complex<double> Cf1d2 = CF_JumpSize(0., 2, p0[distr1_], p0[distr1_p1_], p0[
            distr1_p2_]);
        complex<double> Cf2d2 = CF_JumpSize(0., 2, p0[distr2_], p0[distr2_p1_], p0[
            distr2_p2_]);
        complex<double> Cf3d2 = CF_JumpSize(0., 2, p0[distr3_], p0[distr3_p1_], p0[
            distr3_p2_]);
        complex<double> Cf4d2 = CF_JumpSize(0., 2, p0[distr4_], p0[distr4_p1_], p0[
            distr4_p2_]);

        complex<double> Cf1d4 = CF_JumpSize(0., 4, p0[distr1_], p0[distr1_p1_], p0[
            distr1_p2_]);
        complex<double> Cf2d4 = CF_JumpSize(0., 4, p0[distr2_], p0[distr2_p1_], p0[
            distr2_p2_]);
        complex<double> Cf3d4 = CF_JumpSize(0., 4, p0[distr3_], p0[distr3_p1_], p0[
            distr3_p2_]);
        complex<double> Cf4d4 = CF_JumpSize(0., 4, p0[distr4_], p0[distr4_p1_], p0[
            distr4_p2_]);

        complex<double> Cf1d6 = CF_JumpSize(0., 6, p0[distr1_], p0[distr1_p1_], p0[
            distr1_p2_]);
        complex<double> Cf2d6 = CF_JumpSize(0., 6, p0[distr2_], p0[distr2_p1_], p0[
            distr2_p2_]);
        complex<double> Cf3d6 = CF_JumpSize(0., 6, p0[distr3_], p0[distr3_p1_], p0[
            distr3_p2_]);
        complex<double> Cf4d6 = CF_JumpSize(0., 6, p0[distr4_], p0[distr4_p1_], p0[
            distr4_p2_]);

        double lambda1 = p0[lambda1_], lambda2 = p0[lambda2_], lambda3 = p0[lambda3_],
            lambda4 = p0[lambda4_];

        // variables for simple trapezoid integration on [0,t]
        const int n = 100;
        double step_size = t/n;
        complex<double> integral;

        // ------------------------------- c1 -------------------------------
        // c1 = Real( X0 - integrate( 0.5*(s1*s1 + 2.*p0[rho_]*s1*s2 + s2*s2) + lambda1
            * (-1. + Cf1 + I*s1*Cf1d1) + lambda2 * (-1. + Cf2 + I*s2*Cf2d1) + lambda3 *
            (-1. + Cf3 + I*s1*Cf3d1) + lambda4 * (-1. + Cf4 + I*s2*Cf4d1) )

        integral = 0.0;
        for (double s = 0; s <= t; s += step_size)
        {
                double s1 = sigma1(s, T1, T2, p0), s2 = sigma2(s, T1, T2, p0);

                complex<double> Cf1 = CF_JumpSize(-I*s1, 0, p0[distr1_], p0[distr1_p1_],
                     p0[distr1_p2_]);
                complex<double> Cf2 = CF_JumpSize(-I*s2, 0, p0[distr2_], p0[distr2_p1_],
                     p0[distr2_p2_]);
                complex<double> Cf3 = CF_JumpSize(-I*s1, 0, p0[distr3_], p0[distr3_p1_],
                     p0[distr3_p2_]);
                complex<double> Cf4 = CF_JumpSize(-I*s2, 0, p0[distr4_], p0[distr4_p1_],
                     p0[distr4_p2_]);

                integral += step_size * ( 0.5*(s1*s1 + 2.*p0[rho_]*s1*s2 + s2*s2) +
                    lambda1 * (-1. + Cf1 + I*s1*Cf1d1) + lambda2 * (-1. + Cf2 + I*s2*
                    Cf2d1) + lambda3 * (-1. + Cf3 + I*s1*Cf3d1)        + lambda4 * (-1.
                    + Cf4 + I*s2*Cf4d1) ) ;
        }

        double c1 = real(X0 - integral);

