UNIVERSITE CATHOLIQUE DE LOUVAIN

FACULTE DES SCIENCES

ECOLE DE STATISTIQUE, BIOSTATISTIQUE

ET SCIENCES ACTUARIELLES

Stochastic modelling of temperature for weather derivatives.

Mémoire partiellement ou totalement confidentiel

Promoteurs : Donatien Hainaut
Lecteurs : Jérome Barbarin

Mémoire présenté en vue de l'obtention du Master en sciences actuarielles par :
Nicolas Prélat

Juin 2019

Contents

List of Figures

List of Tables

Abstract. This thesis proposes and compares continuous-time stochastic models for temperature with the purpose of pricing weather derivatives. A Lévy driven Ornstein–Uhlenbeck process with time-dependent parameters and a convolution-closed Generalized Hyperbolic distribution is developed and then used to price derivatives on CAT, HDD and CDD indices. The models are implemented in Python and C++ with temperature data from Stockholm, Sweden.

1 Introduction

Weather derivatives are financial instruments whose payoffs are dependent on the value of underlying weather indices that measure weather conditions such as temperatures or rainfall. Although they could be used for speculative purposes, they are generally designed to protect weather-sensitive companies against unfavorable conditions, just like an insurance contract.

However, as insurance contracts aim to cover against devastating damages from rare and extreme events, weather derivatives offer a protection against recurrent unfavorable conditions which could cause fluctuation in the revenue of the company. While the holder of an insurance contract will be compensated based on the amount of the damages suffered, after demonstrating the existence thereof, weather derivatives establish a clear payoff structure based on the value of a standardized index, regardless how (and if!) the holder is affected.

One of the main driver behind the growth of the weather derivatives market is the energy industry. It is easy to see that energy companies see their balance sheets deeply influenced by the weather conditions, as a colder or warmer winter can affect both the prices and quantity sold. Trading weather derivatives has become a way for these companies to hedge their risks. Weather derivatives are fairly recent products, and have been gaining in popularity since their inception in the 90's. With ever increasing interest in climate and the effects of its evolution on the economy and on lives, it is expected that interest in those products will continue growing in the future. It is therefore imperative to develop a coherent framework for the pricing and risk management of such important products.

The market for weather derivatives is an incomplete markets. The underlying is not a tradeable asset. Weather cannot be bought, stored, sold or even valued. As a result, a direct application of the standard derivative pricing theory, based on the no-arbitrage, complete market assumptions and replicating strategy cannot be used.

The first part of this work will be a quick but sufficiently thorough overview of the necessary mathematical tools required in the application of Lévy process. It will be undertaken under the framework of semimartingales with a focus on integration and Itô's formula which will be used throughout this work. Generalized Hyperbolic distributions will be detailed as they will be the building blocks for the Generalized hyperbolic Lévy process used in our models. A digression will be made to normal variance-mean mixtures as a parallel can be drawn with Lévy process built by Brownian subordination. This will also lead to the specification of convolutions-closed Generalized hyperbolic distributions, which will proves useful later on for the simulation of our temperature process.

After a quick review of the literature of existing models, the second aim of this work will be to establish a reasonably comprehensive model for the dynamics of temperature. The improvements brought to former models will be studied as well as the remaining shortcomings. Then, the characteristics of the market for weather derivatives and the problems they cause for the application of generally accepted classical asset pricing theory will be discussed.

Finally, analytical pricing methods will be briefly discussed before Monte-Carlo simulation will be used to price common weather derivatives and the effects of the our improvements in the model will be studied.

2 Lévy processes

2.1 Preliminaries

A stochastic process is a familly $(X_t)_{t\in[0,T]}$ of random variables indexed by time. For each realization of the randomness $\omega \in \Omega$, the trajectory $X_t(\omega)$ is called the sample path of the process. A stochastic process can thus be seen as a function $X : [0,T] \times \Omega \mapsto E$ of both time t and the randomness ω.

A function is $f : [0,T] \to \mathbb{R}^d$ said to be càdlàg if it is right-continuous with existing left limits:

$$\forall t \in [0,T] : \quad f(t_-) = \lim_{\substack{s\to t\\ s<t}} f(s) \quad \text{and} \quad f(t^+) = \lim_{\substack{s\to t\\ s>t}} f(s) = f(t)$$

Càdlàg function can have jumps or discontinuity points denoted by $\Delta f(t) = f(t) - f(t_-)$. The number of jumps on [0, T] can be countably infinite but has to be finite for jump size $\Delta f(t)$ larger than $\varepsilon > 0$.

A filtration on (Ω, $\mathcal{F}$, $\mathbb{P}$) is an increasing family of σ-algebra $(\mathcal{F}_t)_{t\in[0,T]}$ such that:

$$\mathcal{F}_0 \subseteq \mathcal{F}_s \subseteq \mathcal{F}_t \subseteq \mathcal{F} \quad \text{for} \quad 0 \leq s \leq t$$

The stochastic process X_t is said to be nonanticipating or $\mathcal{F}_t$-adapted if $\sigma(X_t) \subset \mathcal{F}_t$ for all $t \geq 0$. That essentially means that the value of X_t is revealed at time t.

The essence of the filtration is to model the information that is becoming available as time passes. Intuitively, as the value of the process X evolves with time, probabilities of occurrence of particular events evolve with it. Instead of changing the probability measure $\mathbb{P}$ with time, $\mathbb{P}$ will be kept fixed but conditioned on the information $\mathcal{F}_t$ available at time t. An event $A \in \mathcal{F}_t$ is an event whose occurrence can be determined based on information available at time t.

2.2 Definition and basic properties

A Lévy process is a càdlàg stochastic process L_t defined on (Ω, $\mathcal{F}$, $\mathbb{P}$) with $L_0 = 0$ that possesses the following properties:

- **Independent increments** : for any increasing sequence of times $t_0 \dots t_n$, the random variables L_{t_0} , $(L_{t_1} - L_{t_0}), \dots,$ $(L_{t_n} - L_{t_{n-1}})$ are independent.
- **Stationary increments** : for all h > 0, t $\geq$ 0, the distribution of $L_{t+h} - L_t$ does not depend on t : $L_t - L_s \overset{d}{=} L_{t-s}$
- **Stochastic continuity** : $\lim_{h\to 0} \mathbb{P}(|L_{t+h} - L_t| \geq \varepsilon) = 0 \qquad$ for all $\varepsilon > 0$, t $\geq$ 0.

2.3 Infinite divisibility and Lévy-Khinchin representation

A distribution function P is infinitely divisible if $\forall k \in \mathbb{Z}^+$, there exist a sequence of i.i.d. random variables $\eta_1, \dots, \eta_k$ such that the sum $\sum_{i=1}^{k} \eta_i$ is also P distributed. A stochastic process L_t with $L_0 = 0$ follows a infinitely divisible distribution if, for any $n \in \mathbb{Z}^+$ there exist a sequence of i.i.d. random variables $\left\{L_{\frac{t}{n}}\right\}_{k=1}^{n}$ such that

$$L_t \overset{d}{=} L_{\frac{t}{n}} + (L_{\frac{2t}{n}} - L_{\frac{t}{n}}) + \cdots + (L_{\frac{tn}{n}} - L_{\frac{t(n-1)}{n}})$$

Lévy-Khinchin formula : If L_1 follows an infinitely divisible distribution μ defined on $\mathbb{R}$, then its characteristic function is of the form

$$\Phi_{L_1}(z) = E\left[\mathrm{e}^{iL_1 z}\right] = \int_{\mathbb{R}} \mathrm{e}^{\mathrm{i}uz}\mu(\mathrm{d}u) = \mathrm{e}^{\Psi(z)}$$

with cumulant generating function

$$\Psi(z) = i\gamma z - \frac{1}{2}\sigma^2 z^2 + \int_{\mathbb{R}} \left(\mathrm{e}^{izx} - 1 - izx\mathbb{1}_{\{|x|\leq 1\}}\right)\nu(\mathrm{d}x)$$

where $\gamma \in \mathbb{R}, \sigma^2 \geqslant 0$ and ν is a measure on $\mathbb{R}\backslash\{0\}$ with $\int_{\mathbb{R}}\left(x^2 \wedge 1\right)\nu(\mathrm{d}x) < +\infty$

If $(L_t)_{t\geq 0}$ is a Lévy process, then it has an infinitely divisible distribution. Moreover, for any infinitely divisible distribution μ, there exists a Lévy process (L_t) such that the distribution of L_1 is μ.

As a result, a Lévy process is therefore entirely defined by the distribution of its first increment L_1 and its characteristic function is of the form

$$\Phi_{L_t}(z) = \mathrm{E}\left[\mathrm{e}^{izL_t}\right] = \mathrm{e}^{t\Psi(z)}$$

with $\Psi(z) = \ln \Phi_{L_1}(z)$ and $z \in \mathbb{R}$.

2.4 Jump and Lévy measure

Define $\Delta L = (\Delta L_t)_{0\leq t\leq T}$ as the jump part of a Lévy process where $\Delta L_t = L_t - L_{t-}$ and $L_{t-} = \lim_{s\to t} L_s$.
The stochastic continuity property of a Lévy process ensure that $\Delta L_t = 0$ almost surely. As a result, Lévy processes have no deterministic times of discontinuity.

Jump measure of a Lévy process: The jump measure of L_t is a Poisson random measure on $\mathcal{B}\left(\mathbb{R}^+ \times (\mathbb{R}\backslash\{0\})\right)$ with intensity $\nu(dx)dt$ defined as follows:

$$\Pi(B) = \#\Big\{t : \Delta L_t \neq 0 \text{ and } (t, \Delta L_t) \in B\Big\}$$

We will have sets B of the form $[t_1, t_2] \times A$ such that $\Pi([t_1, t_2], A)$ counts the number of jumps of L between t_1 and t_2 such that their sizes fall into set A.

Lévy measure of a Lévy process: The Lévy measure ν of a Lévy process L_t gives the expected number of jumps of L_t per unit time whose size are in a set $A \in \mathcal{B}\left(\mathbb{R}\backslash\{0\}\right)$:

$$\nu(A) = E\Big[\#\Big\{t \in [0,1] : \Delta L_t \neq 0 \text{ and } \Delta L_t \in A\Big\}\Big] = E\Big[\Pi([0,1], A)\Big]$$

The Lévy measure is defined on $\mathbb{R}\backslash\{0\}$ and satisfies $\int_{\mathbb{R}}(x^2 \wedge 1)\,\nu(\mathrm{d}x) < +\infty$

Lévy density of a Lévy process: If the Lévy measure is absolutely continuous with respect to the Lebesgue measure, the Lévy density is defined by

$$k(x) = \frac{\nu(\mathrm{d}x)}{\mathrm{d}x} \text{ such that } \nu(A) = \int_A \nu(\mathrm{d}x) = \int_A k(x)\mathrm{d}x$$

2.5 Lévy-Itô decomposition

The Lévy-Itô decomposition assert that if L_t is a Lévy process with jump measure Π and Lévy measure ν, it can be decomposed into a Brownian motion with drift and a pure jump process.

$$L_t = \gamma t + \sigma W_t + \int_0^t \int_{|x|<1} x\tilde{\Pi}(\mathrm{d}s, \mathrm{d}x) + \int_0^t \int_{|x|\geq 1} x\Pi(\mathrm{d}s, \mathrm{d}x)$$

with $\tilde{\Pi}(\mathrm{d}s, \mathrm{d}x) = \Pi(\mathrm{d}s, \mathrm{d}x) - \nu(\mathrm{d}x)\mathrm{d}s$

It also affirms that the distribution of a Lévy process is entirely determined by a scalar γ, a scalar σ and a positive measure ν. The triplet (σ, ν, γ) is called characteristic triplet or Lévy triplet of the Lévy process L_t.

2.6 Characteristic function and moments of a Lévy process

The moments of a Lévy process can be derived using the following properties of the characteristic function

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

$$\mathrm{E}\left[X^k\right] = \frac{1}{i^k}\frac{\partial^k \Phi_X}{\partial z^k}(0) \quad \forall k = 1, \ldots, n.$$

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

$$\mathrm{E}\left[X^k\right] = \frac{1}{i^k}\frac{\partial^k \Phi_X}{\partial z^k}(0) \quad \forall k = 1, \ldots, 2n.$$

Specifically, for a Lévy process with triplet (σ, ν, γ)

- $\mathrm{E}\left[X_t\right] = t\left(\gamma + \int_{|x|\geq 1} x\nu(\mathrm{d}x)\right)$
- $\mathrm{Var}\left[X_t\right] = t\left(\sigma + \int_{\mathbb{R}} x^2\nu(\mathrm{d}x)\right)$

2.7 Pathwise properties

Finite variation: A Lévy process is of finite variation if and only if its Lévy triplet (Σ, ν, γ) satisfies :

$$\sigma = 0 \quad \text{and} \quad \int_{\mathbb{R}} (|x| \wedge 1)\,\nu(\mathrm{d}x) < +\infty$$

Consequently, a process of finite variation has no Brownian components. If $\sigma \neq 0$ or $\int_{\mathbb{R}} (|x| \wedge 1)\, \nu(\mathrm{d}x) = +\infty$, then the Lévy process is of infinite variation.

Finite activity: A Lévy process is of finite activity if it has a finite number of jump on any interval : $\nu(\mathbb{R}) < \infty$.

In this case, the path of L will have a finite number of jumps on every interval. Moreover, we have a non-zero probability that no jump will occur on some time interval. In contrast, an infinite activity Lévy process with $\nu(\mathbb{R}) = \infty$ will have an infinite number of jump on any time interval.

As a result, we cannot assume that the sum of jumps of a Lévy process necessarily converge. It is possible that

$$\int_0^t \int_{\mathbb{R}} x\Pi(\mathrm{d}s, \mathrm{d}x) = \sum_{s \leq \mathrm{t}} |\Delta L_s| = \infty$$

which is the reason this integral is not used directly in the Lévy-Itô decomposition. The sum of small jump has to be compensated to obtain convergence. However, we will always have that

$$\sum_{s \leq t} |\Delta L_s|^2 < \infty$$

as the Lévy measure satisfies by definition $\int_{\mathbb{R}} (x^2 \wedge 1)\, \nu(\mathrm{d}x) < +\infty$ which ensure that Lévy processes have finite quadratic variation, a required condition to be a semi-martingale.

2.8 Subordination of Lévy processes

Subordinator: A subordinator is Lévy process Z_t with almost surely nondecreasing sample paths : if $s \leq t$ then $Z_s \leq Z_t$ a.s.

Z_t and its Lévy triplet (σ, ν, γ) satisfies the following properties :

- Z_t has no Brownian component : $\sigma = 0$
- Z_t has only positive jump of finite variation : $\nu(\mathbb{R}^-) = 0$ and $\int_{|x| \leq 1} |x| \nu(dx) < \infty$
- Z_t has a positive drift $\gamma - \int_{|x| \leq 1} x\nu(dx) \geq 0$

Moment generating function of a subordinator: Let Z_t be a subordinator with Lévy triplet $(0, \rho, b)$ and Laplace transform

$$\mathcal{L}_{Z_t}(u) = \mathrm{E}\left[e^{-uZ_t}\right] = e^{-tL(u)}$$

with Laplace exponent L(u) of the form $L(u) = bu + \displaystyle\int_0^\infty (e^{ux} - 1)\, \rho(dx)$

Subordination of a Lévy process: Let X_t be a Lévy process with Lévy triplet (σ, ν, γ) and cumulant generating function $\psi(u)$. Let Z_t be a subordinator with Lévy triplet $(0, \rho, b)$ and Laplace exponent L(u). Let both of these processes be defined on the probability space $(\Omega,\, \mathcal{F},\, \mathbb{P})$. Then can create a process

$$Y(t, \omega) = X(Z(t,\, \omega),\, \omega) \qquad \forall\, \omega \in \Omega$$

by subordination. The goal is to make a time deformation: instead of having time passing by at a constant speed, it is now elapsing at a stochastic speed that depends on another Lévy process : the subordinator. The resulting process Y_t is still a Lévy process. Its characteristic function can be obtained by composition of the Laplace exponent of Z with the characteristic exponent of X :

$$\Phi_{Y_t}(u) = \mathbb{E}\left[e^{iuY_t}\right] = \mathcal{L}_{Z_t}(-\psi(u)) = e^{tL(\psi_X(u))}$$

The triplet $(\sigma^Z, \nu^Z, \gamma^Z)$ of the time-changed process $Y_t = X_{Z_t}$ is given by

$$\begin{aligned} \sigma^Z &= b\sigma \\ \nu^Z(B) &= b\nu(B) + \int_0^\infty F_s^X(B)\rho(ds) \\ \gamma^Z &= b\gamma + \int_0^\infty \rho(ds) \int_{|x|\leq 1} xF_s^X(dx) \end{aligned}$$

where F_t^X is the probability distribution of X_t

2.9 Complements on stochastic processes

Martingale: A martingale with regard to a filtration $\mathcal{F}_t$ is a process M_t such that

- M_t is $\mathcal{F}_t$-adapted
- $\mathrm{E}\left[|M_t|\right] < \infty$ for any t $\in [0, T]$.
- $\mathrm{E}\left[M_s|\mathcal{F}_t\right] = M_t$ for all t < s.

Stopping Time: A non-negative random variable τ with respect to filtration of $\mathcal{F}_t$ is called a stopping time $\{\tau \leq t\} \in \mathcal{F}_t$. This means that the information contained in $\mathcal{F}_t$ allows us to determine whether the event $\{\tau \leq t\}$ has already occurred.

Localization: A property of a stochastic process X_t is said to hold locally if there exists a sequence of stopping times τ_n such that $\tau_n \to \infty$ as $n \to \infty$ such that for each n, the property hold for the stopped processes $X\left(t \wedge \tau_n\right)$.

Local martingale: A process for which the martingale property hold localy. Any local martingale M has a unique decomposition

$$M = M^c + M^d$$

where M^c is a continuous local martingale and M^d a purely discontinuous local martingale.

Semimartingale: A semimartingale is a process S_t that can be decomposed into the sum of a local martingale M_t and a process of finite variation A_t, with $M_0 = A_0 = 0$ such that

$$S_t = S_0 + M_t + A_t$$

Semimartingale form the largest class of processes with respect to which the Itô integral can be defined. It is important to note that the decomposition above is not necessarily unique unlike the decomposition for local martingale. We use the notation S^{cm} to denote the continuous martingale component of a semimartingale S. From the Lévy-Itô decomposition, it follows that every Lévy processes is a semimartingale with the decomposition

$$A_t = \gamma t + \int_0^t \int_{|x|\geq 1} x\Pi(\mathrm{d}s, \mathrm{d}x) \quad \text{and} \quad M_t = \sigma W_t + \int_0^t \int_{|x|<1} x\tilde{\Pi}(\mathrm{d}s, \mathrm{d}x)$$

Quadratic variation: The quadratic variation of a stochastic process X_t is defined as

$$[X, X]_t = \lim_{\delta_n \to 0} \sum_{k=1}^{n} \left(X_{t_k} - X_{t_{k-1}} \right)^2$$

with $\delta_n = \max\left(t_{i+1}^n - t_i^n\right) \to 0$ for partitions $0 = t_0 < t_1 < \ldots < t_n = t$ of $[0, t]$.

If X_t is a semimartingale, then $[X, X]_t$ exists and is a non-decreasing adapted process of finite variation. If X_t is a square integrable martingale (if $\mathrm{E}[X_t^2] < \infty$), then its quadratic variation process $[X, X]_t$ exists and $X_t^2 - [X, X]_t$ is a martingale.

For a Lévy process L_t with triplet (σ, ν, γ), the quadratic variation is

$$[L, L]_t = \sigma^2 t + \sum_{s\leq t} |\Delta L_s|^2 = \sigma^2 t + \int_0^t \int_{\mathbb{R}} x^2 \Pi(\mathrm{d}s, \mathrm{d}x)$$

Sharp Bracket Process: The sharp bracket or predictable quadratic variation $\langle S, S\rangle_t$ process of a semimartingale S is the compensator of $[S, S]_t$. In other words, it is the unique predictable process that makes $[S, S]_t - \langle S, S\rangle_t$ into a local martingale. If S_t is a continuous, then $[S, S]_t = \langle S, S\rangle_t$

Denoting $[S, S]_t^c$ as the continuous part of $[S, S]_t$, we have $[S, S]^c = \langle S^{cm}, S^{cm}\rangle$. Since the discontinuous part of the quadratic variation process satisfies $\Delta[S, S]_t = (\Delta S_t)^2$, we can write the decomposition

$$\Big[S, S\Big]_t = \Big\langle S^{cm}, S^{cm}\Big\rangle_t + \sum_{0<s\leq t} |\Delta S_s|^2$$

<u>Note</u>: If S is of finite variation, then $[S, S]_t = \sum_{0<s\leq t} |\Delta S_s|^2$

2.10 Stochastic Integration with respect to Semimartingales

For a left continuous and locally bounded adapted process H, the following stochastic integral exists for a semimartingale integrators S_t and can be calculated as a limit of a Riemann sum which converges in probability :

$$(H \cdot S)_t = \int_0^t H_s \mathrm{d}S_s = \lim_{\delta_n \to 0} \sum_{k=0}^{n-1} H_{t_k} \left(S_{t_{k+1}} - S_{t_k} \right)$$

with $\delta_n = \max\left(t_{i+1}^n - t_i^n\right) \to 0$ for partitions $0 = t_0 < t_1 < \ldots < t_n = t$ of $[0, t]$.
Properties:

- The jumps of the integral occur at the points of jumps of L : $\Delta(H \cdot L)(t) = H_t \Delta S_t$

- $\left[\int_0^\cdot H_s \mathrm{d}S_s, \int_0^\cdot H_s \mathrm{d}S_s\right]_t = \int_0^t H_s^2 \mathrm{d}\Big[S,S\Big]_s$
- $\Big[S,S\Big]_t = S_t^2 - S_0^2 - 2\int_0^t S_{s-}\mathrm{d}S_s$
- If S_t is a semimartingale, the integral $(H \cdot S)_t$ is a semimartingale.

For example, in the case of Lévy process, the stochastic integral $(H\cdot S)_t$ can be decomposed into the sum of a local martingale M_t and a process of finite variation A_t

$$A_t = \gamma\int_0^t H_s \mathrm{d}s + \int_0^t\int_{|x|\geq 1} xH_s\Pi(\mathrm{d}s,\mathrm{d}x) \quad \text{and} \quad M_t = \sigma\int_0^t H_s\mathrm{d}W_s + \int_0^t\int_{|x|<1} xH_s\tilde{\Pi}(\mathrm{d}s,\mathrm{d}x)$$

The stochastic integral displays additional properties when the S is also a martingale :

- $(H \cdot S)_t$ is defined for a larger class of predictable processes.
- If S_t is a local martingale, the integral $(H \cdot S)_t$ is a local martingale.