        // ------------------------------- c2 --------------------------------
        // c2 = -integrate(-s1*s1 - 2.*p0[rho_]*s1*s2 - s2*s2 + lambda1 * s1*s1 * real(
            Cf1d2) + lambda2 * s2*s2 * real(Cf2d2) + lambda3 * s1*s1 * real(Cf3d2) +
            lambda4 * s2*s2 * real(Cf4d2));

        integral = 0.0;
        for (double s = 0; s <= t; s += step_size)
        {
                double s1 = sigma1(s, T1, T2, p0), s2 = sigma2(s, T1, T2, p0);
                integral += step_size * ( -s1*s1 - 2.*p0[rho_]*s1*s2 - s2*s2 + lambda1 *
                     s1*s1 * real(Cf1d2) + lambda2 * s2*s2 * real(Cf2d2) + lambda3 * s1*
                    s1 * real(Cf3d2) + lambda4 * s2*s2 * real(Cf4d2) );
        }
        double c2 = real(-integral);

        // ------------------------------- c4 ---------------------------------
        // c4 = integrate ( lambda1 * pow(s1, 4.) * real(Cf1d4) + lambda2 * pow(s2, 4.)
            * real(Cf2d4) + lambda3 * pow(s1, 4.) * real(Cf3d4) + lambda4 * pow(s2, 4.)
            * real(Cf4d4));

        integral = 0.0;
        for (double s = 0; s <= t; s += step_size)
        {
                double s1 = sigma1(s, T1, T2, p0), s2 = sigma2(s, T1, T2, p0);
```

```
                integral += step_size * (  lambda1 * pow(s1, 4.) * real(Cf1d4) + lambda2
                     * pow(s2, 4.) * real(Cf2d4) + lambda3 * pow(s1, 4.) * real(Cf3d4) +
                     lambda4 * pow(s2, 4.) * real(Cf4d4) );
        }
        double c4 = real(integral);

        // -------------------------------- c6 ---------------------------------
        // c6 =  -integrate( lambda1 * pow(s1, 6.) * real(Cf1d6) + lambda2 * pow(s2, 6.)
             * real(Cf2d6) + lambda3 * pow(s1, 6.) * real(Cf3d6) + lambda4 * pow(s2, 6.)
             * real(Cf4d6));

        integral = 0.0;
        for (double s = 0; s <= t; s += step_size)
        {
                double s1 = sigma1(s, T1, T2, p0), s2 = sigma2(s, T1, T2, p0);
                integral += step_size * ( lambda1 * pow(s1, 6.) * real(Cf1d6) + lambda2
                    * pow(s2, 6.) * real(Cf2d6) + lambda3 * pow(s1, 6.) * real(Cf3d6) +
                    lambda4 * pow(s2, 6.) * real(Cf4d6) );
        }
        double c6 = real(-integral);


        double L = 10.;

        *a = c1 - L * sqrt(c2 + sqrt(c4 + sqrt(c6)));
        *b = c1 + L * sqrt(c2 + sqrt(c4 + sqrt(c6)));
}

// xi function used in the COS method
__device__ double xi(double k,  double a,       double b,       double c,       double d
    )
{
  double ret = 1.0/(1.0+pow(k*pi/(b-a),2))*(cos(k*pi*(d-a)/(b-a))*exp(d)-cos(k*pi*(c-a)
      /(b-a))*exp(c)+k*pi/(b-a)*sin(k*pi*(d-a)/(b-a))*exp(d)-k*pi/(b-a)*sin(k*pi*(c-a)/(
      b-a))*exp(c));

  return ( ret );
}

// psi function used in the COS method
__device__ double psi(double k, double a, double b, double c, double d)
{
  double ret = 0.0;

  if ( k == 0 )
    ret = d-c;
  else
    ret = (sin(k*pi*(d-a)/(b-a))-sin(k*pi*(c-a)/(b-a)))*(b-a)/(k*pi);

  return ( ret );
}

// ************ Kernel for the calculation of option prices in parallel ************  //
__global__ void option_price( int *option_type, double *F0, double *K, double *tau,
    double *T1, double *T2, double *p0, double* results )
{
        complex<double> I(0., 1.);
        extern __shared__ double sdata[]; // shared data of the block, contains the
            summands, "extern" keyword tells the program it is the shared data