Moreover, if S_t is a square-integrable martingale, and $\mathrm{E}\left(\int_0^T H^2(s)\mathrm{d}[S,S]_s\right) < \infty$, then the integral $(H \cdot S)_t$ is a square-integrable martingale with

$$\mathrm{E}\left[(H\cdot S)_t\right] = 0 \quad \text{and} \quad \mathrm{E}\left[(H\cdot S)_t^2\right] = \mathrm{E}\left[\int_0^t H_s^2 \mathrm{d}[S,S]_s\right]$$

2.11 Itô's formula for Semimartingales

Let $(X_t)_{t\geq 0}$ be a semimartingale and $F(s,x) : [0,T]\times\mathbb{R}\to\mathbb{R}$ be a C^2 function. Then $F(t,X_t)$ is a semimartingale and

$$\begin{aligned}
F(t,X_t) = F(0,X_0) &+ \int_0^t \frac{\partial F}{\partial s}(s,X_s)\,\mathrm{d}s + \int_0^t \frac{\partial F}{\partial x}(s,X_{s-})\,\mathrm{d}X_s \\
&+ \frac{1}{2}\int_0^t \frac{\partial^2 F}{\partial x^2}(s,X_{s-})\,\mathrm{d}\Big[X,X\Big]_s^c \\
&+ \sum_{s\leq t}\left[F(s,X_s) - F(s,X_{s-}) - \Delta X_s \frac{\partial F}{\partial x}(s,X_{s-})\right]
\end{aligned}$$

with $[X,X]^c = \Big\langle X^{cm}, X^{cm}\Big\rangle$. For a Lévy process $(X_t)_{t\geq 0}$ with triplet (σ^2,ν,γ), we have $\mathrm{d}\Big[X,X\Big]_s^c = \mathrm{d}\Big\langle\sigma W,\sigma W\Big\rangle_s = \sigma^2\mathrm{d}s$

2.12 Examples of Lévy processes

2.12.1 Brownian motion

The Brownian motion or Wiener process is a Lévy process and a continuous martingale with the following properties (for $0 \leq s,t$)

- $W_t - W_s = W_{t-s}$ is normally distributed : $W_{t-s} \sim \mathcal{N}(0, t-s)$
- $\mathrm{E}\Big[W_s \cdot W_t\Big] = s \wedge t$

- $\left[W,W\right]_t = \left\langle W,W\right\rangle_t = t$

The stochastic integral with regard to a Brownian motion W_t for a regular adapted square integrable process H with $\mathrm{E}\left(\int_0^T H_s^2 \mathrm{d}s\right) < \infty$ is defined as $\int_0^T H_t \mathrm{d}W_t$ and satisfy

- $\mathrm{E}\int_0^T H_t \mathrm{d}W_t = 0$
- $\mathrm{E}\left(\int_0^T H_t \mathrm{d}W_t\right)^2 = \mathrm{E}\left(\int_0^T H_t^2 \mathrm{d}t\right)$
- $\int_0^T H_t \mathrm{d}W_t$ is a continuous martingale
- $\left[\int_0^\cdot H_t \mathrm{d}W_t \mathrm{d}t, \int_0^\cdot H_t \mathrm{d}W_t \mathrm{d}t\right]_T = \int_0^T H_t^2 \mathrm{d}t$

The Itô formula can be considerably simplified for Itô processes $\mathrm{d}Z_t = \mu_t \mathrm{d}t + \sigma_t \mathrm{d}W_t$ with $\mathcal{F}_t$-adapted processes σ_t and μ_t

$$F\left(t, Z_t\right) = F\left(0, Z_0\right) + \int_0^t \frac{\partial F}{\partial s}\left(s, Z_s\right) \mathrm{d}s + \int_0^t \frac{\partial F}{\partial x}\left(s, Z_s\right) \mathrm{d}Z_s + \frac{1}{2}\int_0^t \frac{\partial^2 F}{\partial x^2}\left(s, Z_s\right) \sigma_s^2 \mathrm{d}s$$

2.12.2 Poisson process

Let $(\tau_i)_{i\geq 1}$ be a sequence of independent exponential random variables with parameter λ and $T_n = \sum_{i=1}^n \tau_i$ The process $(N_t, t \geq 0)$ defined by

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

is called a Poisson process with intensity $\lambda > 0$. It is a Lévy process whose increments follow a Poisson distribution $N_t \sim Poi(\lambda t)$ with probability mass function

$$\mathbb{P}\left[N_t = n\right] = \frac{(\lambda t)^n}{n!} \mathrm{e}^{-\lambda t}$$

N_t has the Lévy triplet $(0, 0, \lambda\delta_1)$ where $\delta_x(A)$ denotes the Dirac measure

$$\delta_x(A) = \begin{cases} 0, & x \notin A \\ 1, & x \in A \end{cases}$$

and has the following properties

- $\mathrm{E}[N_t] = \mathrm{Var}[N_t] = \lambda t$
- $\Phi_{N_t}(z) = \mathrm{E}\left[\mathrm{e}^{izN_t}\right] = \exp\left(\lambda t\left[\mathrm{e}^{iz} - 1\right]\right)$ for z $\in \mathbb{R}$

2.12.3 Coumpound Poisson process

A compound Poisson process with intensity λ and jump size distribution $F(A) = P(Y_i \in A)$ is a stochastic process X_t defined as

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

where N_t is a Poisson process with intensity λ and jumps sizes Y_i are independent and identically distributed random variables with distribution F and independent from N_t. X_t has the characteristic function

$$\Phi_{X_t}(z) = \exp\left(t \int_{\mathbb{R}} \left[\mathrm{e}^{izx} - 1\right] \nu(\mathrm{d}x)\right) = \exp\left(t\lambda\Big[\Phi_Y(z) - 1\Big]\right)$$

with $\nu(\mathrm{d}x) = \lambda F(\mathrm{d}x)$ and has the following properties

- X_t has Lévy triplet $\left(\int_{|x|\leq 1} x\nu(\mathrm{d}x), 0, \nu\right)$
- $\mathrm{E}[X_t] = \lambda t \mathrm{E}[Y]$
- $\mathrm{Var}[X_t] = \lambda t \mathrm{E}[Y^2]$

3 Generalized Hyperbolic distributions

3.1 Generalized Hyperbolic distribution

The generalized hyperbolic distribution (GH) is a flexible family of infinitely divisible distributions which offers major advantages over the normal distribution used in Brownian Motions. It allows us for skewness and (semi-)heavy tails while having density function, characteristic function as well as moment generating function in closed form.

We denote the generalized hyperbolic distribution $GH(\lambda, \alpha, \beta, \mu, \delta)$ with density function

$$f_{GH}(x; \lambda, \alpha, \beta, \delta, \mu) = a(\lambda, \alpha, \beta, \delta) \frac{K_{\lambda-\frac{1}{2}}\left(\alpha\sqrt{\delta^2 + (x-\mu)^2}\right) e^{\beta(x-\mu)}}{\left(\sqrt{\delta^2 + (x-\mu)^2}\right)^{\frac{1}{2}-\lambda}}$$

where

$$a(\lambda, \alpha, \beta, \delta) = \frac{(\alpha^2 - \beta^2)^{\lambda/2}}{\delta^\lambda \alpha^{(\lambda-1/2)} \sqrt{2\pi} \mathrm{K}_\lambda(\delta\sqrt{\alpha^2 - \beta^2})}$$

and

$$K_\lambda(z) = \frac{1}{2} \int_0^\infty u^{\lambda-1} \exp\left\{-\frac{z}{2}\left(u + u^{-1}\right)\right\} du$$

is the modified Bessel function of the third kind with index λ.

The location parameter μ, the steepness parameter α , the skewness parameter β and the scaling parameter δ satisfy

- $\mu \in \mathbb{R}$
- $\delta \geq 0, |\beta| < \alpha \quad \text{if } \lambda > 0$
- $\delta > 0, |\beta| < \alpha \quad \text{if } \lambda = 0$
- $\delta > 0, |\beta| \leq \alpha \quad \text{if } \lambda < 0$

The parameter $\lambda \in \mathbb{R}$ identifies the subfamily within generalized hyperbolic distributions. Setting $\lambda = 1$ yield the hyperbolic and $\lambda = -1/2$ the normal inverse Gaussian distribution.

The generalized hyperbolic family is also the superclass of the Gaussian, variance-gamma, normal and t-distribution and has the following mean and variance

$$\mathrm{E}X = \mu + \frac{\beta\delta \mathrm{K}_{\lambda+1}(\delta\gamma)}{\gamma \mathrm{K}_\lambda(\delta\gamma)}$$

$$\mathrm{Var}\, X = \delta^2 \left(\frac{\mathrm{K}_{\lambda+1}(\delta\gamma)}{\delta\gamma \mathrm{K}_\lambda(\delta\gamma)} + \frac{\beta^2}{\gamma^2} \left[\frac{\mathrm{K}_{\lambda+2}(\delta\gamma)}{\mathrm{K}_\lambda(\delta\gamma)} - \left(\frac{\mathrm{K}_{\lambda+1}(\delta\gamma)}{\mathrm{K}_\lambda(\delta\gamma)} \right)^2 \right] \right)$$

for $\gamma = \sqrt{\alpha^2 - \beta^2}$.

The characteristic function is given by

$$\Phi_{\mathrm{GH}}(z) = \mathrm{E}\left[\mathrm{e}^{izX}\right] = e^{i\mu z} \frac{\gamma K_1(\delta\sqrt{(\alpha^2 - (\beta + iz)^2)})}{\sqrt{(\alpha^2 - (\beta + iz)^2)}\, K_1(\delta\gamma)}$$

and the moment generating function exist for z with $|\beta + z| < \alpha$ and is given by

$$M_{\mathrm{GH}}(z) = e^{\mu z} \frac{\gamma K_1(\delta\sqrt{(\alpha^2 - (\beta + z)^2)})}{\sqrt{(\alpha^2 - (\beta + z)^2)}\, K_1(\delta\gamma)}$$

Generalized hyperbolic Lévy motion: The Generalized hyperbolic Lévy motion is a Lévy process $(L_t)_{0\leq t}$ such that L_1 follows a generalized hyperbolic distribution. It is important to note that for $t \neq 1$, the distribution of the Lévy process is not necessarily distributed according to a Generalized hyperbolic distribution. This will be analyzed later on in this section. The Generalized hyperbolic Lévy motion is purely discontinuous with paths of infinite variation and infinite activity with Lévy measure

$$\nu_{\mathrm{GH}}(\mathrm{d}z) = \frac{\mathrm{e}^{\beta z}}{|z|} \left\{ \frac{1}{\pi^2} \int_0^\infty \frac{\exp(-\sqrt{2y + \alpha^2}|z|)}{J_\lambda^2(\delta\sqrt{2y}) + Y_\lambda^2(\delta\sqrt{2y})} \frac{dy}{y} + \lambda \mathrm{e}^{-\alpha|z|} \right\} \mathrm{d}z, \qquad \text{for } \lambda \geq 0$$

and

$$\nu_{GH}(\mathrm{d}z) = \frac{\mathrm{e}^{\beta z}}{\pi^2 |z|} \int_0^\infty \left\{ \frac{\exp(-\sqrt{2y + \alpha^2}|z|)}{J_{-\lambda}^2(\delta\sqrt{2y}) + Y_{-\lambda}^2(\delta\sqrt{2y})} \frac{\mathrm{d}y}{y} \right\} \mathrm{d}z, \qquad \text{for } \lambda < 0$$

with

$$\mathrm{J}_\lambda(z) = (z/2)^\lambda \sum_{k=0}^\infty \frac{(-z^2/4)^k}{k!\Gamma(\lambda + k + 1)}$$

the Bessel functions of the first kind of order λ and

$$\mathrm{Y}_\lambda(z) = \frac{\mathrm{J}_\lambda(z)\cos(\lambda\pi) - \mathrm{J}_{-\lambda}(z)}{\sin(\lambda\pi)}$$

the Bessel functions of the second kind of order λ. In the case of integer order n, the function is defined by taking the limit as a non-integer λ tends to n : $Y_n(x) = \lim_{\lambda\to n} Y_\lambda(x)$

Alternative parameterizations: α and β can be replaced by the alternative parameterizations

- $\rho = \frac{\beta}{\alpha}, \quad \zeta = \delta\sqrt{\alpha^2 - \beta^2}$
- $\chi = \rho\xi, \quad \xi = \frac{1}{\sqrt{1+\zeta}}$

3.2 Normal Inverse Gaussian distribution

The Normal Inverse Gaussian distribution is a special case of the Generalized hyperbolic distribution. For $\lambda = -1/2$ we obtain the Normal Inverse Gaussian distribution:

$$\mathrm{GH}(-1/2, \alpha, \beta, \delta, \mu) \stackrel{d}{=} \mathrm{NIG}(\alpha, \beta, \delta, \mu)$$

The distribution admits closed-form density for $x \in \mathbb{R}$

$$f_{\mathrm{NIG}}(x; \alpha, \beta, \delta, \mu) = \frac{\alpha\delta}{\pi} \frac{K_1(\alpha\sqrt{\delta^2 + (x - \mu)^2})}{\sqrt{\delta^2 + (x - \mu)^2}} e^{(\delta\gamma + \beta(x - \mu))}$$

where $\gamma = \sqrt{\alpha^2 - \beta^2}$ and

- $\mu \in \mathbb{R}$
- $0 < \delta$
- $|\beta| \leq \alpha$

The distribution has the following mean and variance

$$\mathrm{E}X = \mu + \frac{\delta\beta}{\gamma} \qquad \mathrm{Var}\, X = \frac{\delta\alpha^2}{\gamma^3}$$

The characteristic function is given by

$$\Phi_{\mathrm{NIG}}(z) = \mathrm{E}\left[\mathrm{e}^{izX}\right] = e^{i\mu z + \delta\left(\gamma - \sqrt{\alpha^2 - (\beta + iz)^2}\right)}$$

3.3 Hyperbolic distribution

The Hyperbolic distribution is also a special case of the Generalized hyperbolic distribution. For $\lambda = 1$ we obtain the hyperbolic distributions :

$$\mathrm{GH}(-1, \alpha, \beta, \mu, \delta) \stackrel{d}{=} \mathrm{HYP}(\alpha, \beta, \delta, \mu)$$

The distribution admits closed-form density for $x \in \mathbb{R}$

$$f_{\mathrm{HYP}}(x; \alpha, \beta, \delta, \mu) = \frac{\sqrt{\alpha^2 - \beta^2}}{2\delta\alpha \mathrm{K}_1(\delta\sqrt{\alpha^2 - \beta^2})} e^{\left(-\alpha\sqrt{\delta^2 + (x-\mu)^2} + \beta(x-\mu)\right)}$$

where $\gamma = \sqrt{\alpha^2 - \beta^2}$ and

- $\mu \in \mathbb{R}$
- $0 \leq \delta$
- $|\beta| < \alpha$

and the following mean and variance

$$\mathrm{E}X = \mu + \frac{\delta\beta K_2(\delta\gamma)}{\gamma K_1(\delta\gamma)} \qquad \mathrm{Var}\, X = \frac{\delta K_2(\delta\gamma)}{\gamma K_1(\delta\gamma)} + \frac{\beta^2\delta^2}{\gamma^2}\left(\frac{K_3(\delta\gamma)}{K_1(\delta\gamma)} - \frac{K_2^2(\delta\gamma)}{K_1^2(\delta\gamma)}\right)$$

The characteristic function is given by

$$\Phi_{\mathrm{HYP}}(z) = \mathrm{E}\left[\mathrm{e}^{izX}\right] = e^{i\mu z} \frac{\gamma K_1\left(\delta\sqrt{\alpha^2 - (\beta + iz)^2}\right)}{\sqrt{(\alpha^2 - (\beta + iz)^2)} K_1(\delta\gamma)}$$

3.4 Variance Gamma distribution

The Variance Gamma distribution is a special case of the Generalized hyperbolic distribution. For $\delta \to 0$ we obtain the Variance Gamma distribution:

$$\lim_{\delta \to 0} \mathrm{GH}(\lambda, \alpha, \beta, \delta, \mu) \stackrel{d}{=} \mathrm{VG}(\lambda, \alpha, \beta, \mu)$$

The distribution admits closed-form density for $x \in \mathbb{R}$

$$f_{\mathrm{VG}}(x; \lambda, \alpha, \beta, \mu) = \frac{\gamma^{2\lambda} |x - \mu|^{\lambda - 1/2} K_{\lambda - 1/2}(\alpha |x - \mu|)}{\sqrt{\pi} \Gamma(\lambda) (2\alpha)^{\lambda - 1/2}} e^{\beta(x - \mu)}$$

where $\gamma = \sqrt{\alpha^2 - \beta^2}$, $\mu, \alpha, \beta \in \mathbb{R}$ and $\lambda > 0$.

The distribution has the following mean and variance

$$\mathrm{EX} = \mu + \frac{2\beta\lambda}{\gamma^2} \qquad \mathrm{Var}\, X = 2\lambda \left(\gamma^{-2} + \frac{2\beta^2}{\gamma^4} \right)$$

The characteristic function is given by

$$\Phi_{\mathrm{VG}}(z) = e^{i\mu z} \left(\frac{\gamma^2}{\alpha^2 - (\beta + iz)^2} \right)^{\lambda}$$

3.5 Generalized Inverse Gaussian Distributions

The density function of the Generalized Inverse Gaussian distribution is given for $x \in \mathbb{R}^+$ by

$$f_{\mathrm{GIG}}(x; \lambda, \delta, \gamma) = \left(\frac{\gamma}{\delta} \right)^{\lambda} \frac{1}{2K_\lambda(\delta\gamma)} x^{\lambda - 1} e^{-\frac{1}{2}\left(\delta^2 x^{-1} + \gamma^2 x\right)}$$

with the following restrictions on the parameters

- $\delta \geq 0, \gamma > 0, \text{ if } \lambda > 0$
- $\delta > 0, \gamma > 0, \text{ if } \lambda = 0$
- $\delta > 0, \gamma \geq 0, \text{ if } \lambda < 0$

The distribution has the following mean and variance

$$\mathrm{EX} = \frac{\gamma K_{\lambda+1}(\gamma\delta)}{\delta K_\lambda(\gamma\delta)}$$

$$\mathrm{Var}\, X = \frac{\gamma^2}{\delta^2} \left[\frac{K_{\lambda+2}(\gamma\delta)}{K_\lambda(\gamma\delta)} - \left(\frac{K_{\lambda+1}(\gamma\delta)}{K_\lambda(\gamma\delta)} \right)^2 \right]$$

The characteristic function is given by

$$\Phi_{\mathrm{GIG}}(z) = \left(\frac{\gamma}{\sqrt{\gamma^2 - 2iz}} \right)^{\lambda} \frac{K_\lambda(\delta\sqrt{\gamma^2 - 2iz})}{K_\lambda(\delta\gamma)}$$

It is a limiting case of the Generalized Hyperbolic distribution. If we assume

$$\beta = \alpha - \frac{\psi}{2} \qquad \alpha \to \infty, \quad \delta \to 0 \qquad \delta \to 0 \qquad \alpha\delta^2 \to \tau \qquad \mu = 0$$

then

$$f_{GH}(x; \lambda, \alpha, \beta, \delta, \mu) \to f_{\text{GIG}}(x; \lambda, \sqrt{\tau}, \sqrt{\psi})$$

3.5.1 The Gamma distribution

Gamma distribution The density function of the Gamma distribution with shape parameter α and rate parameter β is defined for $x \in \mathbb{R}^+$ by

$$f_{\text{Gamma}}\,(x; \alpha, \beta) = \frac{\beta^\alpha}{\Gamma(\alpha)} x^{\alpha-1} e^{-\beta x} \qquad \alpha, \beta > 0$$

Its cumulative distribution function is the regularized gamma function:

$$F_{\text{Gamma}}\,(x; \alpha, \beta) = \int_0^x f_{\text{Gamma}}\,(s; \alpha, \beta) \mathrm{d}s = \frac{\gamma(\alpha, \beta x)}{\Gamma(\alpha)}$$

where $\gamma(\alpha, \beta x)$ is the lower incomplete gamma function

$$\gamma(\alpha, \beta x) = \int_0^{\beta x} t^{\alpha-1} \mathrm{e}^{-t} \mathrm{d}t$$

The distribution has the following mean and variance

$$\mathrm{EX} = \frac{\alpha}{\beta} \qquad \mathrm{Var}\, X = \frac{\alpha}{\beta^2}$$

The characteristic function is given by

$$\Phi_{\text{Gamma}}\,(z) = \mathrm{E}\left[\mathrm{e}^{izX}\right] = \left(1 - \frac{iz}{\beta}\right)^{-\alpha}$$

The Gamma distribution is a subfamily of the Generalized Inverse Gaussian distribution. In the limiting case $\lambda > 0, \delta = 0$, we have

$$f_{\text{GIG}}(x; \lambda, 0, \gamma) = f_{\text{Gamma}}\,\left(x; \lambda, \gamma^2/2\right)$$

Gamma process: The Gamma process is a pure-jump increasing Lévy process $(L_t)_{0 \leq t}$ such that L_1 follows a Gamma distribution. It has the following Lévy triplet

$$\left(\frac{\alpha(1 - e^{-\beta})}{\beta}, 0, \alpha e^{-\beta x} x^{-1} 1_{(x>0)}\right)$$

The process is often denoted by $\Gamma(t; \alpha, \beta)$ and follows the distribution Gamma$(\alpha t, \beta)$

3.5.2 Inverse Gaussian distribution

The density function of the Inverse Gaussian distribution is defined for $x \in \mathbb{R}^+$ by

$$f_{\mathrm{IG}}(x;\delta,\gamma) = \frac{\delta}{\sqrt{2\pi x^3}} \exp\left(\delta\gamma - \frac{1}{2}\left(\delta^2 x^{-1} + \gamma^2 x\right)\right)$$

The Inverse Gaussian distribution is a special case of the Generalized Inverse Gaussian distribution. For $\lambda = -\frac{1}{2}$ the $\mathrm{GIG}(\lambda,\delta,\gamma)$ reduces to the $\mathrm{IG}(\delta,\gamma)$.

The distribution has the following mean and variance

$$\mathrm{E}X = \frac{\delta}{\gamma} \qquad \mathrm{Var}\, X = \frac{\delta}{\gamma^3}$$

and its characteristic function is given by

$$\Phi_{\mathrm{IG}}(z) = \exp\left(-\delta\left(\sqrt{-2iz+\gamma^2} - \gamma\right)\right)$$

The Lévy measure of the Inverse Gaussian distribution is defined for $x > 0$ by

$$\nu = \frac{1}{\sqrt{2\pi}}\delta x^{-3/2} \exp\left(-\frac{\gamma^2 x}{2}\right)$$

and has in its Lévy triplet the component equals $\gamma = \frac{\delta}{\gamma}(2\Phi(\gamma) - 1)$

Inverse Gaussian process: The Inverse Gaussian process is the Lévy process that follows the distribution $\mathrm{IG}(x;\delta t,\gamma)$

3.6 GH distributions and convolution

The probability distribution of the sum of two or more independent random variables is the convolution of their individual distributions. For independent random variables X and Y with respective distribution f_X and f_Y, the distribution of $Z = X + Y$ can be derived from

$$f_Z(z) = (f_X * f_Y)(z) = \int_{-\infty}^{\infty} f_X(x) f_Y(z-x)dx = \int_{-\infty}^{\infty} f_X(z-x) f_Y(x)dx$$

Many families of infinitely divisible distributions are closed under convolution. In the case of Lévy processes, it means that if the distribution of a Lévy process L_t at one point in time t belongs to a particular family, then the distribution of L_t at all points in time $t > 0$ belong to the same family of distributions. The Brownian motion is a good example : for all time $t > 0$, the Brownian motion W_t follows a normal distribution with mean 0 and variance t.

The majority of generalized hyperbolic distributions fail however to be closed under convolution: a Lévy process with a generalised hyperbolic distribution at one point in time may not have generalized hyperbolic at another point in time. This property can be verified by observing their characteristic functions. For a Lévy process L_t such that L_1 follows a generalized hyperbolic distribution with characteristic function

$$\Phi_{L_1}(z) = \Phi_{\mathrm{GH}}(z) = e^{i\mu z}\,\frac{\gamma K_1(\delta\sqrt{(\alpha^2-(\beta+iz)^2)})}{\sqrt{(\alpha^2-(\beta+iz)^2)}\,K_1(\delta\gamma)}$$

the characteristic function of L_t is $\Phi_{L_t}(z) = \mathrm{e}^{t\Psi(z)} = \mathrm{e}^{t\ln(\Phi_{\mathrm{GH}}(z))}$. For values of $t \neq 1$, the characteristic function will generally no longer be the characteristic function of a generalized hyperbolic distribution. Some subfamilies from the generalized hyperbolic family are nonetheless closed under convolution, which will be the subject of the next subsection.

3.7 GH distributions and normal variance-mean mixture

3.7.1 Normal variance-mean mixture

A normal variance-mean mixture with mixing probability density g is the continuous probability distribution of a random variable Y of the form

$$Y = \eta + \psi V + \sigma\sqrt{V}X$$

with random variables X and V and constant parameter such that

- $X \sim \mathcal{N}(0,1)$
- $X \perp V$
- V has a probability distribution g supported on $(0,\infty)$
- α, β and $\sigma > 0$ are real numbers.

The probability density function f of a normal variance-mean mixture with mixing probability density g can be obtained from

$$f(x) = \int_0^\infty \varphi(x;\eta+\psi v,\sigma^2 v)g(v)dv = \int_0^\infty \frac{1}{\sqrt{2\pi\sigma^2 v}}\exp\left(\frac{-(x-\alpha-\beta v)^2}{2\sigma^2 v}\right)g(v)dv$$

where $\varphi(x;\mu,\sigma^2)$ is the density of a normal distribution of mean μ and variance σ. The class of GH distributions can be obtained by mean-variance mixtures of normal distributions where the mixing distribution is a generalized inverse Gaussian distribution.

$$f_{\mathrm{GH}}(x;\lambda,\alpha,\beta,\delta,\mu) = \int_0^\infty \varphi(x;\mu+\beta v,v)f_{\mathrm{GIG}}\left(v;\lambda,\delta,\sqrt{\alpha^2-\beta^2}\right)dv$$

The distribution of a normal variance-mean mixture can also be thought of as the distribution of the value of a Wiener process $\psi t + \sigma W_t$ observed at a random time point independent of the Wiener process and with probability density function g. From there on, an obvious link can be made with Lévy process obtained by Brownian subordination.

3.7.2 Convolutions of Normal variance-mean mixtures

Let $Y_i = \psi V_i + \sqrt{V_i} X_i$ be normal variance–mean mixtures with independent random variables V_i. They have the same distribution if and only if V_i are identically distributed and the distribution of $Y_1 + Y_2$ is such that

$$Y_1 + Y_2 \stackrel{d}{=} \sqrt{V_1 + V_2} X + \psi \left(V_1 + V_2\right)$$

In other words, the sum $Y_1 + Y_2$ is also a variance–mean mixture with the same scale ψ and the mixing variable $V_1 + V_2$. As a result, the subfamily of distributions of the variance–mean mixture Y_i is closed under convolution if and only if the subfamily of V_i is closed under convolution. (Podgórski and Wallin 2016).

3.7.3 Convolution-closed Generalized Hyperbolic distributions

Within the Generalized Inverse Gaussian distributions, only two subfamilies are closed under convolution: the gamma distributions and the inverse Gaussian distributions. This means that Generalized Hyperbolic distributions obtained by normal mean-variance mixtures are closed under convolutions only if the mixing probability density is chosen from one of these two distributions.

This brings us to two particular distributions of the Generalized Hyperbolic family : the Variance Gamma and Normal Inverse Gaussian distributions, which can be build by normal mean-variance mixtures with respective mixing distribution Gamma and Inversion Gaussian. These two distributions are the building blocks of Lévy process for which the distribution of the increments are closed under convolution, making them particularly attractive

- $\mathrm{NIG}(\alpha, \beta, \delta_1, \mu_1) + \mathrm{NIG}(\alpha, \beta, \delta_2, \mu_2) \stackrel{d}{=} \mathrm{NIG}(\alpha, \beta, \delta_1 + \delta_2, \mu_1 + \mu_2)$
- $\mathrm{VG}\left(\lambda_1, \alpha, \beta, \mu_1\right) + \mathrm{VG}\left(\lambda_2, \alpha, \beta, \mu_2\right) \stackrel{d}{=} \mathrm{VG}\left(\lambda_1 + \lambda_2, \alpha, \beta, \mu_1 + \mu_2\right)$

From the equation

$$f_{\mathrm{GH}}(x; \lambda, \alpha, \beta, \delta, 0) = \int_0^\infty \varphi(x; \beta v, v) f_{\mathrm{GIG}}\left(v; \lambda, \delta, \sqrt{\alpha^2 - \beta^2}\right) dv$$

with location parameter μ set to zero, it is easy to derive the mixing GIG distribution for these particular cases of GH distributions.

	GH Distribution	GIG mixing distribution
Variance Gamma	$\mathrm{GH}\left(\lambda, \alpha, \beta, 0, 0\right)$	$\mathrm{Gamma}\left(\lambda, \frac{1}{2}(\alpha^2 - \beta^2)\right)$
Normal Inverse Gaussian	$\mathrm{GH}\left(-1/2, \alpha, \beta, \delta, 0\right)$	$\mathrm{IG}\left(\delta, \sqrt{\alpha^2 - \beta^2}\right)$

Table 3.1: Mixing parameter for convolution-closed Generalized Hyperbolic distributions

3.8 GH Lévy motions by Brownian subordination

The result of the previous subsection can be linked to the subordination of Brownian motion. If we define the process

$$X_t = \mu t + \beta Z_t + W_{Z_t}$$

where Z_t is a subordinator generated by a Generalized Inverse Gaussian distribution $\mathrm{GIG}\left(\lambda, \delta, \sqrt{\alpha^2 - \beta^2}\right)$ and W_{Z_t} is a Brownian motion subordinated by Z_t, then the process X_t is a Generalized Hyperbolic Lévy motion generated by $\mathrm{GH}(\lambda, \alpha, \beta, \delta, \mu)$.

From there, it is easy to association these results to those from the previous subsection to generate the desired process. We will detail briefly the Variance Gamma and Normal Inverse Gaussian process as well as some of their popular alternative parametrizations.

3.8.1 The Variance Gamma process

The Variance Gamma process is a Lévy process VG_t with increments that follows a Variance Gamma distribution.

$$\mathrm{VG}_t \sim \mathrm{VG}(\lambda t, \alpha, \beta)$$

It can be build by subordinating a Brownian Motion with drift β by a Gamma process $\Gamma_t \sim \mathrm{Gamma}\left(\lambda t, \frac{1}{2}(\alpha^2 - \beta^2)\right)$

$$\mathrm{VG}_t = \beta \Gamma_t + W_{\Gamma_t}$$

The characteristic function of the Variance Gamma process VG_t is given by

$$\Phi_{\mathrm{VG}_t}(z) = \left(\frac{\gamma^2}{\alpha^2 - (\beta + iz)^2}\right)^{\lambda t}$$

It is common in the literature to see the alternative specification $\mathrm{VG}^\diamond(t; \sigma, \nu, \theta)$ defined by

$$\mathrm{VG}^\diamond(t; \sigma, \nu, \theta) = \theta \Gamma_t^\diamond + \sigma W_{\Gamma_t^\diamond}$$

with $\Gamma_t^\diamond \sim \mathrm{Gamma}\left(\lambda t = \frac{t}{\nu}, \sqrt{\alpha^2 - \beta^2} = \gamma = \frac{1}{\nu}\right)$ and the equivalence

$$\sigma^2 = \frac{2\lambda}{\alpha^2 - \beta^2}, \quad \nu = \frac{1}{\lambda}, \quad \theta = \beta\sigma^2$$

Under this specification, the Variance Gamma process follows a $\mathrm{VG}^\diamond(\sigma\sqrt{t}, \frac{\nu}{t}, t\theta)$ distribution with characteristic function

$$\Phi_{\mathrm{VG}_t^\diamond}(z) = \left(1 - iz\theta\nu + \frac{1}{2}\sigma^2\nu z^2\right)^{-\frac{t}{\nu}}$$

The characterization of the Variance Gamma distribution as a CGMY distribution with $Y = 0$ is also prevalent. The relations with the other parametrization are

$$C = \frac{1}{\nu} = \lambda > 0 \qquad \frac{G - M}{2} = \frac{\theta}{\sigma} = \beta \qquad \frac{G + M}{2} = \frac{\sqrt{\frac{2}{\nu} + \frac{\theta^2}{\sigma^2}}}{\sigma} = \alpha$$

$$G = \left(\sqrt{\frac{\theta^2\nu^2}{4} + \frac{1}{2}\sigma^2\nu} - \frac{1}{2}\theta\nu \right)^{-1} > 0 \qquad M = \left(\sqrt{\frac{\theta^2\nu^2}{4} + \frac{1}{2}\sigma^2\nu} + \frac{1}{2}\theta\nu \right)^{-1} > 0$$

$$\sigma^2 = \frac{2C}{MG} \qquad \theta = \frac{C(G-M)}{MG}$$

Under this parametrization, the Variance Gamma process can easily be represented as the difference of two independent Gamma process

$$\Phi_{\mathrm{VG}_t^\dagger}(z) = \Gamma^\dagger(t; C, M) - \Gamma^\dagger(t; C, G)$$

3.8.2 The Normal Inverse Gaussian process

The Normal Inverse Gaussian process is a Lévy process NIG_t with increments that follows a Normal Inverse Gaussian distribution.

$$\mathrm{NIG}_t \sim \mathrm{NIG}\left(\alpha, \beta, \delta t\right)$$

It can be build by subordinating a Brownian Motion with drift β by an Inverse Gaussian process $\chi_t \sim \mathrm{IG}\left(\delta t, \sqrt{\alpha^2 - \beta^2}\right)$

$$\mathrm{NIG}_t = \beta\chi_t + W_{\chi_t}$$

The characteristic function of the Normal Inverse Gaussian process NIG_t is given by

$$\Phi_{\mathrm{NIG}_t}(z) = e^{\delta t\left(\gamma - \sqrt{\alpha^2 - (\beta + iz)^2}\right)}$$

3.9 Simulation of Generalized Hyperbolic Random Variables

From the previous subsections results a simple algorithm to generate a sample Y from a Generalized Hyperbolic distribution $\mathrm{GH}(\lambda, \alpha, \beta, \delta, \mu)$

1. Sample X from a Generalized Inverse Gaussian distribution $\mathrm{GIG}\left(\lambda, \delta, \sqrt{\alpha^2 - \beta^2}\right)$.
2. Sample N from a standard normal distribution $\mathcal{N}(0, 1)$
3. Return $Y = \mu + \beta X + \sqrt{X}N$

4 Review of Temperature Models

In this section will be briefly presented the literature for some of the temperature models that were used as inspiration for this work. The literature on temperature modeling is extensive only a small subset of it is mentioned here. These models have been applied repeatedly by different authors for different cities and periods and the papers resulting from those replications won't be mentioned here, though the references of some can be found in the appendices.

4.1 Alaton (2002)

It is well known and obvious from empirical evidence that temperature move around a predictable annual cycle similar to a sinusoid. As a result, most temperature models make use of mean-reverting process. Alaton et al. (2002) use the following Ornstein-Uhlenbeck model for the daily average temperature (DAT) :

$$\mathrm{d}T_t = \mathrm{d}S_t + \kappa[T_t - S_t]\mathrm{d}t + \sigma(t)\mathrm{d}W_t$$

They model the average temperature as the combination of a linear trend and a sinusoid of amplitude C, angular frequency $\omega = \frac{2\pi}{365}$ and phase φ

$$S_t = A + Bt + C\sin(\omega t + \varphi)$$

They assume no time dependency for the speed of mean reversion, which is model with a constant κ but do not provide a justification for it. The volatility σ_t is a piecewise constant function, with a constant value assigned to each month. The addition of the term $\mathrm{d}S_t = \Big[B + \omega C\cos(\omega t + \varphi)\Big]\,\mathrm{d}t$ follows the argument made in Dornier and Queruel (2000) that without it, the temperature would be a mean reverting process but would not revert to S_t in the long run.

They justify the use of a Brownian motion by stating that the detrended and deseasonalized temperature variation are close to normally distributed, but do not provide any statistical test to prove it.

4.2 Brody (2002)

Brody et al. (2002) argue for the existence of long-range time-dependency based on the ST method developed by Syroka and Toumi (2001) and suggest the use of a fractional Brownian motion (fBm).

$$dT_t = \kappa(t)[S_t - T_t]\mathrm{d}t + \sigma(t)\mathrm{d}W_t^H$$

Seasonality in the mean and volatility is modeled by a simple sinusoid function similar to the one used for S(t) in Alaton (2000). They didn't include the component $\mathrm{d}S(t)$ which means the temperature will no revert to the seasonal mean $S(t)$.

$$S_t = a_0 + a_1 \sin\left(\frac{2\pi}{365}t + \varphi_1\right) \quad \text{and} \quad \sigma(t) = \beta_0 + \beta_1 \sin\left(\frac{2\pi}{365}t + \varphi_2\right)$$

They introduce the idea of time dependency in the speed of mean reversion $\kappa(t)$ but do not proceed to fit the model to data. Moreover, the assume κ to be constant in their fictive example.

Starting from $\tilde{T}_t = T_t - S_t$, they apply the ST method in order to quantify how the variability of the fluctuation of $\tilde{T}(t)$ depends on time. They find a Hurst parameter with a value of H=0.61 for temperature recorded from 1772 to 1999 in central England and use that result to justify the existence of long-range dependence. Benth and Saltyte-Benth (2005) comment that the analysis should have been performed after removing all seasonalities and found that fractional Brownian motion does not seem appropriate for the Norwegian temperature used in their analysis.

4.3 Benth and Saltyte-Benth (2005)

They generalize the model of Alaton (2002) by replacing the Brownian motion with a Levy process L(t), namely a generalized hyperbolic process.

$$\mathrm{d}T_t = \mathrm{d}S_t + \kappa\Big[T_t - S_t\Big]\mathrm{d}t + \sigma(t)\mathrm{d}L_t$$

They use the following discretization

$$\Delta T_t = \Delta S_t + \kappa[T_{t-1} - S_{t-1}]\Delta t + \sigma(t-1)\Delta L_t \quad \text{with } \Delta t = 1 \text{ and } \Delta Y_t = Y_t - Y_{t-1}$$

to obtain the discrete-time model

$$\tilde{T}_t = (1+\kappa)\tilde{T}_{t-1} + \sigma(t)\varepsilon_t \quad \text{with } \tilde{T}_t = T_t - S_t$$

Because they did not find any significant linear trend in their analysis, the seasonality is modeled with a simple sinusoid

$$S_t = A + C\sin(\omega t + \varphi)$$

They mention however that the absence of linear trend may be due to only 14 years of historic data being used. They use their hypothesis of constant mean reversion to regress the deseasonalized temperature on the deseasonalized temperature of the previous day to investigate the mean reversion parameter κ

$$\tilde{T}_t - (1+\kappa)\tilde{T}_{t-1} = \sigma(t)\varepsilon_t$$

They were not able to find any clear monthly or yearly pattern in any of the city where they analysis was performed. They compute yearly and monthly value of κ then take the average for each series respectively and observe that these average values are very close. They use the two previous result to justify the use of a constant mean reversion parameter κ. However, they do not provide average values for each individual month or year. It is important to note that the model above may lead to very unstable observation for κ. If the temperature T_t is observed very close to its predicted mean S_t, then $\tilde{T}_t$ will be closed to zero. If the next observation $\tilde{T}_{t+1}$ is considerably higher, then the measured value $(1+\kappa)$ will be very high.

They propose a multiplicative time series model for the residuals given by

$$\tilde{\varepsilon_t} = \sigma(t)\varepsilon_t$$

with

$$\sigma^2(t) = c + \sum_{i=1}^{I_2} a_i \sin\left(\frac{2i\pi}{365}t\right) + \sum_{j=1}^{J_2} d_j \left(\frac{2j\pi}{365}t\right)$$

Finally, they reject the hypothesis that the residuals after dividing out the seasonal variation $\varepsilon_t = \frac{\tilde{\varepsilon_t}}{\sigma_t}$ are independent and identically normally distributed for roughly half of the city used in their analysis and suggest the use of the generalized hyperbolic family. They also inspect for fractionality by inspecting autocorrelation function of the residual. They conclude that a fractional model does not seem necessary as they do not observe decay at a hyperbolic rate as predicted by the fractional Brownian motion

4.4 Benth and Saltyte-Benth (2007)

In this paper, they propose a similar Ornstein–Uhlenbeck model, this time restricted to a Standard Brownian motion W_t in order to make analytical pricing possible. They conduct their analysis on 45 years of daily data in Stockholm, Sweden.

$$\mathrm{d}T_t = \mathrm{d}S_t + \kappa\Big[T_t - S_t\Big]\mathrm{d}t + \sigma(t)\mathrm{d}W_t$$

They use a similar truncated Fourier series to model the seasonal component S_t and $\sigma^2(t)$ as in their previous paper, this time with a linear trend

$$S_t = a + bt + \sum_{i=1}^{I_1} a_i \sin\left(\frac{2i\pi}{365}(t - f_i)\right) + \sum_{j=1}^{J_1} b_j \cos\left(\frac{2j\pi}{365}(t - g_i)\right)$$

$$\sigma^2(t) = c + \sum_{i=1}^{I_2} a_i \sin\left(\frac{2i\pi}{365}t\right) + \sum_{j=1}^{J_2} d_j \left(\frac{2j\pi}{365}t\right)$$

They find an explicit solution for the dynamic of the temperature using Itô's formula

$$T_t = S_t + (T_0 - S_0)\mathrm{e}^{-\kappa t} + \int_0^t \sigma(u)\mathrm{e}^{-\kappa(t-u)}\mathrm{d}W_u$$

which is then discretized to yield the discrere time series model

$$\tilde{T}_{t+1} = \alpha\tilde{T}_t + \tilde{\sigma}(t)\epsilon_t$$

where ϵ_t is i.i.d. standard normally distributed, $\alpha = \mathrm{e}^{-x}$ and $\tilde{\sigma}(t) = \alpha\sigma(t)$.

They observe a slight increase in the variance for the summer compared with the spring and fall seasons. They observe a rapid decay in the autocorrelation for the first lags and acknowledge that GARCH model could be appropriate but decide not to investigate the matter as the analytical pricing of the derivatives will be significantly more difficult.

5 A note on data handling and the data used

As the dynamics of temperature has a much bigger predictable component compared to the dynamics of equity stocks or commodity prices, we can confidently use more efficient, exhaustive and therefore complicated tools for its modeling. In practice, it means that we are required to uses data from much longer period in order to avoid over-fitting. As the dynamics of the temperature is much more stable through time than that of financial asset, this does not pose a problem if we are able to get our hands on high quality data.

Unfortunately, weather data are not nearly as accessible as financial data. Nonetheless, the European Climate Assessment & Dataset project[1] provides a wide array of daily measurement of weather variables for an impressive collection of cities. Though the models laid out in the next sections could have been improved further by using hourly measurement, only the daily average temperature[2] was available and therefore was used to the calibration of the dynamic of the temperature in continuous time.

Before any analysis gets underway, it is important to inspect the integrity of the data. It is important to verify that no data is missing nor has unrealistic value. Different techniques for handling missing data can be found in the litterature. This was fortunately not necessary as our dataset did not suffer from missing data. It is also important to limit the scope of the data used. While using data from a very large period can seem attractive, it is also fair to argue that data from too long ago probably will not reflect the dynamic of today, and therefore introduce more harm than good in our analysis. In this work, the data from 1960 to 2019 was used, and the leap years were removed.

6 Modelling of Daily Average Temperature (DAT)

We will focus our work on models of the form, improving the model in the litterature by allowing the speed of mean reversion to depend on time, while using a Lévy process L_t as the driving noise

$$\mathrm{dT}_t = \mathrm{d}S_t + \kappa(t)\Big[S_t - T_t\Big]\mathrm{d}t + \sigma(t)\mathrm{d}L_t$$

If we define $\tilde{T}_t := T_t - S_t$ to be the detrended and seasonalized temperature, we can rewrite the model as the Ornstein–Uhlenbeck process

$$d\tilde{T}_t = -\kappa(t)\tilde{T}_t\mathrm{d}t + \sigma(t)\mathrm{d}L_t$$

Using Itô's formula for semimartingale with

- $F(t, X) = \mathrm{e}^{\int_0^t \kappa(\xi)\mathrm{d}\xi} X$
- $F(0, \tilde{T}_0) = \tilde{T}_0$
- $\frac{\partial F}{\partial s}(s, T_s) = \kappa(s)\mathrm{e}^{\int_0^s \kappa(\xi)\mathrm{d}\xi}\tilde{T}_s = \kappa(s)F(s, \tilde{T}_s)$
- $\frac{\partial F}{\partial x}(s, T_s) = \mathrm{e}^{\int_0^s \kappa(\xi)\mathrm{d}\xi}$

[1] https://www.ecad.eu/

[2] Defined as the average of the highest and lowest temperature for a particular day.

- $\frac{\partial^2 F}{\partial x^2}\left(s, T_s\right) = 0$

we find that $\left[F\left(s, T_s\right) - F\left(s, T_{s-}\right) = \Delta T_s \frac{\partial F}{\partial x}\left(s, T_{s-}\right)\right]$ and $\frac{\partial F}{\partial x}\left(s, T_{s-}\right) = \frac{\partial F}{\partial x}\left(s, T_s\right)$.
As a result, the Itô formula simplifies as

$$F(t,\tilde{T}_t) = F(0,\tilde{T}_0) + \int_0^t \frac{\partial F}{\partial s}(s,\tilde{T}_s)\mathrm{d}s + \int_0^t \frac{\partial F}{\partial x}(s,\tilde{T}_s)\,\mathrm{d}\tilde{T}_s$$

or in differential form

$$\begin{aligned} dF(t,\tilde{T}_t) &= \frac{\partial F}{\partial t}(t,\tilde{T}_t)\mathrm{d}t + \frac{\partial F}{\partial x}(t,\tilde{T}_t)\mathrm{d}\tilde{T}_t \\ &= \kappa(t)\mathrm{e}^{\int_0^t \kappa(\xi)\mathrm{d}\xi}\tilde{T}_t\mathrm{d}t + \mathrm{e}^{\int_0^t \kappa(\xi)\mathrm{d}\xi}\Big[-\kappa(t)\tilde{T}_t\mathrm{d}t + \sigma(t)\mathrm{d}L_t\Big] \\ &= \mathrm{e}^{\int_0^t \kappa(\xi)\mathrm{d}\xi}\sigma(t)\mathrm{d}L_t \end{aligned}$$

Integrating both sides and rearranging leaves us with

$$\begin{aligned} \tilde{T}_t &= \tilde{T}_0\mathrm{e}^{-\int_0^t \kappa(\xi)\mathrm{d}\xi} + \mathrm{e}^{-\int_0^t \kappa(\xi)\mathrm{d}\xi}\int_0^t \mathrm{e}^{\int_0^s \kappa(\xi)\mathrm{d}\xi}\sigma(s)\mathrm{d}L_s \\ &= \tilde{T}_0\mathrm{e}^{-\int_0^t \kappa(\xi)\mathrm{d}\xi} + \int_0^t \mathrm{e}^{-\int_s^t \kappa(\xi)\mathrm{d}\xi}\sigma(s)\mathrm{d}L_s \end{aligned}$$

Using the previous results, we find

$$\tilde{T}_t\mathrm{e}^{\int_0^t \kappa(\xi)\mathrm{d}\xi} - \tilde{T}_{t-1}\mathrm{e}^{\int_0^{t-1} \kappa(\xi)\mathrm{d}\xi} = \int_{t-1}^t \mathrm{e}^{\int_0^s \kappa(\xi)\mathrm{d}\xi}\sigma(s)\mathrm{d}L_s$$

which leads to

$$\tilde{T}_t = \tilde{T}_{t-1}\mathrm{e}^{-\int_{t-1}^t \kappa(\xi)\mathrm{d}\xi} + \int_{t-1}^t \mathrm{e}^{-\int_s^t \kappa(\xi)\mathrm{d}\xi}\sigma(s)\mathrm{d}L_s$$

This equation will be the basis for the analysis in the following subsections. Different variations of this Ornstein–Uhlenbeck model will be investigated. We will compare cases where the mean reversion parameter is assumed to be constant as well as different distributions for the driving Lévy process.