        // thread tx handles the summand with index k=tx
        int tx = threadIdx.x;
        int bx = blockIdx.x;

        // normalized log start value
        double x = log(F0[bx] / K[bx]);

        // calculate truncation limits and store them in a, b
        double a, b;
        calc_truncation_limits(&a, &b, x, tau[bx], T1[bx], T2[bx], p0);
        double U = 2.0 / (b - a)*(-xi(tx, a, b, a, 0) + psi(tx, a, b, a, 0));

        if (tx == 0) U *= 0.5; // 0th summand is weighted with 1/2

        complex<double> phi = CF(double(tx)*pi/(b - a), 0, tau[bx], T1[bx], T2[bx], p0);

        // calculate the (tx)-th summand

        // phi * exp( i*k*pi*(x-a)/(b-a) )*U_k
        sdata[tx] = real(phi * exp( I*double(tx)*pi*(x-a)/(b-a) ) * U); // as in (30) in
             COS.pdf

        // wait until all threads have filled in sdata[]
        __syncthreads();

        // reduction sum them up
        for (int offset = blockDim.x / 2; offset > 0; offset >>= 1)
        {
                if (tx < offset) sdata[tx] += sdata[tx + offset];
                __syncthreads();
        }

        // 0th thread contains the whole sum and returns the result for the current
            block
        if (tx == 0) {
                results[bx] = K[bx] * sdata[0];

                // default output is the put price, in case of a call it needs to get
                    adjusted
                if (option_type[bx] == 1) // use the put-call parity to get the call
                    price
                        results[bx] += F0[bx] - K[bx];
        }
}


/**************************** Copy Data, Error Function, Compute Option prices...
    **********************/

#ifdef __cplusplus
  extern "C"
#endif

// copies data to the GPU global memory
void copy_data(int *option_type, double *F0, double *K, double *tau, double *T1, double
    *T2, double *W, double *observed, int n)
{
        h_option_type = option_type; h_F0 = F0; h_K = K; h_tau = tau; h_T1 = T1; h_T2 =
            T2; h_W = W; h_observed = observed;

        // allocate space on global GPU memory for the option data and the results array
        cudaMalloc((void**)&d_option_type, n*sizeof(int));
        cudaMalloc((void**)&d_F0, n*sizeof(double));
        cudaMalloc((void**)&d_K, n*sizeof(double));
        cudaMalloc((void**)&d_tau, n*sizeof(double));
        cudaMalloc((void**)&d_T1, n*sizeof(double));
        cudaMalloc((void**)&d_T2, n*sizeof(double));
        cudaMalloc((void**)&d_results, n*sizeof(double));

        cudaMemcpy(d_option_type, h_option_type, n*sizeof(int), cudaMemcpyHostToDevice);
        cudaMemcpy(d_F0, h_F0, n*sizeof(double), cudaMemcpyHostToDevice);
        cudaMemcpy(d_K, h_K, n*sizeof(double), cudaMemcpyHostToDevice);
        cudaMemcpy(d_tau, h_tau, n*sizeof(double), cudaMemcpyHostToDevice);
        cudaMemcpy(d_T1, h_T1, n*sizeof(double), cudaMemcpyHostToDevice);
        cudaMemcpy(d_T2, h_T2, n*sizeof(double), cudaMemcpyHostToDevice);

        // set the number of blocks to n, each block handles pricing of one option!
        num_blocks = n;
        // inform the rest of the code that te data has been loaded
        printf("Option data sucessfuly copied.\n");
        bLoad = true;
}

#ifdef __cplusplus
  extern "C"
#endif

void dealloc_data (void)
{
  cudaFree(d_option_type);
  cudaFree(d_F0);
  cudaFree(d_K);
  cudaFree(d_tau);
  cudaFree(d_T1);
  cudaFree(d_T2);
  cudaFree(d_results);

  // inform the rest of the code that te data has been removed
  bLoad = false;
        printf("Option data sucessfuly deallocated.\n");
}

#ifdef __cplusplus
  extern "C"
#endif
void set_block_size( int block_size_ )
{
   block_size = block_size_;
}