6.1 Trend and seasonnality

The trend and seasonality are first modelled as a simple sinusoid with yearly frequency and a linear trend following the method used in Alaton(2002)

$$S_t = \alpha + \beta t + \gamma\sin(\omega t + \varphi)$$

With $\omega = \frac{2\pi}{365}$. The value of the parameters are found by minimizing the sum of square

$$\mathrm{argmin}_{\alpha,\beta,\theta,\lambda}\,\|T_t - S_t^*\|_2^2$$

for $S_t^* = \alpha + \beta t + \theta \sin(\omega t) + \lambda \cos(\omega t)$ and using the relations

$$\gamma = \sqrt{\theta^2 + \lambda^2} \quad \text{and} \quad \varphi = \arctan\left(\frac{\lambda}{\theta}\right) - \pi$$

Fitting the model to temperature in Stockholm, Sweden from 1960 to 2019, we find the following values for the parameters

α	β	γ	φ
6.18	9.83×10^5	10.18	-1.94

Table 6.1: Parameters for S_t with only 1 sine wave

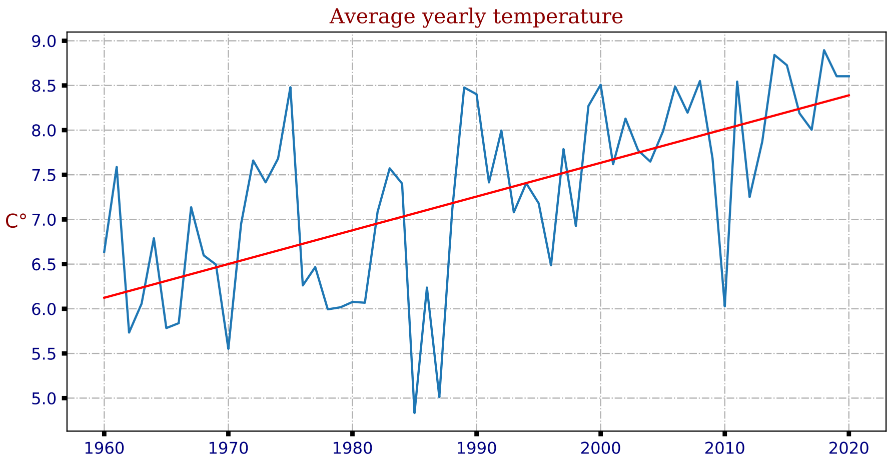


Figure 6.1: Yearly average temperature from 1960 to 2020

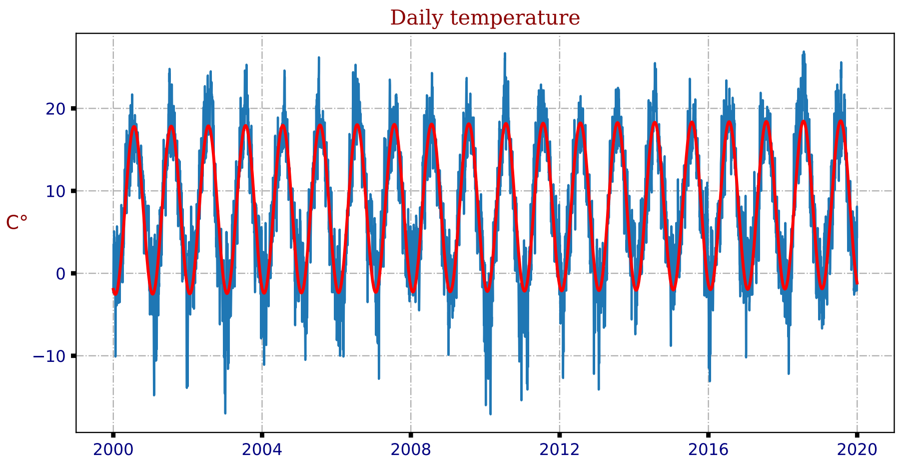


Figure 6.2: Daily temperatures (Blue) and fitted trend and seasonality (Red)

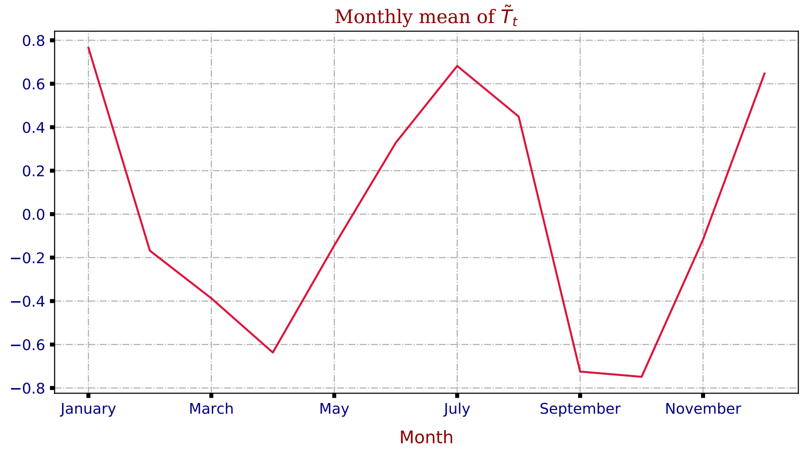


Figure 6.3: Monthly mean of the detrended and deseasonalized temperatures $\tilde{T}_t$

In order to investigate the possible existence of another cyclical component, we computed the discrete Fourier transform of the temperature. As we found a small spike at the angular frequency 2, we added another sine wave to our model. The model becomes

$$S_t = \alpha + \beta t + \theta_1 \sin(\omega t + \varphi_1) + \theta_2 \sin(2\omega t + \varphi_2)$$

We find the values of the parameters by minimizing

$$\text{argmin}_{\alpha,\beta,\theta_1,\varphi_1,\theta_2,\varphi_2} \|T - S\|_2^2$$

to find

α	β	θ_1	φ_1	θ_2	φ_2
6.18	9.84×10^5	10.18	-1.94	-0.77	29.62

Table 6.2: Parameters for S_t with 2 sine waves

As we can see in the following figures, the addition of the second sine wave greatly improves the model. Although we had a mean very close to zero for the entire time series of detrended temperatures, it was the result of misfits compensating each others. The monthly average detrended temperature in Figure 6.3 actually looks like a sine wave of frequency 2. For the improved model, we can see in Figure 6.5 that the average value for each months is much closer to zero. Looking at the average yearly value Figure 6.6, we see that it also seems to move randomly around 0, without any visible trend or pattern.

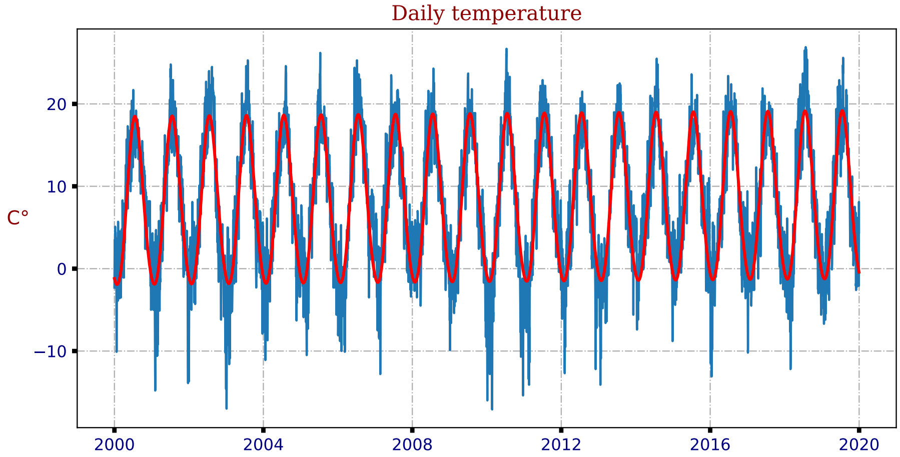


Figure 6.4: Daily temperatures (Blue) and fitted trend and seasonality (Red)

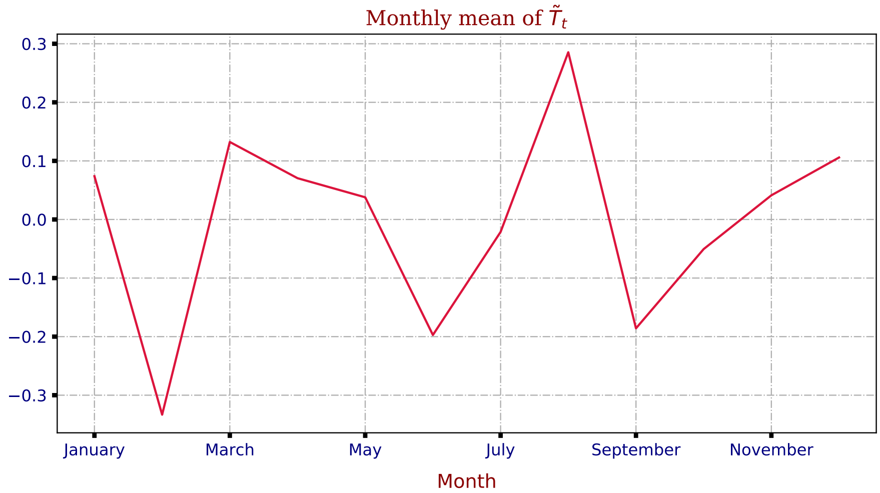


Figure 6.5: Monthly mean of the detrended and deseasonalized temperatures $\tilde{T}_t$

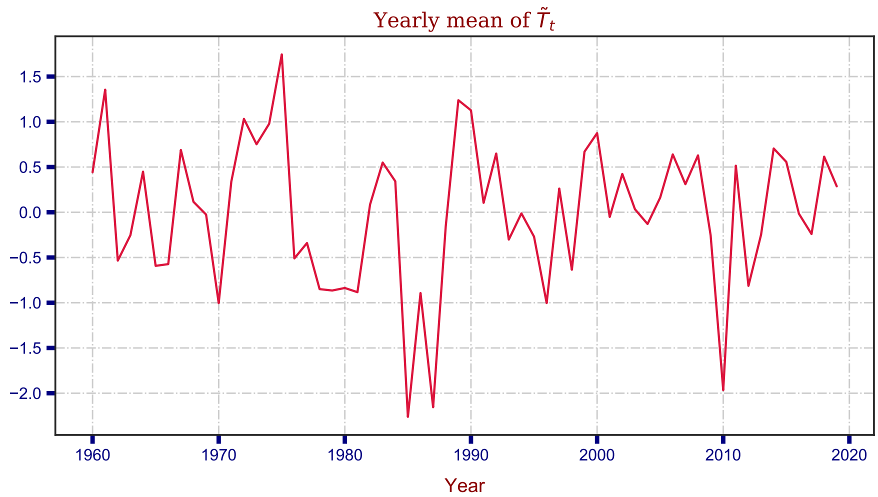


Figure 6.6: Yearly mean of the detrended and deseasonalized temperatures $\tilde{T}_t$

6.2 Modelling the speed of mean reversion

The objective of this subsection is to fit a function $\kappa(t)$ that best describes the speed at which the temperature T_t reverts to its predicted mean S_t. In other words, the speed at which the detrended and deseasonalized temperature $\tilde{T}_t$ returns to zero. For the model where the speed of mean reversion is allowed to depend on time, we make the assumption that it repeats the same annual cycle. This will be accomplished by restricting $\kappa(t)$ to a sum of sinusoids

$$\kappa(t) = \lambda + \sum_{i=1}^{N} \phi_i \sin(\gamma_i \omega t + \varphi_i) \text{ with } \quad \omega = \frac{2\pi}{365}$$

As $\sin\Big(\gamma_i \omega t + \varphi_i\Big)$ is a periodic function with period $\frac{365}{\gamma_i}$, restricting γ_i to be in the set of positive integers $\gamma_i \in \mathbb{Z}^+$ greater or equal to 1 ensures that $\kappa(t)$ will repeat a 1 year period. Observing the configuration designed earlier

$$\tilde{T}_t = \tilde{T}_{t-1} \mathrm{e}^{-\int_{t-1}^{t} \kappa(\xi)\mathrm{d}\xi} + \int_{t-1}^{t} \mathrm{e}^{-\int_{s}^{t} \kappa(\xi)\mathrm{d}\xi} \sigma(s)\mathrm{d}L_s$$

and assuming a similar structure for $\sigma(t)$, it is obvious that estimating the parameters by maximizing the (log-)likelihood function will be extremely complicated. The density function of a Lévy process, assuming it is available in closed form, generally depends on numerous parameters. Trying to estimate the parameters for the volatility and mean reversion functions on top of it may not be feasible. As a result, least-square methods will be used to find estimates of the parameters of $\kappa(t)$.

When the speed of mean reversion is assumed to be constant, the stochastic differential equation can be simplified to

$$\mathrm{d}\tilde{T}_t = -\kappa \tilde{T}_t \mathrm{d}t + \sigma(t)\mathrm{d}L_t$$

with solution

$$\tilde{T}_t = \tilde{T}_{t-1} \mathrm{e}^{-\kappa} + \int_{t-1}^{t} \mathrm{e}^{-\kappa(t-s)} \sigma(s)\mathrm{d}L_s$$

and the model can be reformulated as a discrete-time AR(1) process $\Big(\tilde{T}_t\Big)_{t\in\mathbb{N}}$

$$\tilde{T}_t = \rho \tilde{T}_{t-1} + Z^{\diamond}_{t-1}$$

where $\rho = \mathrm{e}^{-\kappa}$ and $Z^{\diamond}_{t-1} = \displaystyle\int_{t-1}^{t} \mathrm{e}^{-\kappa(t-s)} \sigma(s)\mathrm{d}L_s$.

Minimization of the sum of squares is achieved by solving

$$\frac{\partial}{\partial \rho} \sum_{t=1}^{N-1} \Big(\tilde{T}_{t+1} - \rho \tilde{T}_t\Big)^2 = 2 \sum_{t=1}^{n-1} \Big(\rho \tilde{T}_t^2 - \tilde{T}_t \tilde{T}_{t+1}\Big) = 0$$

such that we obtain a closed-form solution for the estimator

$$\hat{\rho} = \frac{\sum_{t=1}^{n-1} \tilde{T}_t \tilde{T}_{t+1}}{\sum_{t=1}^{n-1} \tilde{T}_t^2}$$

The estimator for κ can then be retrieved from $\hat{\kappa} = -\ln \hat{\rho}$. We find using this procedure the following

$\hat{\rho}$	$\hat{\kappa}$
0.801	0.222

Table 6.3: Constant mean reversion parameters

When the speed of mean reversion is time-dependent,the model can be rewritten as

$$\tilde{T}_t = \rho_{t-1}\tilde{T}_{t-1} + Z^{\diamond}_{t-1}$$

where $\rho_{t-1} = \mathrm{e}^{-\int_{t-1}^{t} \kappa(\xi)\mathrm{d}\xi}$ and $Z^{\diamond}_{t-1} = \int_{t-1}^{t} \mathrm{e}^{-\int_{s}^{t} \kappa(\xi)\mathrm{d}\xi}\sigma(s)\mathrm{d}L_s$.

Let

- $\Upsilon(\tau) = \int_{\tau}^{\tau+1} \kappa(\xi)d\xi = \lambda + \sum_{i=1}^{S} \Psi_{\phi_i,\gamma_i,\varphi_i}(\tau)$
- $\Psi_{\phi,\gamma,\varphi}(\tau) := \int_{\tau}^{\tau+1} \phi \sin(\gamma\omega\xi + \varphi)\,\mathrm{d}\xi = -\frac{\phi}{\gamma\omega}\Big[\cos(\gamma\omega\xi + \varphi)\Big]_{\xi=\tau}^{\xi=\tau+1}$

A faily smooth estimation of the periodic function $\kappa(t)$ can be obtained by the following procedure:

1. Splitting the observations in 365 groups, one for each day of the year.
2. Computing for each day $\hat{\rho}_t$ using the least square method
3. Retrieving $\hat{\kappa}_t = -\ln \hat{\rho}_t$.
4. Finding using least square the parameters $\mathrm{argmin}_{\lambda,\boldsymbol{\phi},\boldsymbol{\gamma},\boldsymbol{\varphi}} \sum_{t=1}^{365} \Big(\Upsilon(t) - \hat{\kappa}_t\Big)^2$ where $\boldsymbol{\phi}, \boldsymbol{\gamma}$ and $\boldsymbol{\varphi}$ are vectors of length S.

Setting S = 4 and $\boldsymbol{\gamma} = (1, 2, 3, 4)$, we find the following parameters for $\Upsilon(\tau)$

λ	ϕ_1	φ_1	ϕ_2	φ_2	ϕ_3	φ_3	ϕ_4	φ_4
0.2352	−0.0296	0.0649	−0.0368	0.2683	0.0163	1.601	−0.0007	0.246

Table 6.4: Parameter for the time-dependent speed of mean reversion

As we can see on Figure 6.7, the speed of mean reversion does not appear to be constant during the year. The red curve, which represent the sequence of fitted coefficient $\hat{\kappa}_t$ seems to fluctuate over the year. Though it is fairly volatile, it is obvious that it seems to form cluster above and below the blue line, which represents the speed of mean reversion had we assumed it was constant. Those fluctuations, though they may not appear important at first sight, reveal that the speed of mean reversion might be twice as fast during some periods compared to others. Assuming the speed of mean reversion to be constant may lead to significantly overestimating or underestimating its real value during the contract and may have dramatic effect on the price of the derivative.

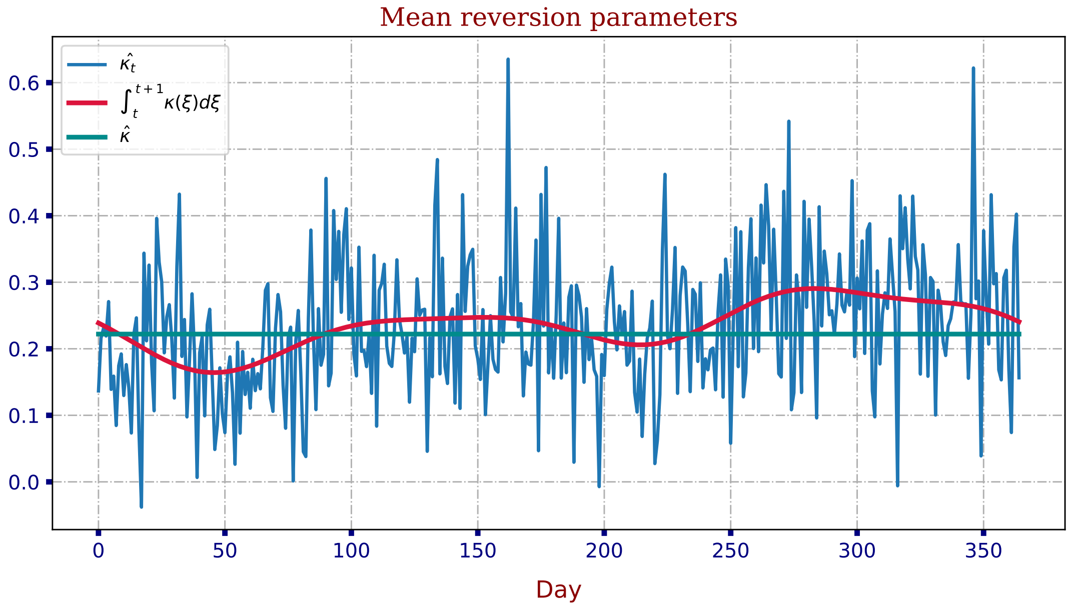


Figure 6.7: Plot of the fitted mean reversion function $\Upsilon(t)$

6.3 Modelling the seasonal volatility

As Lévy processes have independents and identically distributed increments from an infinite divisible distribution, the approximation

$$\int_t^{t+1} \mathrm{e}^{-\int_s^{t+1} \kappa(\xi)\mathrm{d}\xi} \sigma(s)\mathrm{d}L_s \approx \sigma(t) \int_t^{t+1} \mathrm{e}^{-\kappa(t)[t+1-s]}\mathrm{d}L_s$$
$$\stackrel{d}{=} \sigma(t)\mathrm{e}^{-\kappa(t)} \int_0^1 \mathrm{e}^{\kappa(t)\cdot s}\mathrm{d}L_s$$

is equivalent to approximate $\sigma(s)$ and $\kappa(s)$ as taking the value at $s = t$ and keeping them constant over the domain of integration. As $\sigma(t)$ and $\kappa(t)$ are functions that represent respectively the volatility and speed of mean reversion, it is unlikely that they will move significantly over the domain of integration which has a length of one day. The approximation is therefore assumed to be acceptable.

For any centered Lévy process with finite second moments and characteristic function

$$\Phi_{L_t}(z) = E\left[e^{izL_t}\right] = e^{\Psi(z)t}$$

it can be shown for a deterministic function $h(s)$ that (under some regularity conditions) the following stochastic integral

$$K_h(\eta, \tau) = \int_\eta^\tau h(s)\mathrm{d}L_s$$

exists and has a characteristic function $\Phi_{K(a,b)}(z)$ such that

$$\log \Phi_{K_h(\eta,\tau)}(z) = \int_\eta^\tau \Psi\Big(zh(y)\Big)\mathrm{d}y$$

If we apply this result to the stochastic integral $K_h(0,1) = \displaystyle\int_0^1 e^{\kappa s} dL_s$, we find

$$\log \Phi_{K_h(0,1)}(z) = \int_0^1 \Psi\Big(e^{\kappa y} z\Big)\mathrm{d}y$$

Using the change of variable $w = e^{\kappa y} z$ with $\mathrm{d}y = (w\kappa)^{-1}\mathrm{d}w$, we have

$$\int_0^1 \Psi\Big(e^{\kappa y} z\Big)\mathrm{d}y = \kappa^{-1} \int_z^{e^\kappa z} \frac{\Psi(w)}{w} dw$$

For example, applying this result to a Brownian motion W_t with $\Psi(z) = -z^2/2$, yields

$$\kappa^{-1} \int_z^{e^\kappa z} \frac{-w^2/2}{w} dw = \frac{-z^2}{2}\left[\frac{e^{2\kappa} - 1}{2\kappa}\right]$$

which is the characteristic function of a gaussian variable with variance $\frac{e^{2\kappa}-1}{2\kappa}$. The same result can be found using Itô isometry

$$\int_0^1 e^{\kappa s} dW_s \sim \mathcal{N}\left(0, \int_0^1 e^{2\kappa s}\mathrm{d}s\right) \quad \text{with } \int_0^1 e^{2\kappa s}\mathrm{d}s = \left[\frac{\mathrm{e}^{2\kappa s}}{2\kappa}\right]_0^1 = \frac{e^{2\kappa} - 1}{2\kappa}$$

If the Lévy process L_t is chosen to be a Brownian motion W_t, the model then becomes

$$\tilde{T}_{t+1} - \tilde{T}_t e^{-\int_t^{t+1} \kappa(\xi)\mathrm{d}\xi} \approx e^{-\kappa(t)}\sigma(t) \int_0^1 e^{\kappa(t)\cdot s} dW_s \approx \tilde{\sigma}(t)\Delta W_t$$

with $\Delta W_t \sim \mathcal{N}(0,1)$ and $\tilde{\sigma}(t) = \sigma(t)\left(\dfrac{1 - \mathrm{e}^{-2\kappa(t)}}{2\kappa(t)}\right)^{\frac{1}{2}}$

In the litterature, κ is generally assumed to be constant and the following approximation is generally made

$$\mathrm{e}^{-\kappa} \int_t^{t+1} \sigma(s)\mathrm{e}^{-\kappa(t-s)}\mathrm{d}L_s \approx e^{-\kappa}\sigma(t)\Delta L_t$$

This approximation may not be reasonnable depending on the value of κ. In the case of Brownian motion for example, approximating

$$\int_t^{t+1} \sigma(s)\mathrm{e}^{-\kappa(t+1-s)}\mathrm{d}W_s \overset{d}{\approx} \sigma(t)\mathrm{e}^{-\kappa} \int_0^1 \mathrm{e}^{\kappa s}\mathrm{d}W_s \overset{d}{=} \sigma(t)\left(\frac{1 - \mathrm{e}^{-2\kappa}}{2\kappa}\right)^{\frac{1}{2}} \Delta W_t$$

by $e^{-\kappa}\sigma(t)\Delta W_t$ is equivalent to assume that $e^{-\kappa} \approx \left(\dfrac{1 - \mathrm{e}^{-2\kappa}}{2\kappa}\right)^{\frac{1}{2}}$

Another approximation could be to take the mean value of $\mathrm{e}^{\kappa s}$ over the interval

$$\int_0^1 \mathrm{e}^{\kappa s} ds = \frac{e^{\kappa}-1}{\kappa} \text{ to yield the approximation } \sigma(t)\frac{e^{\kappa}-1}{\kappa}\Delta W_t$$

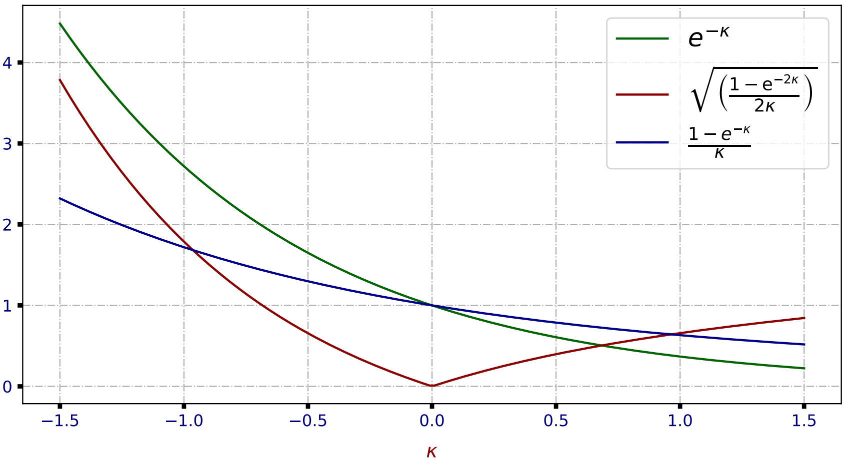


Figure 6.8: Comparaison of the different approximations for varying κ

In any case, it is obvious from Figure 6.8 that these approximations are potentially very unreliable. The results found using the isometry are best in the case of Brownian motions.

Though the previous results can be used to find the exact distribution of the stochastic integral and possibly use likelihood-based methods for find the parameters for $\sigma(t)$ and the Lévy process, it will hardly be feasible in our case, with complicated function $\sigma(t)$ and sophisticated Lévy process. We will choose consequently to embark on another path.

6.3.1 Modelling volatility assuming Brownian increments

If we assume a the Lévy process L_t to be a Brownian Motion W_t, then using

$$\begin{aligned} Z_t^{\diamond} = \tilde{T}_{t+1} - \tilde{T}_t \mathrm{e}^{-\int_s^{t+1}\kappa(\xi)\mathrm{d}\xi} &= \int_t^{t+1} \mathrm{e}^{-\int_s^{t+1}\kappa(\xi)\mathrm{d}\xi}\sigma(s)\mathrm{d}L_s \\ &\overset{d}{\approx} e^{-\kappa(t)}\sigma(t)\int_0^1 e^{\kappa(t)\cdot s} dW_s \end{aligned}$$

and Itô's isometry we can write the following

$$Z_t^{\dagger} := Z_t^{\diamond} \cdot \left(\frac{1-\mathrm{e}^{-2\kappa(t)}}{2\kappa(t)}\right)^{-\frac{1}{2}} \sim \mathcal{N}\left(0, \sigma^2(t)\right)$$

From there on, we can find a smooth estimator for $\sigma^2(t)$ using a similar method and form as for the mean reversion.

$$\sigma^2(t) = \lambda + \sum_{i=1}^{S} \phi_i \sin(\gamma_i \omega t + \varphi_i)$$

We keep the same restrictions on $\sigma^2(t)$ as for $\kappa(t)$ to ensure that the volatility repeats a yearly cycle. The procedure in this case is

1. Splitting the observations in 365 groups , one for each day of the year.
2. Computing for each day the mean of the squared $Z_t^\dagger$:

$$\hat{\sigma}_\tau^2 = \frac{\sum_{t=1}^N Z_t^{\dagger 2} \mathbb{1}_{\{t \bmod 365 = \tau\}}}{\sum_{t=1}^N \mathbb{1}_{\{t \bmod 365 = \tau\}}} \quad \text{for } \tau = 0, 1, \cdots, 364$$

3. Finding using least square the parameters $\operatorname{argmin}_{\lambda,\boldsymbol{\phi},\boldsymbol{\gamma},\boldsymbol{\varphi}} \sum_{t=1}^{365} \left(\sigma^2(t) - \hat{\sigma}_t^2\right)^2$ where $\boldsymbol{\phi}, \boldsymbol{\gamma}$ and $\boldsymbol{\varphi}$ are vector of length S.

Setting S = 4 and $\boldsymbol{\gamma} = (1, 2, 3, 4)$, we find the following parameters for $\sigma^2(t)$:

λ	ϕ_1	φ_1	ϕ_2	φ_2	ϕ_3	φ_3	ϕ_4	φ_4
5.536	2.267	−1.25	−1.480	−1.1534	0.6464	0.823	−0.0513	−0.97

Table 6.5: Parameters for $\sigma^2(t)$ assuming Brownian increments

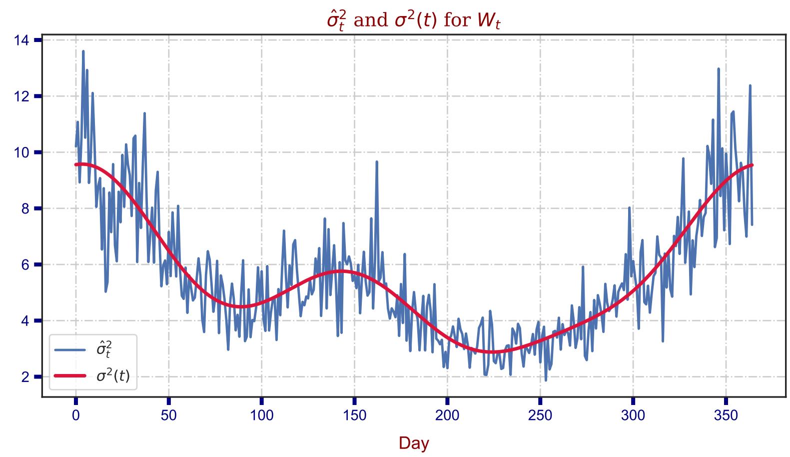


Figure 6.9: Comparison of the daily squared residuals $\hat{\sigma}_\tau^2$ and fitted variance function $\sigma^2(t)$

Finally, we can extract our increments under the assumption that L_t is a Brownian motion by dividing the stochastic integrals $Z_t^\diamond$ by their standard deviations

$$\Delta W_t = \frac{Z_t^\diamond}{\sigma(t)} \cdot \left(\frac{1 - \mathrm{e}^{-2\kappa(t)}}{2\kappa(t)}\right)^{-\frac{1}{2}}$$

Figure 6.10 shows that seasonnalities in the residuals were close to completely removed and we are left with a centered noise without discernible pattern in either the daily mean of the estimated Brownian increment as well as their squared counterpart. This is shown in Figures 6.11 and 6.12.

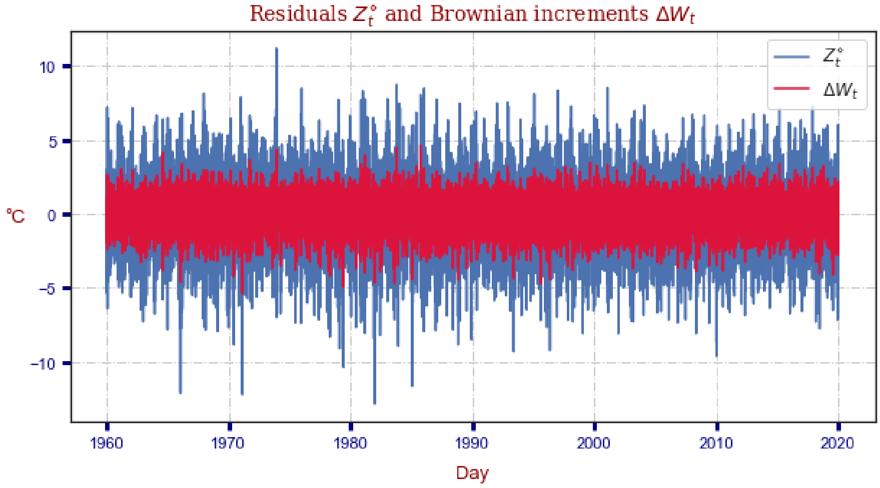


Figure 6.10: Comparison of the residuals $Z_t^\diamond$ and Brownian Increments ΔW_t

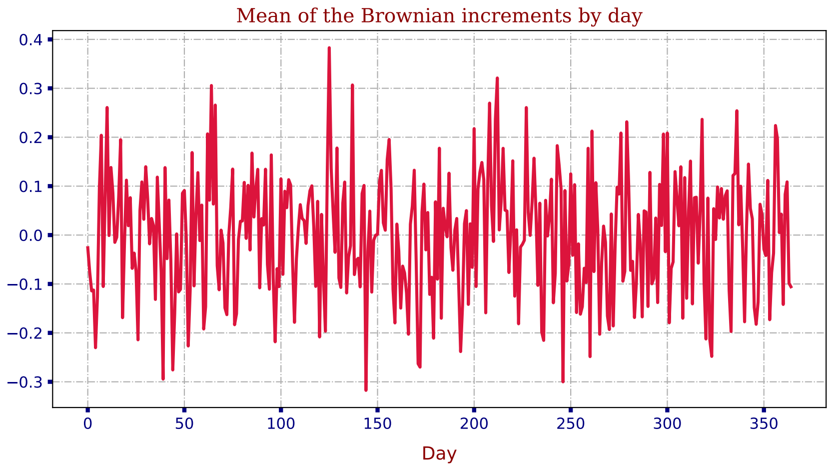


Figure 6.11: Daily Mean of the estimated Brownian increments ΔW_t

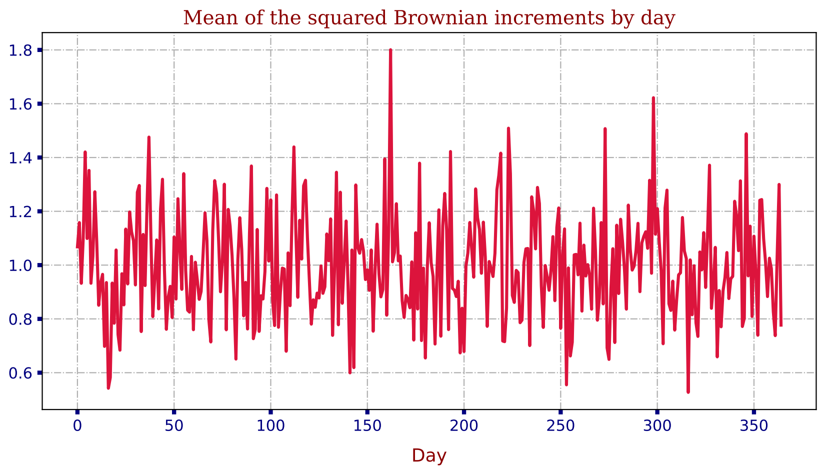


Figure 6.12: Daily Mean of the squared estimated Brownian increments

6.3.2 Modelling volatility for general Lévy increments

Let

$$K_\psi(0,t) := \int_0^t \psi(s)\mathrm{d}L_s$$

If we put restrictions on the parameters of the Lévy process L_t with triplet (σ, ν, γ) to ensure that it is a martingale, for example with zero mean, a property similar to Itô's isometry can be used and $K_\psi(0,t)$ is a square integrable martingale with

$$\mathrm{E}\left[K_\psi(0,t)\right] = 0 \quad \text{and} \quad \mathrm{E}\left[K_\psi(0,t)^2\right] = \mathrm{E}\left[\int_0^t \psi^2(s)d[L,L]_s\right]$$

where

$$\mathrm{E}\left[\int_0^t \psi^2(s)d[L,L]_s\right] = \int_0^t \sigma^2\psi^2(s)ds + \mathrm{E}\left[\int_0^t \int_{\mathbb{R}} x^2\psi^2(s)\Pi(ds,dx)\right]$$

In the absence of Brownian component ($\sigma = 0$), the variance of the stochastic integral is simply the expectation of the square of the sum of products

$$\mathrm{Var}\left[K_\psi(0,t)\right] = \mathrm{E}\left[K_\psi(0,t)^2\right] = E\left[\sum_{s\le t}\Big(\psi(s)\Delta L_s\Big)^2\right]$$

This result and the basic property of variance $\mathrm{Var}(aX) = a^2\mathrm{Var}(X)$ will be the basis for our method. Integrating the Ornstein–Uhlenbeck stochastic differential equation

$$d\tilde{T}_t = -\kappa(t)\tilde{T}_t\mathrm{d}t + \sigma(t)\mathrm{d}L_t$$

we find

$$\begin{aligned}\tilde{T}_{t+\Delta_t} - \tilde{T}_t &= -\int_t^{t+\Delta_t} \kappa(s)\tilde{T}_s\mathrm{d}s + \int_t^{t+\Delta_t} \sigma(s)\mathrm{d}L_s \\ &\approx -\kappa(t)\int_t^{t+\Delta_t} \tilde{T}_s\mathrm{d}s + \sigma(t)\int_t^{t+\Delta_t} \mathrm{d}L_s\end{aligned}$$

such that, if the process X_t is observed continuously, we can then isolate the increment

$$\sigma(t)\,\Delta L_t := \sigma(t)\Big[L_{t+\Delta_t} - L_t\Big] \approx \left[\tilde{T}_{t+\Delta_t} - \tilde{T}_t + \kappa(t)\int_t^{t+\Delta_t} \tilde{T}_s\mathrm{d}s\right]$$

When the process is observed discretely, we can resort to a trapezoidal approximation of the integral

$$\sigma(t)\,\Delta L_t \approx \left[\tilde{T}_{t+\Delta_t} - \tilde{T}_t + \kappa(t)\frac{\tilde{T}_{t+\Delta_t} - \tilde{T}_t}{2}\Delta_t\right]$$

As we have daily observation of the temperature process, we use the previous result with $\Delta_t = 1$ to extract our approximation of the seasonalized increments $\sigma(t)\Delta L_t$.

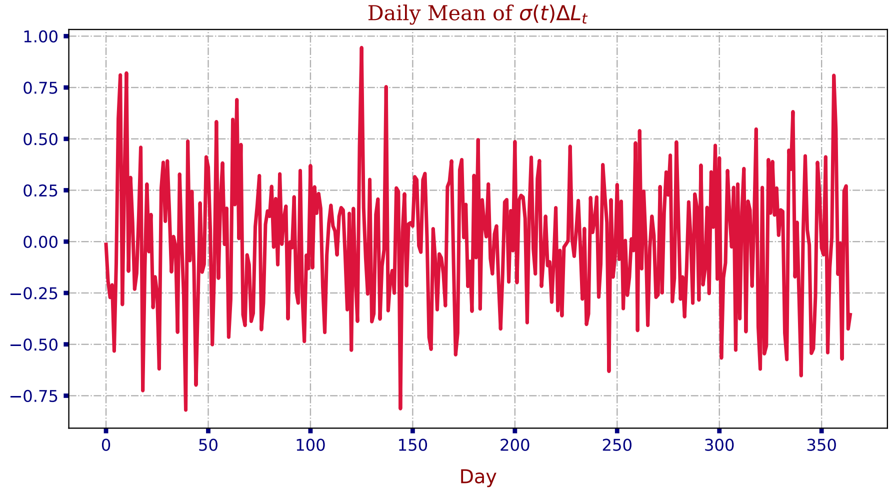


Figure 6.13: Daily Mean of the residuals $Z_t^{\diamond}$

Assuming ΔL_t to be centered, we can generate a smooth approximation of $\sigma^2(t)$ in the general Lévy case using the same method as for the Brownian increment. We set this time $Z_t^{\dagger} = \sigma(t)\,\Delta L_t$ and find the following values for the parameter of $\sigma^2(t)$.

λ	ϕ_1	φ_1	ϕ_2	φ_2	ϕ_3	φ_3	ϕ_4	φ_4
6.137	2.461	1.265	-1.642	-1.111	0.69	0.816	-0.0856	-0.7977

Table 6.6: Parameters for $\sigma^2(t)$ assuming Lévy increments

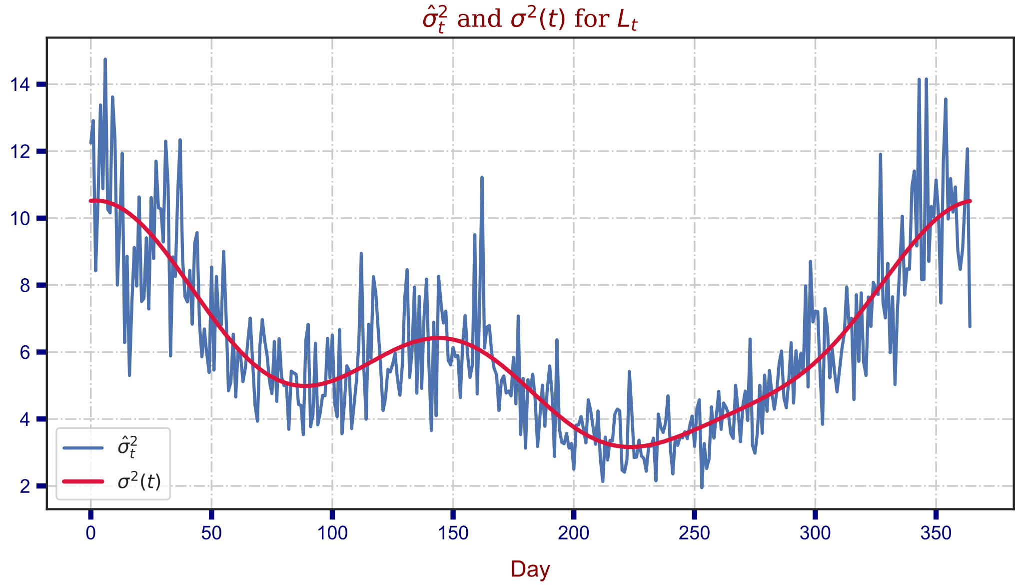


Figure 6.14: Comparison of the daily squared residuals $\hat{\sigma}_\tau^2$ and fitted variance function $\sigma^2(t)$

We can observe in the following figures 6.15, 6.16 and 6.17 results similar to the results obtain for the normal distribution, namely that we obtain seemingly centered Lévy increments with no discernible time-dependence in the variance or the mean.

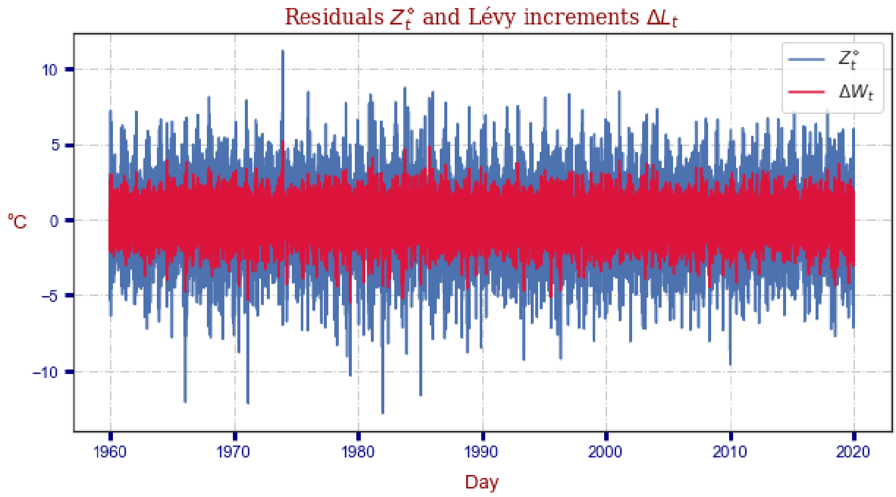


Figure 6.15: Comparison of the residuals $Z_t^\diamond$ and Lévy Increments ΔL_t

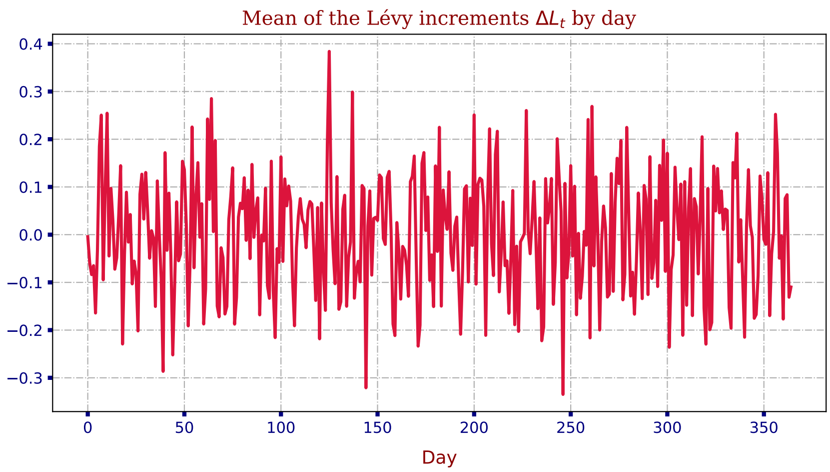


Figure 6.16: Mean of the Lévy increments ΔL_t by day

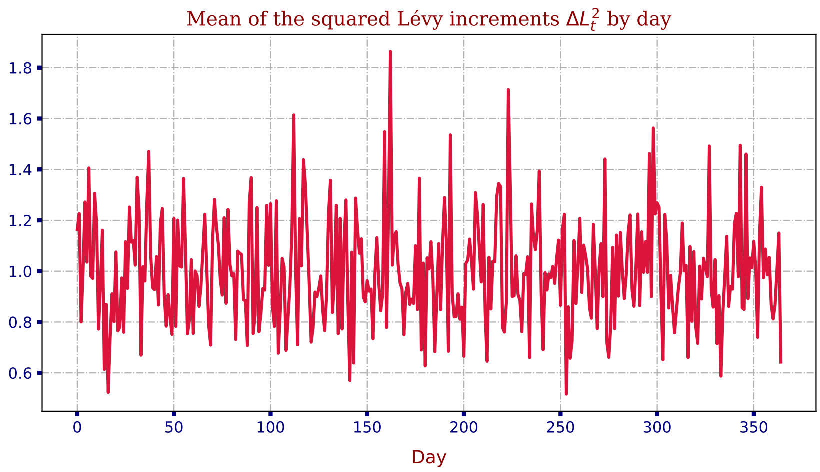


Figure 6.17: Mean of the squared Lévy increments ΔL_t^2 by day

6.4 Modelling the distribution of the residuals

It is obvious from Figure 6.18 that the Normal distribution from the Brownian motion provides a very bad fit for the extracted Brownian increments, especially at the lower tails. Though models based on Brownian motion are generally more tractable, and much much likely to lead to analytical formulas, the rigid normal distribution is not appropriate to use in our case.

Generalized Hyperbolic distributions from Figures 6.21 and 6.20 provide a much better fit. In the following section, we will try to demonstrate the importance of a proper calibration of the random component of the models. Using a ill-fitted distribution means in practice that the probabilities of occurrence some events may be drastically different from expected. This can have devastating effects in the context of finance or insurance.