#ifdef __cplusplus
        extern "C"
#endif

// computes option prices for loaded data
void compute_option_prices(double *p0, int use_IV, double *_out)
{
        // check if data has been loaded
        if (!bLoad)
        {
                printf("Error: call copy_data first to load option data\n");
                return;
        }

        // copy the parameter array to the GPU memory
        cudaMalloc((void**)&d_p0, N_PARAMETERS*sizeof(double));
        cudaMemcpy(d_p0, p0, N_PARAMETERS*sizeof(double), cudaMemcpyHostToDevice);

        // calculate the option prices on the GPU
        option_price<<<num_blocks, block_size, block_size*sizeof(double)>>>(
            d_option_type, d_F0, d_K, d_tau, d_T1, d_T2, d_p0, d_results);

        // copy the results back to _out
        cudaMemcpy(_out, d_results, num_blocks*sizeof(double), cudaMemcpyDeviceToHost);

        // is use_IV == 1, replace option values with their implied volatilities
        if (use_IV)
                for (int index = 0; index < num_blocks; ++index)
                        _out[index] =
                            implied_volatility_from_a_transformed_rational_guess(_out[
                            index], h_F0[index], h_K[index], h_tau[index], h_option_type
                            [index]);

        // clean up
        cudaFree(d_p0);
}
```

```
#ifdef __cplusplus
  extern "C"
#endif

// computes the RMSE or Average Relative Error using option prices or IVs for
// a given a parameter array p0 and loaded option data

// use_RelErr = 1 -> relative error, use_RelErr = 0 -> RMSE
double error_func ( double *p0, int use_IV, int use_RelErr )
{
  // check if data has been loaded
  if ( !bLoad )
  {
    printf("Error: call copy_data first to load option data\n");
    return 0.0;
  }

  double error = 0.0;

  // host variable to store the calculated option prices
  double* h_results = new double[num_blocks];
        compute_option_prices(p0, use_IV, h_results);

  // calculate the error
  for(int index = 0 ; index < num_blocks ; ++index)
  {
    // error += weight[i] * ( model - observed )^2

                // use the relative error or not
                if (use_RelErr) {
                        error += h_W[index] * abs(h_results[index] - h_observed[index])
                            / h_observed[index];
                }       else {
                        error += pow(h_W[index] * (h_results[index] - h_observed[index])
                            , 2);
                }
  }

        error = error/num_blocks;
  if( !use_RelErr ) error = sqrt(error);

  // clean up
  delete [] h_results;
  cudaFree( d_p0 );

  // return the error
  return ( error );
}
```

Listing A.2: cuda_pricing.h

```
#ifndef cuda_pricing_H
#define cuda_pricing_H

#ifdef __cplusplus
extern "C" {
#endif

double error_func(double *p0, int use_IV, int use_RelErr);

void compute_option_prices(double *p0, int use_IV, double *_out);

void copy_data(int *option_type, double *F0, double *K, double *tau, double *T1, double
    *T2, double *W, double *observed, int n);

void dealloc_data(void);

void set_block_size(int block_size_);
#ifdef __cplusplus
}
#endif

#endif
```

Listing A.3: gpuCalib.c

```
#ifdef __cplusplus
extern "C" {
#endif

#include <R.h>
#include <Rinternals.h>
#include <R_ext/Rdynload.h>

#include "cuda_pricing.h"

SEXP Error_Func(SEXP p0, SEXP use_IV, SEXP use_RelErr)
{
  // object for returning the result
  SEXP out;

  double *_p0, _out;
  int _use_IV, _use_RelErr;

  _p0 = REAL(p0);
  _use_IV = INTEGER(use_IV)[0];
        _use_RelErr = INTEGER(use_RelErr)[0];

  PROTECT(out = allocVector(REALSXP,1));
  _out = error_func ( _p0, _use_IV, _use_RelErr );
  REAL(out)[0] = _out;
  UNPROTECT(1);

  return out;
}

SEXP Compute_Option_Prices(SEXP option_type, SEXP F0, SEXP K, SEXP tau, SEXP T1, SEXP T2
    , SEXP p0, SEXP use_IV)
{