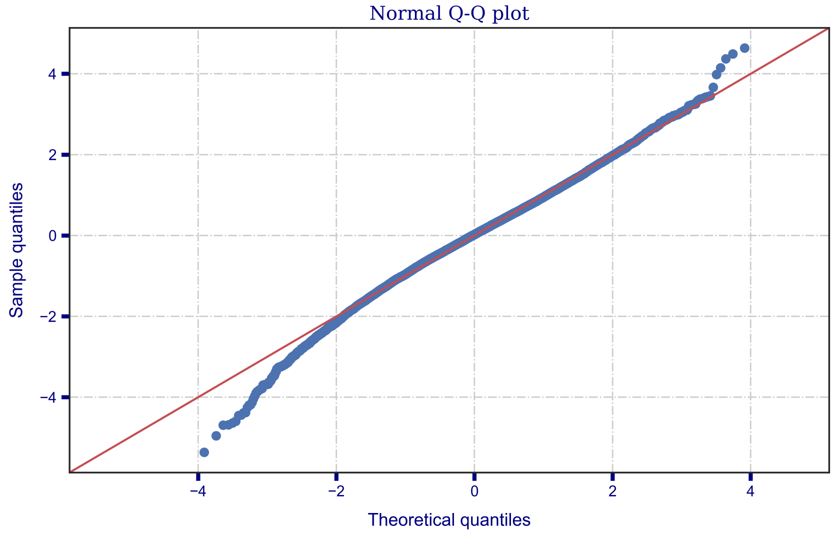


Figure 6.18: Normal QQ-Plot of the Brownian increments

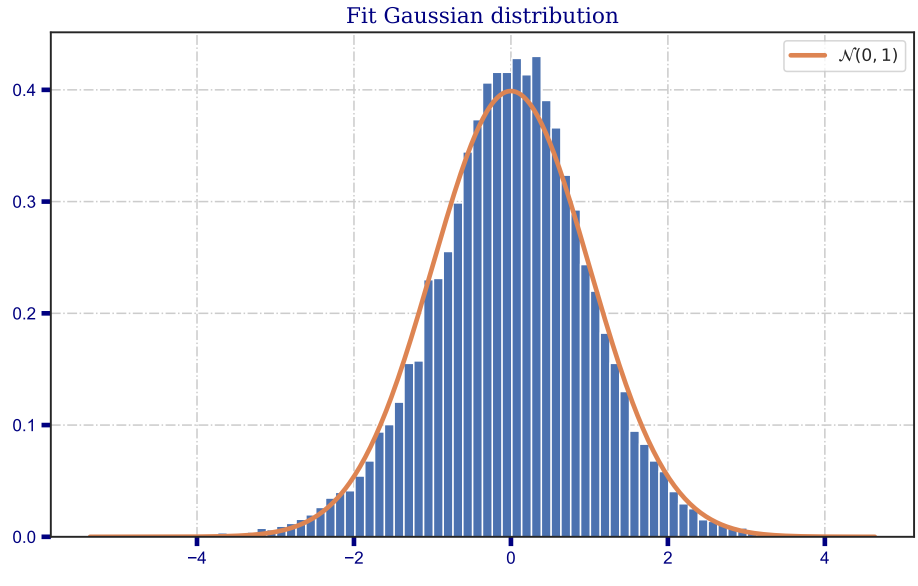


Figure 6.19: Density Plot of the Brownian residuals compared to Normal distribution

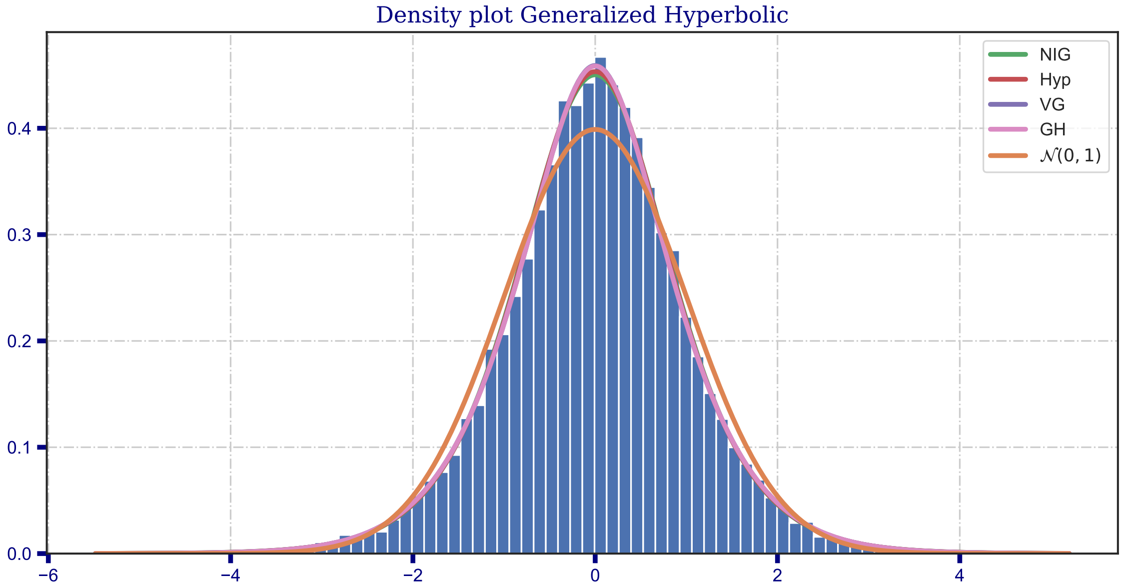


Figure 6.20: Density Plot of the Lévy residuals compared to Generalized Hyperbolic distributions

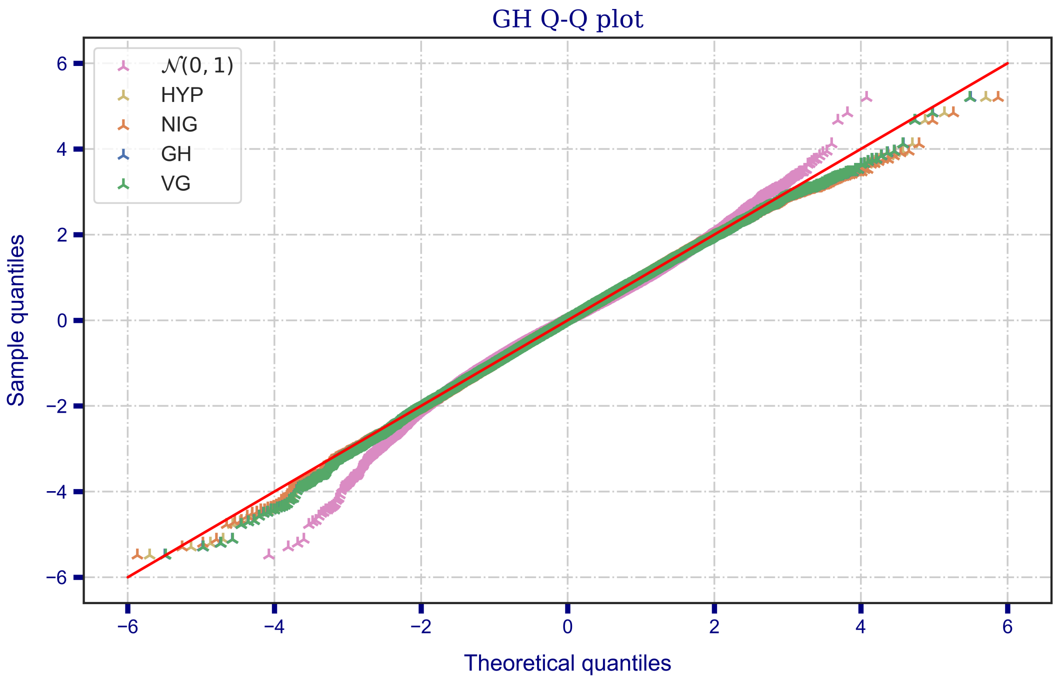


Figure 6.21: QQ-Plot Lévy increments

Below can be found the parameters for the Generalized Hyperbolic distributions. The R package GHYP[3] was used for the calibration.

	λ	α	β	δ
GH	2.954556	2.442374	0.000119	0.190832
NIG	-0.5	1.618664	-0.000062	1.621384
VG	3.021098	2.459503	-0.000062	0
HYP	1	1.985415	0.000010	1.173791

Table 6.7: Parameters for the fitted Generalized Hyperbolic distributions

6.5 Analysis of the autocorrelation of the residuals

Under both hypothesis regarding the distributions of the residuals, we are left with some autocorrelations in the residuals, as shown in Figure 6.22. This mean that our models can still be improved in some ways. It remains to discover how and if it will even be worth the trouble.

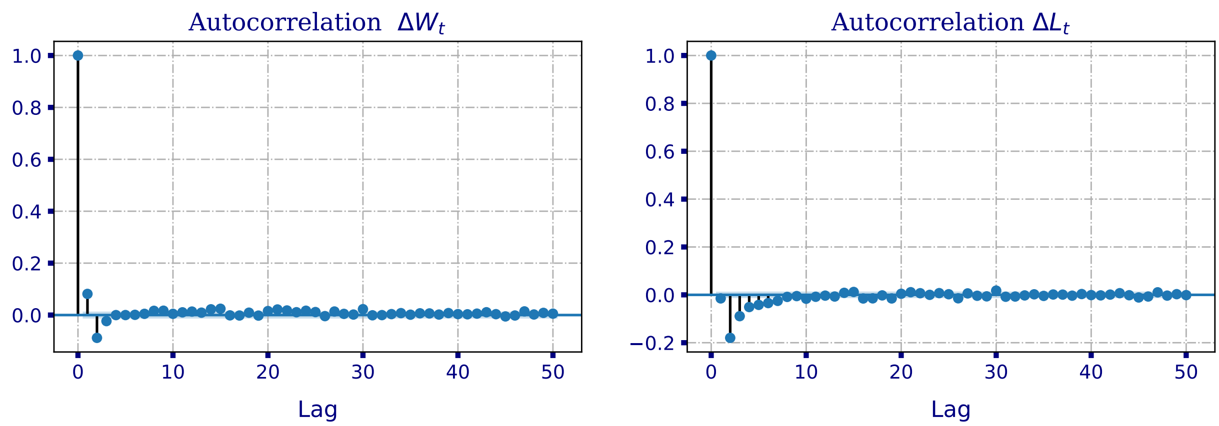


Figure 6.22: Autocorrelation of the Brownian and Lévy increments.

It is also important to note that the autocorrelation structure in Figure 6.22 does not provide evidence for the presence of fractionality in the residuals. This seems to indicate that a fractional Brownian motion (fBm) as in Brody (2002) is not appropriate in our model.

[3]https://cran.r-project.org/web/packages/ghyp/index.html

7 Pricing Temperature Derivatives

7.1 Common Weather derivatives products on the CME

Contracts based on CAT indexes: In Europe, contracts for the summer months are generally based on the CAT index. The value of this index is the sum of the daily average temperature (DAT) over the contract period, measured as the simple average of the minimum and maximum temperature for each day. In London, one CAT index future contract pays off £20 per index point, while it pays off €20 per unit in all other European locations. These contracts usually have a monthly or seasonal duration.

Contracts based on HDD or CDD indexes: In the USA, Canada, and Australia, HDD or CDD indexes are the norms. A HDD is the number of degrees by which the daily temperature is below a base temperature, and a CDD is the number of degrees by which the daily temperature is above the base temperature.

- Daily HDD = max(0, base temperature − daily average temperature)
- Daily CDD = max(0, daily average temperature − base temperature)

The base temperature is usually 65 °F in the USA and 18 °C in Europe and Japan. These contracts also usually have a monthly or seasonal duration.

Generally, weather derivatives are either a vanilla options or futures based on these indices. A Future in the context of weather derivative will pay at maturity the value of the underlying index, regardless of its value. An option will pay a non negative value, with the following payoff structures depending on the type of option

- CALL = max(0, Index Value at maturity − Strike)
- PUT = max(0, Strike − Index Value at maturity)

7.2 Risk Neutral Pricing of Temperature Derivatives

A general framework to compute the fair value of an insurance contract is to derive the (discounted) expected payoff or compensation perhaps increased by various fees. In the context of exchange-traded financial product however, the fair value of contingent claims is established under the framework of the fundamental theorems of asset pricing : the fair value of a derivative contract in a complete and arbitrage free market is the expected value of the future payoff under the unique risk-neutral measure (or equivalent martingale measure) discounted at the risk-free rate. This implies that the value of the derivative contract is a martingale. Assuming $\pi_t(B)$ to be the value of the contingent claim at time t which gives a payoff of B at maturity T, we have

$$\pi_t(B) = E^Q\Big[e^{-rT}B|\mathcal{F}_t\Big]$$

However, as the underlying of a weather derivative cannot be perfectly stored and/or traded, and the payoffs cannot be perfectly replicated with existing products, the market is incomplete. As a result, assuming the market is arbitrage-free, there may be a plurality equivalent risk neutral measure and therefore the price of the derivative is no longer unique but rather a range of prices that would be derived under the different equivalent martingale measures.

$$\pi_t(B) \in \left[\inf_{Q\in\mathcal{M}} E^Q\left[e^{-rT}B|\mathcal{F}_t\right], \sup_{Q\in\mathcal{M}} E^Q\left[e^{-rT}B|\mathcal{F}_t\right]\right]$$

where $\mathcal{M}$ is the set of all equivalent martingale measures.

The presence of possible jumps in the value of the underlying, a feature of some of the models considered in this work may also lead to market incompleteness as some risks cannot be hedged, even in continuous time. For those models, in the context of derivative products whose underlying can be partially hedged/replicated, its value will be the cost of the replicating strategy plus an additional risk premium to compensate for the residual risk. In the context of weather derivatives however, one may question the existence of reliable hedging strategies, even for partial hedging.

Another option in theory would be to calibrate the model to market prices though this may not be feasible as realistic models for the dynamics of temperatures require a sizable number of parameters and market data for the generally illiquid weather derivatives is unlikely to be sufficient.

Finally, it is also pertinent to do a pragmatic assessment of the complexity of potential changes of measures relating to our complex temperatures models. In light of the previous arguments and in order to reduce the scope of this work, the decision as been made, without loss of generalization, to analyze the value of the derivatives under our models under the real measure, without losing sight that those methods can be extended under to other appropriate probability measures.

7.3 Change of measure and analytical pricing

In Benth(2005) and Benth(2007), a sub-family of probability measures is detailled using the Esscher transform defined by

$$\frac{dQ^\theta}{dP} = Z^\theta(\tau_{\max})$$

where $\tau_{\max}$ is a fixed time horizon including the trading time for all relevant futures and

$$Z^\theta(t) = \exp\left(\int_0^t \theta(s)\mathrm{d}L(s) - \int_0^t \phi(\theta(s))\mathrm{d}s\right)$$

with $\theta(t)$ a real-valued measurable and bounded function expressing the time-varying market price of risk and $\phi(\lambda)$ is the logarithm of the moment generating function of L_1. If the Lévy process L is a Brownian motion, then the Esscher transform reduces to a Girsanov change-of-measure

$$Z^\theta(t) = \exp\left(\int_0^t \frac{\theta(s)}{\sigma(s)}\mathrm{d}B(s) - \frac{1}{2}\int_0^t \frac{\theta^2(s)}{\sigma^2(s)}\mathrm{d}s\right)$$

A number of analytical formula are developed in Benth(2007) for the pricing of common weather derivative helped greatly by the following assumptions.

For Contracts based on CAT indexes on a time interval $[\tau_1, \tau_2]$, the value of the underlying index is assumed to be

$$\text{CAT}(\tau_1, \tau_2) = \int_{\tau_2}^{\tau_2} T_s ds$$

while for their HDD or HDD counterparts, the value is assumed to be

$$\text{HDD} = \int_{\tau_1}^{\tau_2} \max[c - T_s, 0] ds \qquad \text{and} \qquad \text{CDD} = \int_{\tau_1}^{\tau_2} \max[T_s - c, 0] ds$$

The price of the derivative is then simply computed as the discounted expectation of the pay-offs under the appropriate probability measure. Though a closed-from solution is derived for models driven by Brownian motions, one can argue that the area under the curve for a particular day may differ from the pre-established method for computing the average, namely the average of the highest and lowest temperature recorded for that day. Besides, these closed-form equations are not derived for models driven by more general Lévy processes.

Consequently, we will use Monte-Carlo methods to benefit from more accurate and versatile techniques to price weather derivatives on the models derived from the previous sections.

7.4 Monte-Carlo pricing of weather derivatives

Computing the expected pay-offs of derivatives based on our models is not a trivial task. As seen in the previous sections, deriving the probability density function of the stochastic integral can prove to be very challenging, if possible at all. Adding a complex payoff structure on top of it makes the task significantly more difficult. Monte Carlo simulation methods provide a convenient and reliable solution to this issue. Knowing the distribution of every random component of a complex system, we can generate an approximation of the probability density function of the entire system by sampling instances of the system and observing the empirical distribution. Applied to our weather derivative problem, we can approach the expected payoff by sampling a large number N of temperature paths T_i, computing their payoffs $F(T_i)$ and then taking the empirical average of all the payoff samples. This is simply a result from the law of large numbers.

$$\frac{e^{-rT}}{N} \sum_{i=1}^{N} F(T) \underset{a.s.}{\longrightarrow} E\Big[e^{-rT} F(T_i)\Big]$$

The Ornstein–Uhlenbeck part of the model can easily be generated

$$d\tilde{T}_t = -\kappa(t)\tilde{T}_t \mathrm{d}t + \sigma(t)\mathrm{d}L_t$$

using the Euler–Maruyama discretization of its stochastic differential equation

$$\tilde{T}_{t+\Delta_t} = \tilde{T}_t - \kappa(t)\tilde{T}_t\Delta_t + \sigma(t)\Delta L_t$$

with $\Delta L_t = L_{t+\Delta_t} - L_t$ and then adding the deterministic trend component $T_t = \tilde{T}_t + S_t$. Assuming we chose a Variance Gamma process as the driving noise, then $\Delta L_t \sim \text{VG}_t \sim \text{VG}\left(\lambda\Delta_t, \alpha, \beta\right)$. Samples from that distribution can be generated by sampling N from

$\mathcal{N}(0,1)$ and X from Gamma $\left(\lambda\Delta_t, \frac{1}{2}(\alpha^2 - \beta^2)\right)$ and computing

$$\Delta L_t \sim \beta N + \sqrt{X} N$$

The equivalent for the Normal Inverse Gaussian Distribution is generated by sampling X from a IG $\left(\delta\Delta_t, \sqrt{\alpha^2 - \beta^2}\right)$. In the case of the Brownian motion, the increments follows a standard normal distribution $\Delta L_t \sim \mathcal{N}(0, \sqrt{\Delta_t})$.

The goal for the rest of this section will be to bring to light the importance of allowing the speed of mean reversion to depend on time and modeling the noise with distributions that are more flexible than the normal distribution. To do that the empirical distribution of a panel of futures and options on the different indices and over different period will be analyzed. The four following models will be compared in this section

- Model 1 : Time-dependent speed of mean reversion with Variance Gamma process.
- Model 2 : Constant speed of mean reversion with Variance Gamma process.
- Model 3 : Time-dependent speed of mean reversion with Brownian Motion.
- Model 4 : Constant speed of mean reversion with Brownian Motion.

Note: For the products considered in the following subsections, the density will be approximated by Monte Carlo simulation using 200.000 simulations and a discretization step of $\Delta_t = \frac{1}{100}$. We will assume for the sake of simplicity that the risk-free rate r is 0.

7.4.1 CAT indices

During the month of February 2019, the time-dependent value of the mean reversion function $\kappa(t)$ is lower that the constant value measured in the other models.

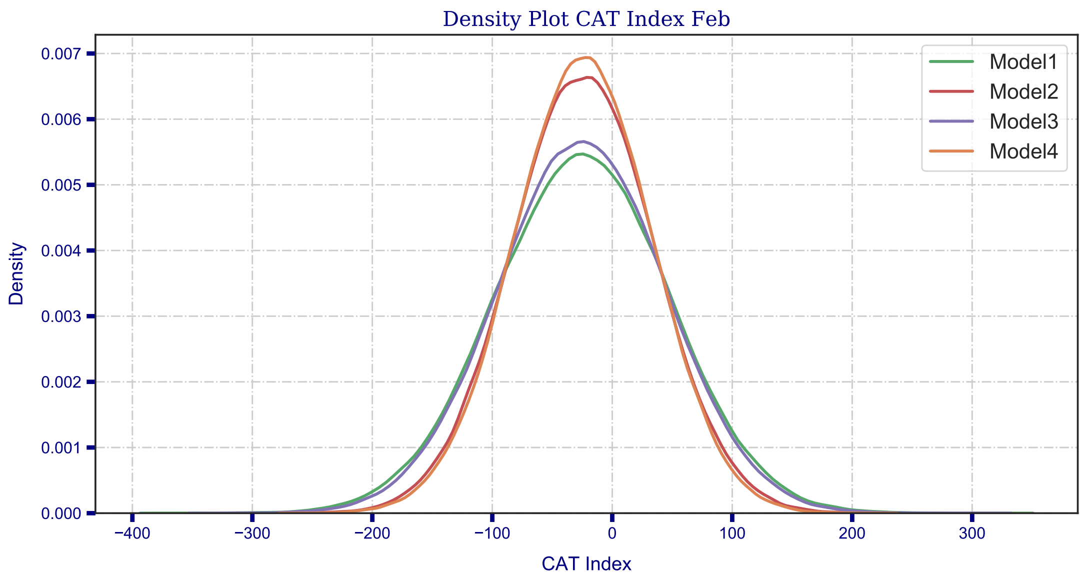


Figure 7.1: Density plot of the CAT Index value of February 2019

As we can see in Figure 7.1, the empirical distributions of the models 2 and 4 with constant speed of mean reversion are much more concentrated around the mean whereas the models with the time-dependent speed of mean reversion have much heavier tails. The opposite can be observed for the value of the CAT index measured from September to November 2019 in Figure 7.2.

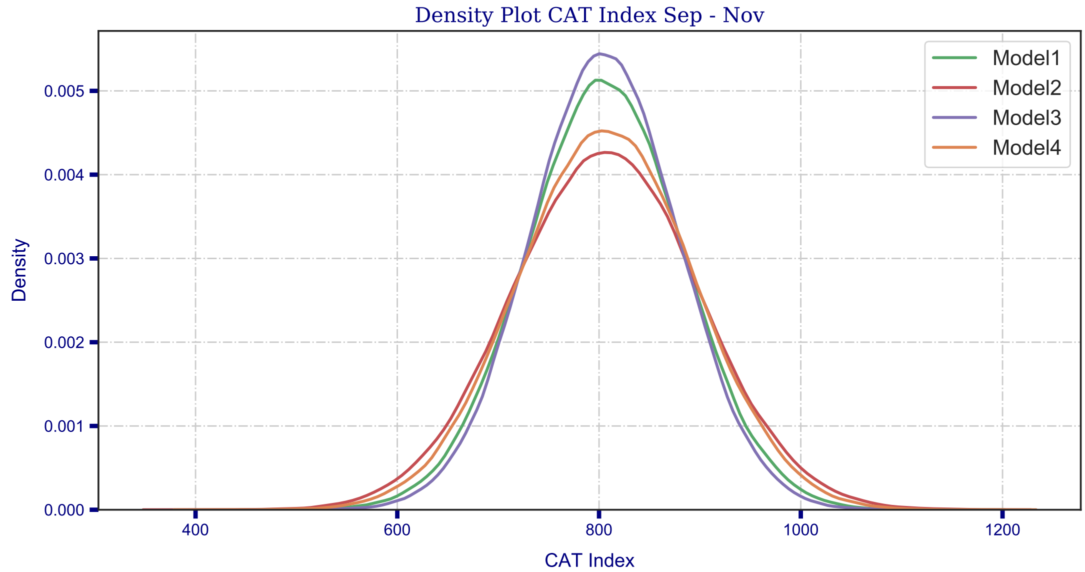


Figure 7.2: Density plot of the CAT Index value of September to November 2019

Here, the models with constant speed of mean reversion have the heavier tails. It is also important to note that for both products, the distributions of payoffs of Variance Gamma based models is more slightly more spread out as this distribution allows for more extreme events.

Though the Future for the CAT indices have close to identical prices across all models for both measurement periods, as shown in the table below, the differences in the shape of the distributions of the payoffs can make a big difference from a risk management perspective. Using an inappropriate model can dramatically overestimate or underestimate the likelihood of extreme result.

	Model 1	Model 2	Model 3	Model 4
Feb CAT	-25.12	-24.35	-25.19	-24.1617
Sep - Nov CAT	805.74	806.79	805.52	807.54

Table 7.1: Prices of the Future on CAT indices for Feb and Sep - Nov 2019

Moreover, the four models may lead to significantly differences in prices in the context of options, depending on the period and strike prices. A good illustration is call options on the CAT index of February 2019 valued over a range of positive strikes shown in Figure 7.3. As a large part of the distribution of the CAT index value is negative, and options ensure non negative payoffs, the models with time dependent speed of mean reversion benefits from their flatter distributions and heavier tails. The value of the options for those models is remarkably higher.

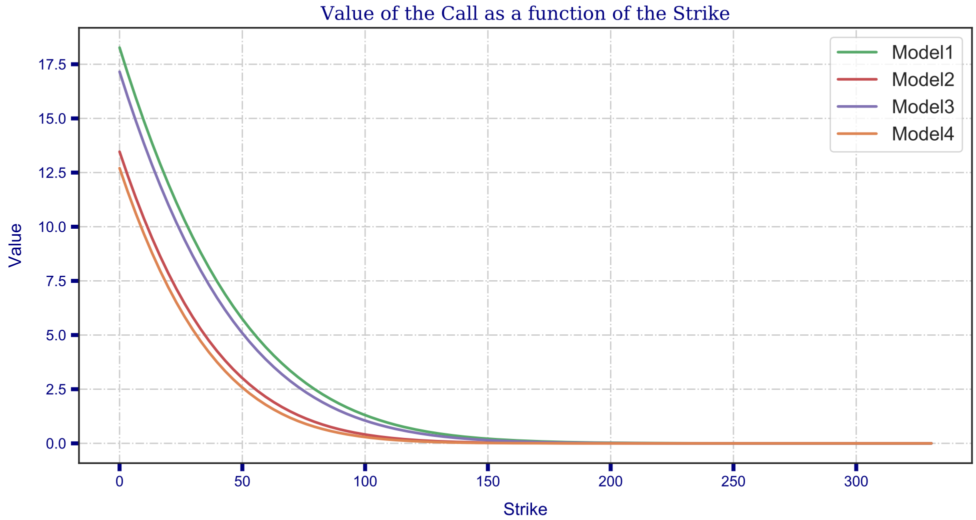


Figure 7.3: Values of the options on the CAT Index of February 2019

7.4.2 HDD/CDD indices

In this subsection, futures and options on HDD/CDD indices for the period from June to August 2019 will be inspected. This period was not chosen randomly, as the expected temperature over that period is around the base temperature of 18 °C usually considered for such indices. Figures 7.4 and 7.5 below show the distributions of the values of the CDD and HDD indices over that period.

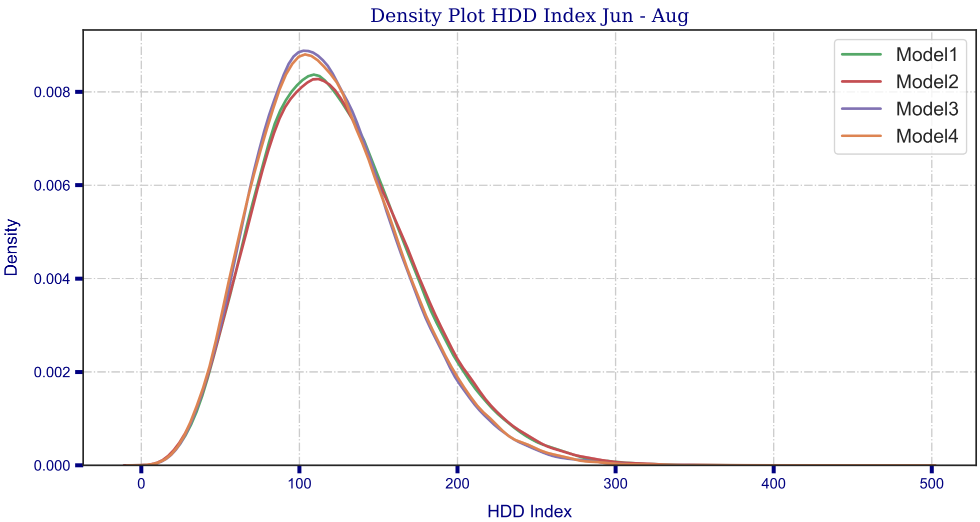


Figure 7.4: Density plot of the HDD Index value of June to August 2019

Over that time period, the time dependent speed of mean reversion is close, though slightly higher than its constant counterpart. This means that the we should not expect major differences in the results to come from that parameter. However, we clearly see that for both indices, the distributions of the models 1 and 2 are flatter with heavier tails, as well as slightly shifted to the right. It is fair to believe that these differences are due to Variance Gamma distribution used in those models.

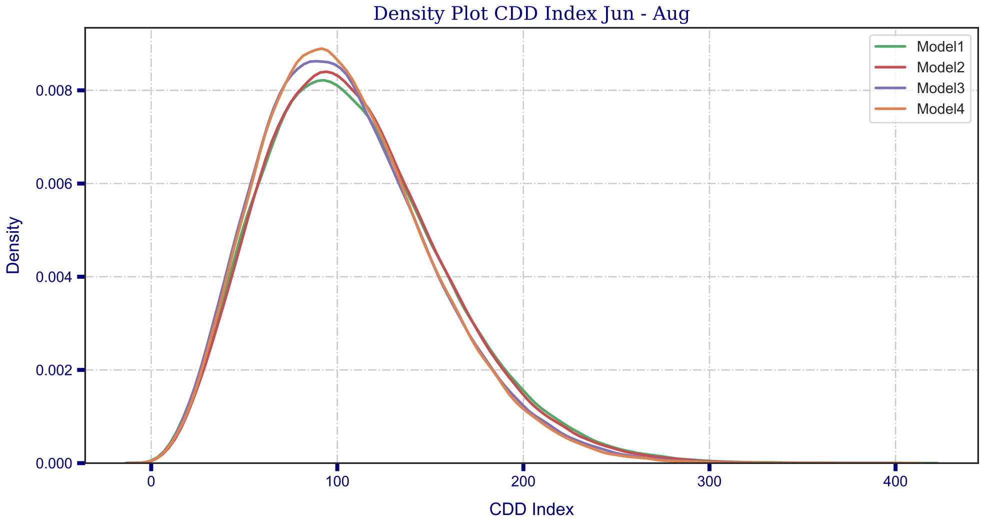


Figure 7.5: Density plot of the CDD Index value of June to August 2019

As the HDD and CDD indices takes respectively

- Daily HDD = max $(0, 18 -$ daily average temperature$)$
- Daily CDD = max $(0,$ daily average temperature $- 18)$

as daily value, the heavier tails of the models making use of the flatter Variance Gamma distribution is much more likely to cause larger jumps in either directions, leading to higher expected value for the HDD and CDD indices, as shown in the following table

	Model 1	Model 2	Model 3	Model 4
HDD Jun - Aug	124.24	124.51	119.68	119.75
CDD Jun - Aug	109.43	109.29	104.88	104.70

Table 7.2: Prices of the Future on HDD and CDD indices for Jun - Aug 2019

It is interesting to note that both the Variance Gamma distribution and the time-dependent speed of mean reversion are capable of significantly influencing the distribution of the indices on their own.

8 Conclusion

The introduction of time-dependent speed of mean reversion and Generalized Hyperbolic distributions seems justified in the context of stochastic modeling of temperature processes. It results in significant improvement in the quality of the fit and provides important improvements in terms of proper pricing and risk management of derivatives based on temperature indices.

Allowing the speed on mean reversion to depend on time does not add significant complexity to the models. A solution for the stochastic differential equation can still be derived straightforwardly and the calibration of the parameters is not considerably complexified. It appears, at least in the city of Stockholm considered in this work, that the speed of mean reversion fluctuates substantially during the year. The highest value for the speed of mean reversion is almost twice as high as the lowest during the year, and the implications of that fact is demonstrated in the section about derivative pricing and management.

Generalized Hyperbolic distributions are shown to be a far better match to the empirical distribution of the residuals compared to the normal distribution, especially at the tails. They allow for a superior modeling of extreme moves and therefore allow to quantify more realistically the likelihood of unexpected events. In the products based on indices from June to August 2019, where the time-dependent speed of mean reversion is close to that of the model where it is constant, it is shown that the distribution of payoff and the price of the Futures are significantly different for the models making use of the Variance Gamma distribution and those making use of the normal distribution.

Nevertheless, the framework provided by the normal variance-mean mixture allows for easy sampling from distributions of that family that are closed under convolution, by making a parallel to subordinated Brownian motion. It is then comfortable to confidently build fast algorithms to generate a sufficient number of path for Monte-Carlo methods to be convenient and reliable.

Appendices

A Models with constant mean reversion

This section provide the graphics and parameters for the models with a constant speed of mean reversion.

A.1 Under the Brownian assumption

We find the following parameters for $\sigma^2(t)$:

λ	ϕ_1	φ_1	ϕ_2	φ_2	ϕ_3	φ_3	ϕ_4	φ_4
5.486	2.323	1.191	−1.414	−1.300	0.6257	0.6458	−0.0247	−0.921

Table A.1: Parameters for $\sigma^2(t)$ assuming Brownian increments and constant κ

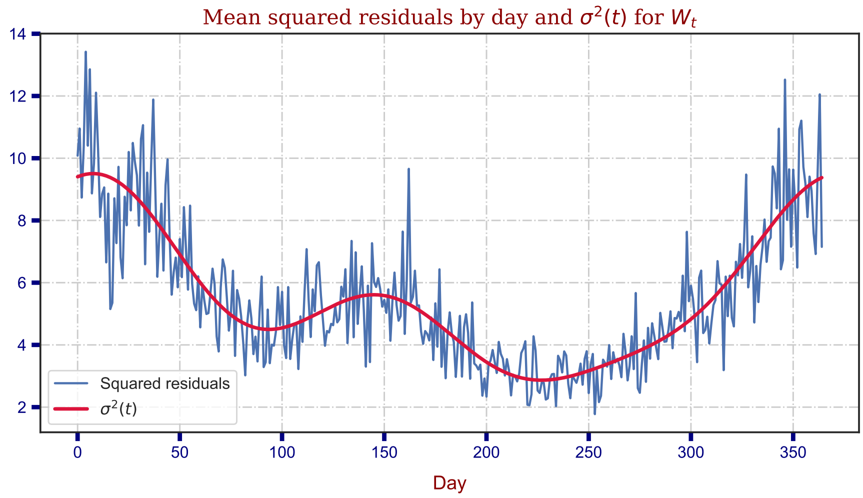


Figure A.1: Comparison of the daily squared residuals $\hat{\sigma}_\tau^2$ and fitted variance function $\sigma^2(t)$

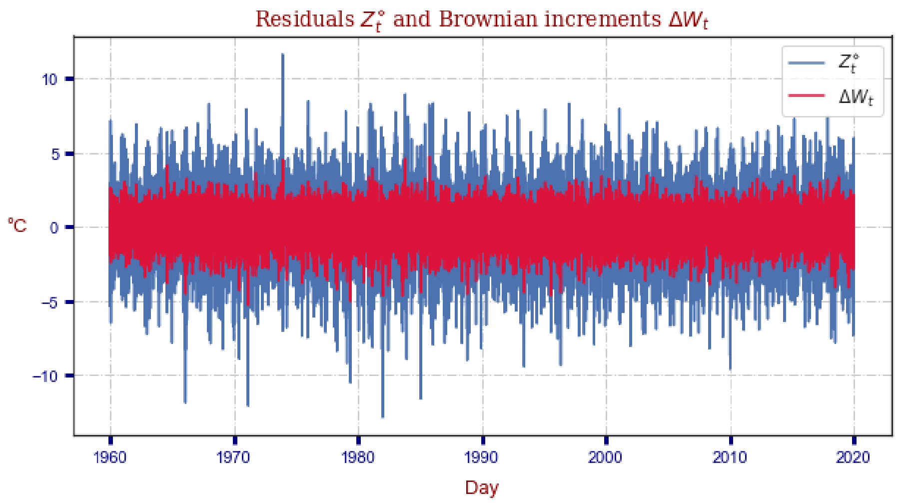


Figure A.2: Comparison of the residuals $Z_t^\diamond$ and Brownian Increments ΔW_t

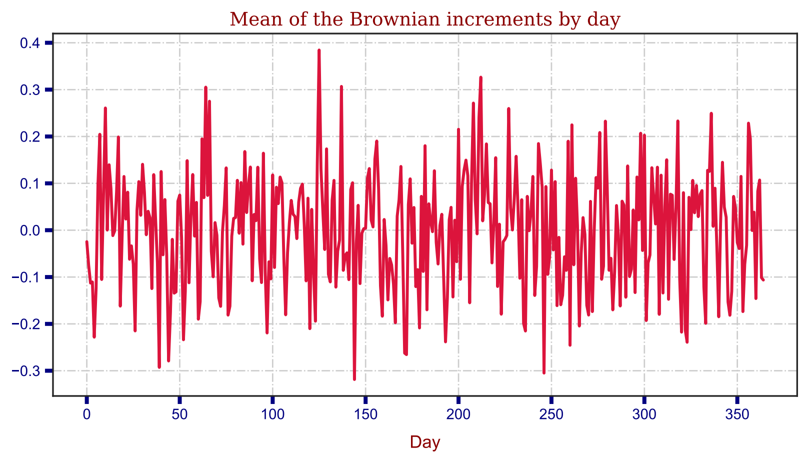


Figure A.3: Daily Mean of the estimated Brownian increments ΔW_t

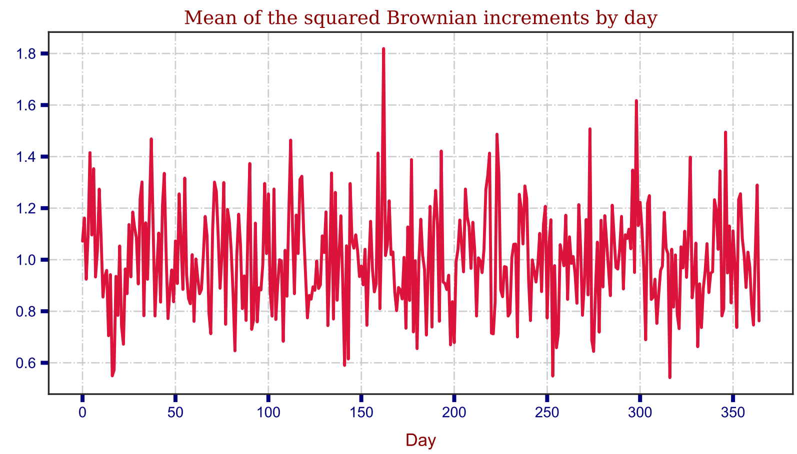


Figure A.4: Daily Mean of the squared estimated Brownian increments

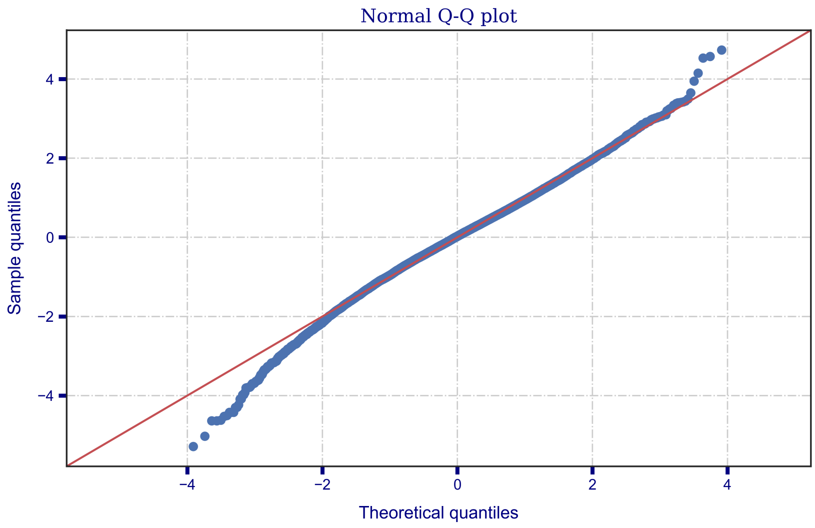


Figure A.5: Mean of the squared Lévy increments ΔL_t^2 by day

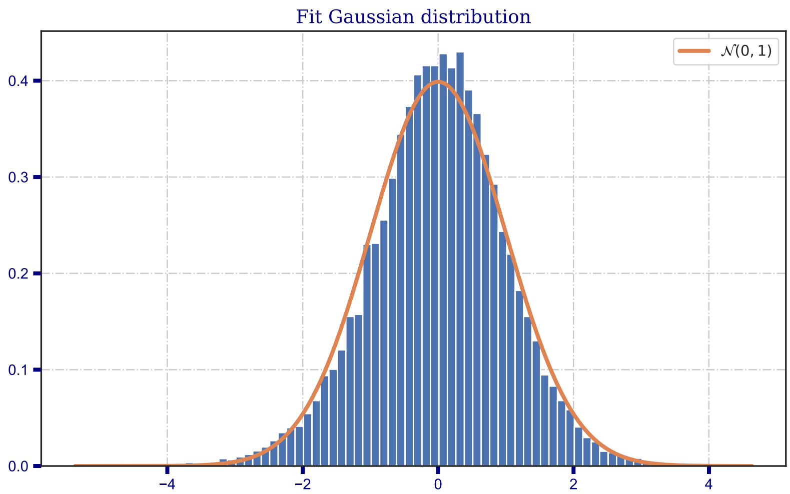


Figure A.6: Lévy increments ΔL_t

A.2 Under the general Lévy assumption

We find the following parameters for $\sigma^2(t)$:

λ	ϕ_1	φ_1	ϕ_2	φ_2	ϕ_3	φ_3	ϕ_4	φ_4
6.073	2.512	1.204	-1.567	-1.249	0.674	0.6593	0.0611	-0.897

Table A.2: Parameters for $\sigma^2(t)$ assuming Lévy increments and constant κ

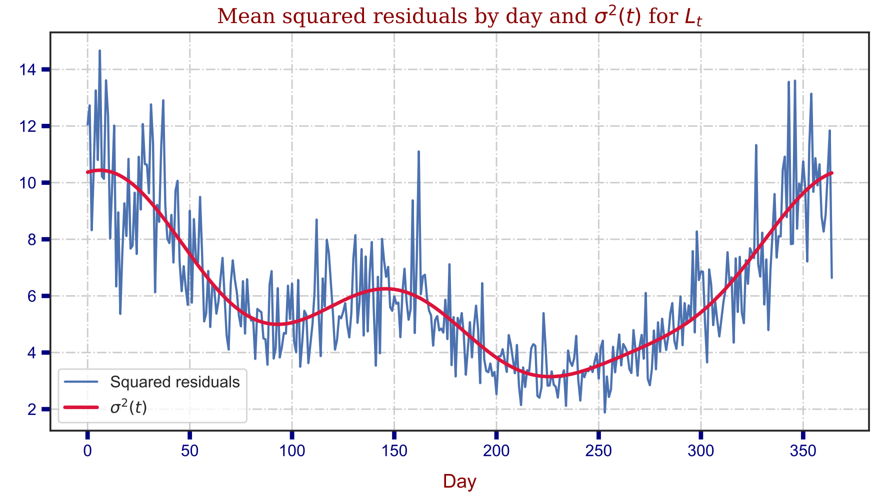


Figure A.7: Comparison of the daily squared residuals $\hat{\sigma}^2_\tau$ and fitted variance function $\sigma^2(t)$

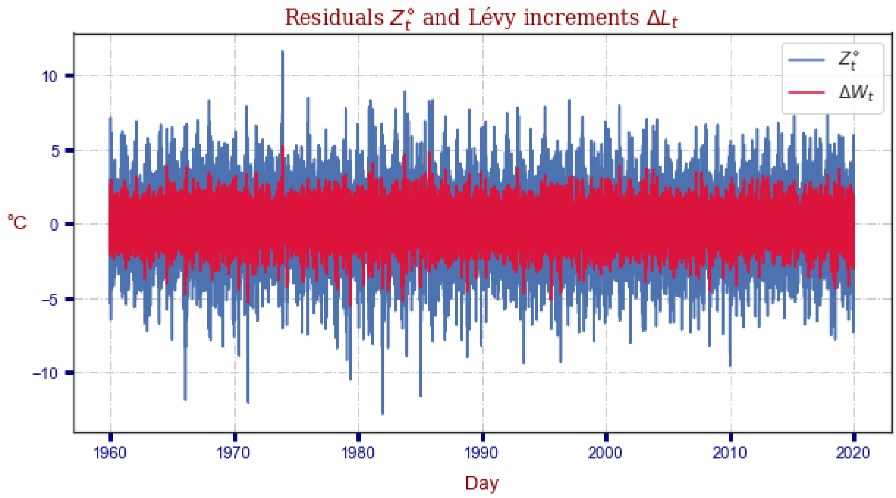


Figure A.8: Comparison of the residuals $Z_t^\diamond$ and Lévy Increments ΔL_t

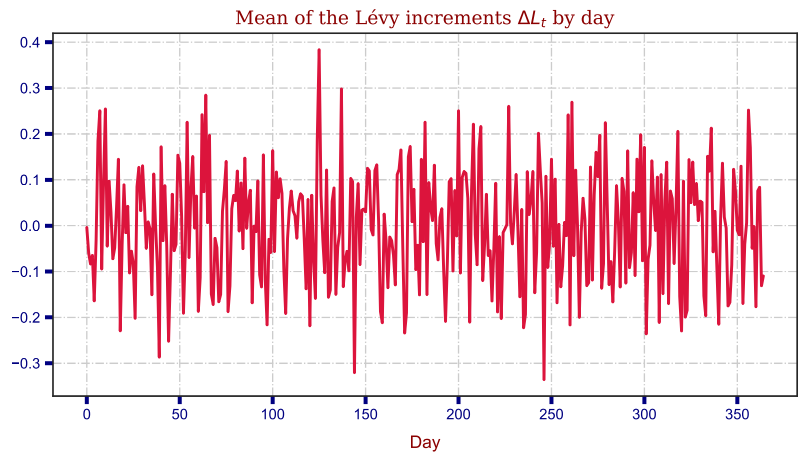


Figure A.9: Mean of the Lévy increments ΔL_t by day

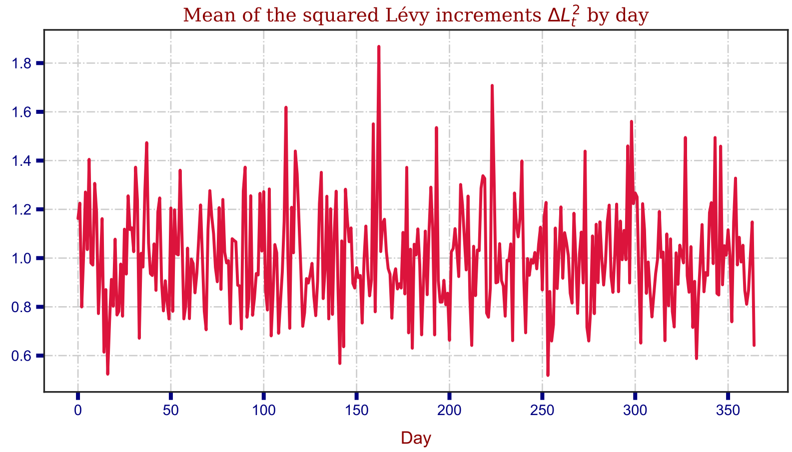


Figure A.10: Mean of the squared Lévy increments ΔL_t^2 by day

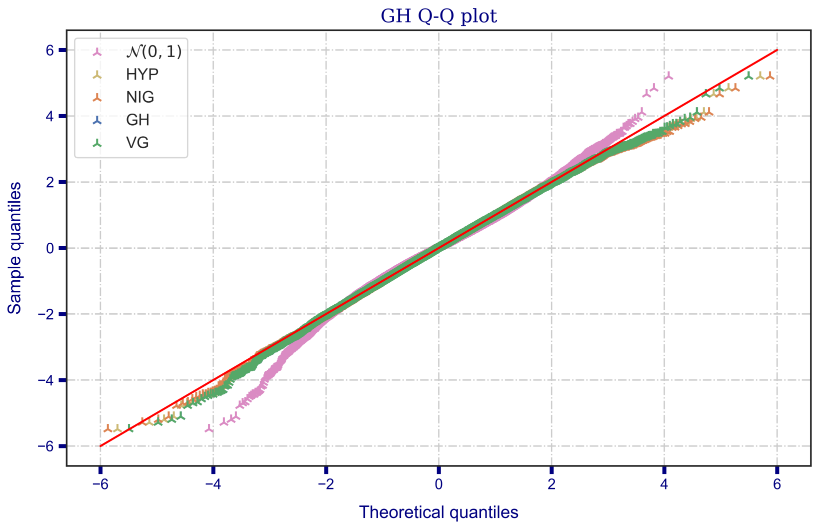


Figure A.11: Mean of the squared Lévy increments ΔL_t^2 by day

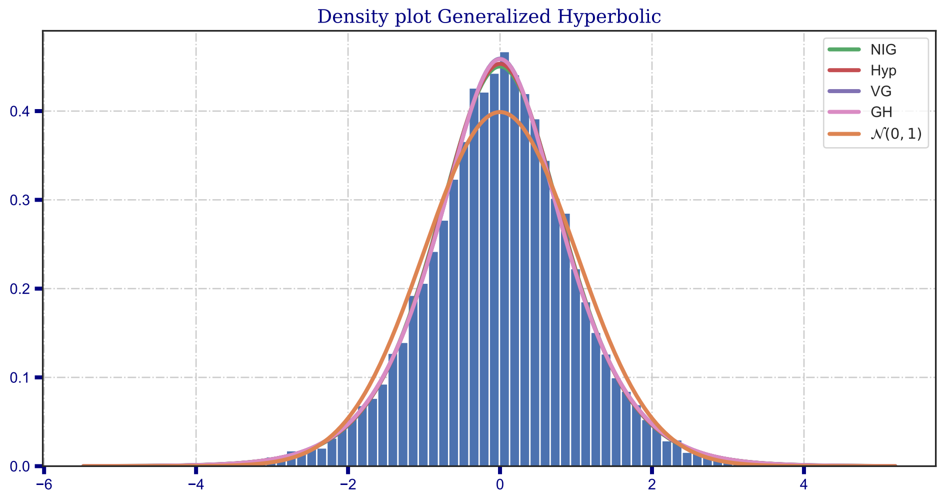


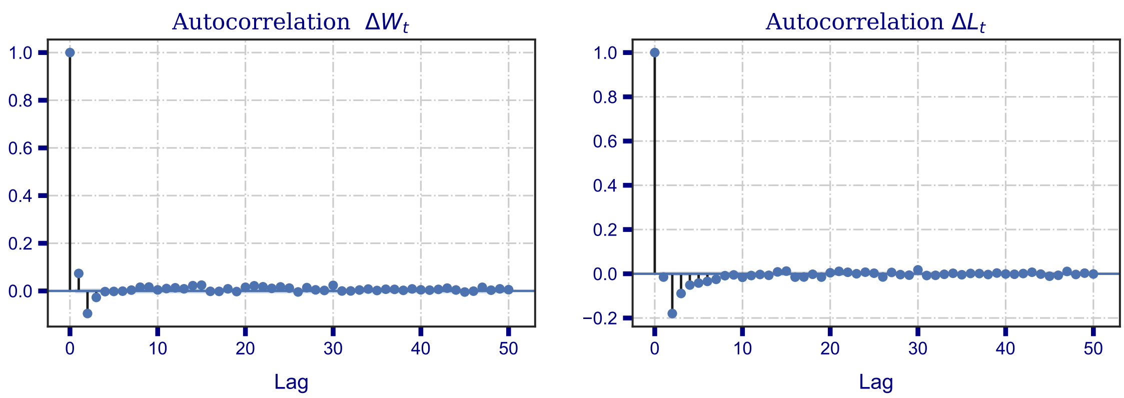
Figure A.12: Lévy increments ΔL_t

Below can be found the parameters for the fitted Generalized Hyperbolic distributions