        // copy the data to GPU memory
  int *_option_type;
  double *_F0, *_K, *_tau, *_T1, *_T2;

  _option_type = INTEGER(option_type);
  _F0         = REAL(F0);
  _K          = REAL(K);
  _tau        = REAL(tau);
  _T1         = REAL(T1);
  _T2         = REAL(T2);

  R_len_t n = length(F0);

        // don't need {W, observed} since they are not used anywhere
  copy_data(_option_type, _F0, _K, _tau, _T1, _T2, NULL, NULL, n);

  double *_p0;
  int _use_IV;

  _p0 = REAL(p0);
  _use_IV = INTEGER(use_IV)[0];

        // compute the options prices and store them in the array pointer *_out
        double *_out = malloc( sizeof(double)*n );
        compute_option_prices ( _p0, _use_IV, _out );
        dealloc_data();
  // copy the data from _out to the R object, free _out and return out
  SEXP out = PROTECT( allocVector(REALSXP, n) ); // object for returning the result
  for( int index=0; index<n; ++index )
        REAL(out)[index] = _out[index];

        free(_out); // free _out
  UNPROTECT(1);

  return out;
}

SEXP Copy_Data(SEXP option_type, SEXP F0, SEXP K, SEXP tau, SEXP T1, SEXP T2, SEXP W,
    SEXP observed )
{
  int *_option_type;
  double *_F0, *_K, *_tau, *_T1, *_T2, *_W, *_observed;

  _option_type = INTEGER(option_type);
  _F0        = REAL(F0);
  _K         = REAL(K);
  _tau       = REAL(tau);
  _T1        = REAL(T1);
  _T2        = REAL(T2);
  _W         = REAL(W);
  _observed = REAL(observed);

  R_len_t n = length(F0);

  copy_data(_option_type, _F0, _K, _tau, _T1, _T2, _W, _observed, n);
        return R_NilValue;
}

SEXP Dealloc_Data()
{
  dealloc_data();
        return R_NilValue;
}

SEXP Set_Block_Size(SEXP BLKSZ)
{
  int _blk_size;
  _blk_size = INTEGER(BLKSZ)[0];

  set_block_size (_blk_size);
  return R_NilValue;
}

// DLL scaffolding for R

R_CallMethodDef callMethods[] = {
  {"Error_Func", (DL_FUNC)&Error_Func, 3},
        {"Compute_Option_Prices", (DL_FUNC)&Compute_Option_Prices, 8},
  {"Copy_Data", (DL_FUNC)&Copy_Data, 8},
  {"Dealloc_Data", (DL_FUNC)&Dealloc_Data, 0},
  {"Set_Block_Size", (DL_FUNC)&Set_Block_Size, 1},
  {NULL, NULL, 0}
};

void R_init_myLib (DllInfo *info) {
  R_registerRoutines (info, NULL, callMethods, NULL, NULL);
}

#ifdef __cplusplus
} // closing brace for extern "C"
#endif
```

Listing A.4: gpuCalib.R

```
Compute_Option_Prices<-function(option_type, F0, K, tau, T1, T2, p, use_IV)
{
    if (!is.loaded('gpuCalib')) {
       dyn.load('gpuCalib.dll')
    }
    ret<-.Call("Compute_Option_Prices", as.integer(option_type), as.numeric(F0), as.
        numeric(K), as.numeric(tau), as.numeric(T1), as.numeric(T2), as.numeric(p), as.
        integer(use_IV) )
    return (ret)
}

Error_Function<-function(p, use_IV, use_RelErr)
{
    if (!is.loaded('gpuCalib')) {
       dyn.load('gpuCalib.dll')
    }
    RMSE<-.Call( "Error_Func",  as.numeric(p), as.integer(use_IV), as.integer(use_RelErr
        ) )
    return (RMSE)
}

Copy_Data<-function( option_type, F0, K, tau, T1, T2, W, observed )
{
    if (!is.loaded('gpuCalib')) {
       dyn.load('gpuCalib.dll')
    }
    Null <- .Call("Copy_Data", as.integer(option_type), as.numeric(F0), as.numeric(K),
        as.numeric(tau), as.numeric(T1), as.numeric(T2), as.numeric(W), as.numeric(
        observed))
}

Dealloc_Data<-function()
{
    if (!is.loaded('gpuCalib')) {
       dyn.load('gpuCalib.dll')
    }
    Null<-.Call("Dealloc_Data")
}

Set_Block_Size<-function(block_size)
{
    if (!is.loaded('gpuCalib')) {
       dyn.load('gpuCalib.dll')
    }
    Null<-.Call("Set_Block_Size", as.integer(block_size))
}
```

Bibliography

[1] F. Fang and C. W. Oosterlee, "A novel pricing method for european options based on fourier-cosine series expansions," *SIAM J. Sci. Comput.*, vol. 31, pp. 826–848, Nov. 2008.

[2] R. Cont and P. Tankov, *Financial modelling with jump processes.* Chapman & Hall/CRC financial mathematics series, Boca Raton (Fla.), London, New York: Chapman & Hall/CRC, 2004.

[3] P. E. Protter, *Stochastic integration and differential equations.* Applications of mathematics, Berlin, Heidelberg, New York: Springer, 2004.

[4] T. Rheinländer and J. Sexton, *Hedging Derivatives*, vol. 15 of *Advanced Series on Statistical Science and Applied Probability.* World Scientific, May 2011.

[5] K.-i. Sato, *Lévy Processes and Infinitely Divisible Distributions (Cambridge Studies in Advanced Mathematics).* Cambridge University Press, 1st ed., Nov. 1999.

[6] R. Aid, *Electricity Derivatives.* Springer Briefs in Quantitative Finance, Springer International Publishing, 2015.

[7] D. Heath, R. Jarrow, and A. Morton, "Bond Pricing and the Term Structure of Interest Rates: A New Methodology for Contingent Claims Valuation," *Econometrica*, vol. 60, pp. 77–105, January 1992.

[8] K. W. Alexander Eydeland, *Energy and Power Risk Management: New Developments in Modeling, Pricing, and Hedging.* Springer Briefs in Quantitative Finance, Wiley Finance, Feb. 2003.

[9] J. Viehmann, "Risk premiums in the german day-ahead electricity market," *Energy Policy*, vol. 39, no. 1, pp. 386–394, 2011.

[10] D. Frestad, "Common and unique factors influencing daily swap returns in the Nordic electricity market, 1997-2005," *Energy Economics*, vol. 30, pp. 1081–1097, May 2008.

[11] D. Frestad, "Correlations among forward returns in the Nordic electricity market," *Int. J. Theor. Appl. Financ*, vol. 12, no. 5, pp. 589–603, 2009.

[12] S. K. Dennis Frestad, Fred Espen Benth, "Modeling term structure dynamics in the nordic electricity swap market," *The Energy Journal*, vol. 31, no. 2, pp. 53–86, 2010.

[13] S. Koekebakker and F. Ollmar, "Forward curve dynamics in the nordic electricity market," *Managerial Finance*, vol. 31, no. 6, pp. 73–94, 2005.

[14] F. E. Benth and S. Koekebakker, "Stochastic modeling of financial electricity contracts," *Energy Economics*, vol. 30, no. 3, pp. 1116 – 1157, 2008.

[15] R. Kiesel, G. Schindlmayr, and R. H. Börger, "A two-factor model for the electricity forward market," *Quantitative Finance*, vol. 9, no. 3, pp. 279–287, 2009.

[16] G. Benmenzer, E. Gobet, and C. Jérusalem, "Arbitrage free cointegrated models in gas and oil future markets," *ArXiv e-prints*, Dec. 2007.

[17] S. Ohana, “Modeling global and local dependence in a pair of commodity forward curves with an application to the {US} natural gas and heating oil markets,” *Energy Economics*, vol. 32, no. 2, pp. 373 – 388, 2010.

[18] R. F. Engle and C. W. J. Granger, “Co-integration and Error Correction: Representation, Estimation, and Testing,” *Econometrica*, vol. 55, pp. 251–76, March 1987.

[19] J. J. Lucia and E. S. Schwartz, “Electricity prices and power derivatives: Evidence from the nordic power exchange,” *Review of Derivatives Research*, vol. 5, no. 1, pp. 5–50, 2002.

[20] l. Cartea and M. Figueroa, “Pricing in electricity markets: a mean reverting jump diffusion model with seasonality,” finance, EconWPA, 2005.

[21] M. T. Barlow, “A diffusion model for electricity prices,” *Mathematical Finance*, vol. 12, no. 4, pp. 287–298, 2002.

[22] A. Cartea, M. G. Figueroa, and H. Geman, “Modelling Electricity Prices with Forward Looking Capacity Constraints,” Birkbeck Working Papers in Economics and Finance 0802, Birkbeck, Department of Economics, Mathematics & Statistics, Feb. 2008.

[23] R. Carmona and M. Coulon, *A Survey of Commodity Markets and Structural Models for Electricity Prices*, pp. 41–83. New York, NY: Springer New York, 2014.

[24] T. Kanamura and K. Ōhashi, “A structural model for electricity prices with spikes: Measurement of spike risk and optimal policies for hydropower plant operation,” *Energy Economics*, vol. 29, no. 5, pp. 1010 – 1032, 2007.

[25] T. Kanamura, “A supply and demand based volatility model for energy prices,” *Energy Economics*, vol. 31, no. 5, pp. 736–747, 2009.

[26] J.-M. Courtault, Y. Kabanov, B. Bru, P. Crepel, I. Lebon, and A. Le Marchand, “Louis bachelier on the centenary of théorie de la spéculation,” post-print, HAL, 2000.

[27] J. P. Boyd, “Chebyshev and fourier spectral methods,” 1999.

[28] C. M. Bender and S. A. Orszag, *Advanced mathematical methods for scientists and engineers texte imprimé*. International series in pure and applied mathematics, Auckland, Bogota, Paris: McGraw-Hill Book Company, 1978.

[29] J. Sanders and E. Kandrot, *CUDA by Example: An Introduction to General-Purpose GPU Programming*. Addison-Wesley Professional, 1st ed., 2010.

[30] S. Cook, *CUDA Programming: A Developer's Guide to Parallel Computing with GPUs*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1st ed., 2013.

[31] M. LLC, “Cuda c/c++ basics,” 2011.

[32] M. F. Dixon, S. Khan, and M. Zubair, “gpusvcalibration: A R Package for Fast Stochastic Volatility Model Calibration Using GPUs,” *Social Science Research Network Working Paper Series*, Feb. 2014.

[33] R. Storn and K. Price, “Differential evolution; a simple and efficient heuristic for global optimization over continuous spaces,” *J. of Global Optimization*, vol. 11, pp. 341–359, Dec. 1997.

[34] J. Liu and J. Lampinen, “A fuzzy adaptive differential evolution algorithm,” *Soft Computing*, vol. 9, no. 6, pp. 448–462, 2005.

[35] P. Jäckel, “Let's be rational,” *Wilmott*, vol. 2015, no. 75, pp. 40–53, 2015.

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Declaration of originality

The signed declaration of originality is a component of every semester paper, Bachelor's thesis, Master's thesis and any other degree paper undertaken during the course of studies, including the respective electronic versions.

Lecturers may also require a declaration of originality for other written papers compiled for their courses.

I hereby confirm that I am the sole author of the written work here enclosed and that I have compiled it in my own words. Parts excepted are corrections of form and content by the supervisor.

Title of work (in block letters):

Implementation of a Lévy driven electricity forward model

Authored by (in block letters):
For papers written by groups the names of all authors are required.

Name(s):	**First name(s):**
Mihaljević	Srećko

With my signature I confirm that
- I have committed none of the forms of plagiarism described in the 'Citation etiquette' information sheet.
- I have documented all methods, data and processes truthfully.
- I have not manipulated any data.
- I have mentioned all persons who were significant facilitators of the work.

I am aware that the work may be screened electronically for plagiarism.

Place, date	**Signature(s)**
Zürich, 29.08.2016	Srećko Mihaljević

For papers written by groups the names of all authors are required. Their signatures collectively guarantee the entire content of the written paper.