	λ	α	β	δ
GH	2.953263	2.442156	0.000015	0.193215
NIG	-0.5	1.618144	0.000028	1.620796
VG	3.003475	2.451575	-0.000556	0
HYP	1	1.986294	0.000012	1.175125

Table A.3: Parameters for the fitted Generalized Hyperbolic distributions

A.3 Autocorrelation of the residuals



2

Figure A.13: Autocorrelation of the Brownian and Lévy increments.

B Codes

B.1 Python code for the calibration

```
from pathlib import Path
import math
import pingouin
import time
import timeit
import pandas as pd
import numpy as np
import scipy as scp
import scipy.optimize as spo
from scipy.optimize import curve_fit
from scipy.integrate import simps, trapz, romb, quad
from scipy import special
from scipy import optimize
import scipy.stats as scs
import statsmodels as sm
import statsmodels.api as smi
import numba as nb
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from math import log, sqrt, exp
from numba import jit, njit, prange, int32, float64, vectorize
import scipy.fftpack
from statsmodels.graphics.tsaplots import plot_acf
from scipy.special import k1, kv, gamma
from rpy2.robjects import r, pandas2ri
from rpy2 import robjects as ro
import seaborn as sns

StockholmData =
↪ pd.read_table('C:/Users/nicol/Dropbox/Mémoire/TG_STAID000010.txt', sep
↪ = ',')
font = {'family': 'serif',
'color':  'darkred',
'weight': 'ultralight',
'size': 14,
}
font2 = {'family': 'serif',
'color':  'navy',
'weight': 'ultralight',
'size': 14,
}

plt.rcParams['xtick.labelsize'] = 11
plt.rcParams['ytick.labelsize'] = 11
plt.rcParams['xtick.color'] = 'navy'
plt.rcParams['ytick.color'] = 'navy'
plt.rcParams['ytick.major.width'] = 3
plt.rcParams['xtick.major.width'] = 3
plt.rcParams['grid.linestyle'] = '-.'
plt.rcParams['axes.grid'] = True

################################################################
```

```python
#################################################################
def GH_pdf(x, Lambda = 2.954556, Alpha = 2.442359, Beta = 0.0001190245, Delta = 0.1908328):
    return np.sqrt(Alpha**2 - Beta**2)**Lambda * kv(Lambda-0.5, Alpha*np.sqrt(Delta**2 + x**2)) * np.exp(Beta*x) * np.sqrt(Delta**2 + x**2)**(Lambda-0.5) /(Delta**Lambda * Alpha**(Lambda-0.5) * np.sqrt(2*np.pi) * kv(Lambda, Delta*np.sqrt(Alpha**2 - Beta**2)))

def NIG_pdf(x, Alpha = 1.617545, Beta = 2.83658e-05, Delta = 1.620211):
    return Alpha*Delta*k1(Alpha*np.sqrt(Delta**2 + x**2))*np.exp(Delta*np.sqrt(Alpha**2 - Beta**2)+Beta*x)/(np.pi * np.sqrt(Delta**2 + x**2))

def HYP_pdf(x, Alpha = 1.984489, Beta = 9.43859e-05, Delta = 1.172987):
    return np.sqrt(Alpha**2 - Beta**2)*np.exp(-Alpha*np.sqrt(Delta**2 + x**2)+ Beta*x)/(2*Delta*Alpha*k1(Delta*np.sqrt(Alpha**2 - Beta**2)))

def VG_pdf(x, Lambda = 3.020783, Alpha =  2.458418, Beta = 1.681182e-05):
    return ((Alpha**2 - Beta**2)**Lambda)*(np.abs(x)**(Lambda-0.5))*np.exp(Beta*x)*kv(Lambda-0.5, Alpha*np.abs(x))/(np.sqrt(np.pi)*gamma(Lambda)*((2*Alpha)**(Lambda-0.5)))

#################################################################
def FitLevy_rWrap(pyL, path):
    #Step 1 Fit Lévy
    r('library(ghyp)')
    r('library(stats)')
    L_t = pandas2ri.py2ri(pyL)
    ro.globalenv['L_t'] = L_t
    r('L_t = unname(unlist(L_t))')
    r('GHyp.fit <- fit.ghypuv(L_t, opt.pars = c(mu = FALSE), mu = 0)')
    r('coef(GHyp.fit, type = "alpha.delta")')
    r('GHyp.fit <- fit.ghypuv(L_t, opt.pars = c(mu = FALSE), mu = 0)')
    r('GHyp.coef <- coef(GHyp.fit, type = "alpha.delta")')
    r('nig.fit <- fit.NIGuv(L_t, opt.pars = c(mu = FALSE), mu = 0)')
    r('nig.coef <- coef(nig.fit, type = "alpha.delta")')
    r('Hyp.fit <- fit.hypuv(L_t, opt.pars = c(mu = FALSE), mu = 0)')
    r('Hyp.coef <- coef(Hyp.fit, type = "alpha.delta")')
    r('VGuv.fit <- fit.VGuv(L_t, opt.pars = c(mu = FALSE), mu = 0)')
    r('VGuv.coef <- coef(VGuv.fit, type = "alpha.delta")')
    r('Results <- rbind(data.frame(GHyp.coef), data.frame(nig.coef), data.frame(Hyp.coef), data.frame(VGuv.coef))')

    ParamGH = ro.globalenv['Results']
    PyLevyParam = pandas2ri.ri2py(ParamGH)
    PyLevyParam.set_index(np.array(["GH","NIG","HYP", "VG"]), drop=True, inplace=True)
    #Step 2 check fit
    print("GH parameters: \n \n")
    print(PyLevyParam)
    GH_lambda, GH_alpha, GH_delta, GH_beta, GH_mu = PyLevyParam.loc['GH'].values
    NIG_lambda, NIG_alpha, NIG_delta, NIG_beta, NIG_mu = PyLevyParam.loc['NIG'].values
```

```
    HYP_lambda, HYP_alpha, HYP_delta, HYP_beta, HYP_mu = PyLevyParam.loc['HYP'].values
    VG_lambda, VG_alpha, VG_delta, VG_beta, VG_mu = PyLevyParam.loc['VG'].values
    r('p  = ((1:length(L_t))-0.5)/length(L_t)')
    r('qGH = qghyp(p, object = GHyp.fit)')
    r('qNIG = qghyp(p, object = nig.fit)')
    r('qHyp = qghyp(p, object = Hyp.fit)')
    r('qVG = qghyp(p, object = VGuv.fit)')
    r('qN = qnorm(p,mean=0,sd=1)')
    r('TableQuantile <- data.frame(cbind(data.frame(sort(L_t)), data.frame(qN), data.frame(qGH), data.frame(qNIG), data.frame(qHyp), data.frame(qVG)))')
    r('colnames(TableQuantile) = c("Sample", "qN", "qGH", "qNIG","qHyp", "qVG")')
    quantile = pandas2ri.ri2py(ro.globalenv['TableQuantile'])

    plt.figure(figsize=(10,6))
    plt.scatter(quantile.qN, quantile.Sample, label = r"$\mathcal{N}\left(0, 1 \right)$", marker = '2', s = 60, color = 'C6')
    plt.scatter(quantile.qHyp, quantile.Sample, label = 'HYP', marker = '2', s = 60, color = 'C8')
    plt.scatter(quantile.qNIG, quantile.Sample, label = 'NIG', marker = '2', s = 60, color = 'C1')
    plt.scatter(quantile.qGH, quantile.Sample, label = 'GH', marker = '2', s = 60, color = 'C0')
    plt.scatter(quantile.qVG, quantile.Sample, label = 'VG', marker = '2', s = 60, color = 'C2')
    plt.plot(np.linspace(-6,6, 10), np.linspace(-6,6, 10), color = 'red')
    plt.legend(fontsize = 'large')
    plt.title("GH Q-Q plot", fontdict = font, color='navy')
    plt.savefig(path + 'qqplot2.png', bbox_inches='tight', dpi=600)
    plt.show()
    plt.close()

    GH_p = lambda z: GH_pdf(x = z, Lambda = GH_lambda, Alpha = GH_alpha, Beta = GH_beta, Delta = GH_delta)
    NIG_p = lambda z: NIG_pdf(x = z, Alpha = NIG_alpha, Beta = NIG_beta, Delta = NIG_delta)
    HYP_p = lambda z: HYP_pdf(x = z, Alpha = HYP_alpha, Beta = HYP_beta, Delta = HYP_delta)
    VG_p = lambda z: VG_pdf(x = z, Lambda = VG_lambda, Alpha =  VG_alpha, Beta = VG_beta)

    Dists = zip([NIG_p, HYP_p, VG_p, GH_p, scs.norm.pdf], ['NIG', 'Hyp', 'VG', 'GH', r"$\mathcal{N}\left(0, 1 \right)$"], ['C2', 'C3', 'C4', 'C6', 'C1'])
    g = plt.figure(figsize=(12,6))
    plt.hist(pyL, bins=80, density=True)
    lnspc = np.linspace(pyL.min(), pyL.max(), 600)
    for dist, label, col in Dists:
        plt.plot(lnspc, dist(lnspc), label = label, linewidth = 3, color = col)
    plt.title('Density plot ' + 'Generalized Hyperbolic', fontdict = font, color='navy')
```

```python
    plt.legend()
    plt.savefig(path + 'LévyDensityPlot.pdf', bbox_inches='tight')
    plt.show()
    plt.close()
    return None

################################################################
def Gaussian_fit(W, path):
    print('Fit gaussian : \n \n')
    g = plt.figure(figsize=(10,6))
    plt.hist(W, bins=80, density=True)
    lnspc = np.linspace(W.min(), W.max(), 600)
    dist = getattr(scs, 'norm')
    plt.plot(lnspc, dist.pdf(lnspc, 0, 1), linewidth = 3, label = r"$\mathcal{N}\left(0, 1 \right)$")
    plt.title('Fit Gaussian distribution', fontdict = font, color='navy')
    plt.legend()
    plt.savefig(path + 'BMDensityPlot.pdf', bbox_inches='tight')
    plt.show()
    #QQplot
    fig, ax = plt.subplots(figsize=(10,6))
    smi.qqplot(W, dist=scs.norm, loc=0, scale=1, fit=False, line='45', ax = ax) # = ..
    plt.title("Normal Q-Q plot", fontdict = font, color='navy')
    plt.savefig(path + 'BMqqplot.png', bbox_inches='tight', dpi=600)
    plt.show()
    plt.close()
    return True
################################################################
################################################################

class TempSerie():
    def __init__(self, Data, Location):
        self.Location = Location
        Data.drop(columns = ['STAID', ' SOUID'], inplace = True)
        Data.columns = ['Date','Temp', 'Quality']
        Data['Date'] = pd.to_datetime(Data['Date'], format = '%Y%m%d')
        Data['Temp'] = Data['Temp']/10
        Data.index = Data['Date']
        Data = Data[~((Data['Date'].dt.month == 2) & (Data['Date'].dt.day == 29))]
        Data = Data[Data['Date'].dt.year < 2020]
        Data['N'] = np.arange(Data.shape[0]) + 1
        self.Data = Data
    def Cut(self, YearFrom, YearTo):
        Data2 = self.Data
        Data2 = Data2[(Data2['Date'].dt.year >= YearFrom) & (Data2['Date'].dt.year <= YearTo)].copy()
        Data2['N'] = np.arange(Data2.shape[0]) + 1
        Data2.index = Data2['Date']
        return Data2
    def Fit(self, fit, Zoom, ConstantKappa):
        FromYear, ToYear = fit
        FromYearZoom, ToYearZoom = Zoom
        Slice = self.Cut(FromYear, ToYear).copy()
```

```python
        Folder = 'C:/Users/nicol/Dropbox/Mémoire/Tempfig/' + 'FitConstK' + str(ConstantKappa) + str(FromYear) + str(ToYear) +'/'
        Path(Folder).mkdir(parents=True, exist_ok=True)
        self.Folder = Folder
        omega = 2/365*np.pi
        FunctionS_t = lambda t, a, b, c, phi, d, phi2 : a + t*b + c*np.sin(omega*t + phi) + d*np.sin(2*omega*t + phi2)
        ParamS_t, _ = curve_fit(FunctionS_t, Slice.N, Slice.Temp, [5, 0, 10, 0, 1, 30] )
        ParamS_tStr = ['a', 'b', 'c', 'phi', 'd', 'phi2']
        print('\n\nParameters for S(t)\n')
        for name, param in zip(ParamS_tStr, ParamS_t):
            print(name + ' = ' + str(param))

        fig, ax = plt.subplots(figsize=(10,5))
        plt.title("Daily temperature", fontdict = font)
        ax.set_ylabel('°C', rotation='horizontal', color='darkred', size = 13, labelpad=10)
        Slice['S_t'] = FunctionS_t(Slice.N, ParamS_t[0], ParamS_t[1], ParamS_t[2], ParamS_t[3], ParamS_t[4], ParamS_t[5])
        Slice['Detrended'] = Slice.Temp - Slice.S_t
        Slice['Detrended+1'] = Slice['Detrended'].shift(-1)
        Slice['Detrended-1'] = Slice['Detrended'].shift(1)
        print("Mean of Detrended and Deseasonalized temperatures TildeT_t: " + str(Slice.Detrended.mean()))
        print("Std of Detrended and Deseasonalized temperatures TildeT_t: " + str(Slice.Detrended.std()))
        ZoomData = Slice[(Slice['Date'].dt.year >= FromYearZoom) & (Slice['Date'].dt.year <= ToYearZoom)].copy()
        plt.plot(ZoomData.Date, ZoomData.Temp)
        plt.plot(ZoomData.Date, ZoomData.S_t, linewidth = 2, color='red')
        plt.show()
        fig.savefig(Folder + 'lin2sinzoom.pdf', bbox_inches='tight')
        fig, ax = plt.subplots(figsize=(10,5))
        plt.title("Detrended and deseasonalized temperatures", fontdict = font)
        ax.set_ylabel('°C', rotation='horizontal', color='darkred', size = 13, labelpad=10)
        plt.plot(ZoomData.Date, ZoomData.Detrended, color='crimson')
        plt.show()
        fig.savefig(Folder + 'lin2sinzoomRES.pdf', bbox_inches='tight')

        #monthly mean
        MonthlyDetrended = Slice.groupby(by=[Slice.index.month]).Detrended.mean()
        MonthlyDetrended.index = ['January', 'February', 'March', 'April', 'May',                             'June', 'July', 'August', 'September', 'October', 'November', 'December']
        fig, ax = plt.subplots(figsize=(10,5))
        plt.title(r"Monthly mean of $\tilde{T}_{t}$", fontdict = font)
        ax.set_xlabel('Month', rotation='horizontal', color='darkred', size = 13, labelpad=10)
        plt.xticks([ 2*x for x in range(0, 6)], ['January', 'March', 'May', 'July', 'September', 'November'])
        plt.plot(MonthlyDetrended.index, MonthlyDetrended, color='crimson')
        plt.show()
```

```python
    fig.savefig(Folder + 'lin2sinzoomMonthlyRes.pdf', bbox_inches='tight')

    YearlyDetrended = Slice.groupby(by=[Slice.index.year]).Detrended.mean()
    fig, ax = plt.subplots(figsize=(10,5))
    plt.title(r"Yearly mean of $\tilde{T}_{t}$", fontdict = font)
    ax.set_xlabel('Year', rotation='horizontal', color='darkred', size = 13, labelpad=10)
    plt.plot(YearlyDetrended.index, YearlyDetrended, color='crimson')
    plt.show()
    fig.savefig(Folder + 'lin2sinzoomYearlyRes.pdf', bbox_inches='tight')

    Slice['T_t*T_t+1'] = Slice['Detrended']*Slice['Detrended+1']
    Slice['T_t**2'] = Slice['Detrended+1']**2
    ConstRho = Slice['T_t*T_t+1'][:-1].sum()/Slice['T_t**2'][:-1].sum()
    ConstKappa = -np.log(ConstRho)
    print('Rho = ' + str(ConstRho))
    print('Kappa = ' + str(ConstKappa))

    if ConstantKappa == False:
        Rho_t = np.zeros(365)
        tempdf = Slice[:-1]
        for i in np.arange(365):
            tempdf2 = tempdf[tempdf.N.mod(365) == i]
            Rho_t[i] = tempdf2['T_t*T_t+1'].sum()/tempdf2['T_t**2'].sum()
        kappa_t = - np.log(Rho_t)
        FunctionIntKappa = lambda t, Lambda, phi1, varphi1, phi2, varphi2, phi3, varphi3, phi4, varphi4 : Lambda + phi1*(np.cos(omega*t+varphi1) - np.cos(omega*(t+1)+varphi1))/(omega) + phi2*(np.cos(2*omega*t+varphi2) - np.cos(2*omega*(t+1)+varphi2))/(2*omega) + phi3*(np.cos(3*omega*t+varphi3) - np.cos(3*omega*(t+1)+varphi3))/(3*omega) + phi4*(np.cos(3*omega*t+varphi4) - np.cos(4*omega*(t+1)+varphi4))/(4*omega)

        Paramkappa_t, _ = curve_fit(FunctionIntKappa, np.arange(365), kappa_t, [0.2, 0, 0, 0, 0, 0, 0, 0, 0])
        Paramkappa_tStr = ['Lambda', 'phi1', 'varphi1', 'phi2', 'varphi2', 'phi3', 'varphi3', 'phi4', 'varphi4']
        print('\n\nParameters for kappa(t)\n')
        for name, param in zip(Paramkappa_tStr, Paramkappa_t):
            print(name + ' = ' + str(param))
        fig, ax = plt.subplots(figsize=(10,5))
        plt.title("Mean reversion parameters", fontdict = font)
        ax.set_xlabel('Day', rotation='horizontal', color='darkred', size = 13, labelpad=10)
        plt.plot(np.arange(365), kappa_t, label=r'$\hat{\kappa_t}$', linewidth = 1.8)
```

```python
        IntKappaCycle = FunctionIntKappa(np.arange(365), Paramkappa_t[0], Paramkappa_t[1], Paramkappa_t[2], Paramkappa_t[3], Paramkappa_t[4], Paramkappa_t[5], Paramkappa_t[6], Paramkappa_t[7], Paramkappa_t[8])
        plt.plot(np.arange(365), IntKappaCycle, label=r'$ \int_{t}^{t +1 } \kappa(\xi) d \xi$', color='crimson', linewidth = 2.6)
        plt.plot(np.arange(365), np.full(np.arange(365).shape[0], ConstKappa), label=r'$\hat{\kappa}$', color='darkcyan', linewidth = 2.6)
        plt.legend(fontsize = 'medium')
        plt.show()
        fig.savefig(Folder + 'kappa_t.pdf', bbox_inches='tight')

        Slice['Intkappa_t'] = FunctionIntKappa(Slice.N, Paramkappa_t[0], Paramkappa_t[1], Paramkappa_t[2], Paramkappa_t[3], Paramkappa_t[4], Paramkappa_t[5], Paramkappa_t[6], Paramkappa_t[7], Paramkappa_t[8])
        FunctionKappa = lambda t : Paramkappa_t[0] + Paramkappa_t[1]*np.sin(omega*t+Paramkappa_t[2]) + Paramkappa_t[3]*np.sin(2*omega*t+Paramkappa_t[4]) + Paramkappa_t[5]*np.sin(3*omega*t+Paramkappa_t[6]) + Paramkappa_t[7]*np.sin(3*omega*t+Paramkappa_t[8])

        Slice['kappa_t'] = FunctionKappa(Slice.N)
        Slice['Z_t'] = Slice['Detrended+1'] - Slice['Detrended'] * np.exp(-Slice['Intkappa_t'])
        Slice['SquaredZ_t'] = Slice['Z_t']**2

        fig, ax = plt.subplots(figsize=(10,5))
        plt.title(r"Residuals : $Z^{\diamond}_{t} = \tilde{T}_{t+1} - \tilde{T}_{t} \mathrm{e}^{- \int_{t}^{t +1 } \kappa(\xi) d \xi}$", fontdict = font)
        ax.set_ylabel('°C', rotation='horizontal', color='darkred', size = 13, labelpad=10)
        plt.plot(Slice.Date, Slice['Z_t'], color='crimson')
        plt.show()
        fig.savefig(Folder + 'ResidualsZ_t.pdf', bbox_inches='tight')
    elif ConstantKappa ==True:
        Slice['kappa_t'] = ConstKappa
        Slice['Z_t'] = Slice['Detrended+1'] - Slice['Detrended'] * np.exp(-Slice['kappa_t'])
        Slice['SquaredZ_t'] = Slice['Z_t']**2
        fig, ax = plt.subplots(figsize=(10,5))
        plt.title(r"Residuals : $Z^{\diamond}_{t} = \tilde{T}_{t+1} - \tilde{T}_{t} \mathrm{e}^{- \kappa}$", fontdict = font)
        ax.set_ylabel('°C', rotation='horizontal', color='darkred', size = 13, labelpad=10)
        plt.plot(Slice.Date, Slice['Z_t'], color='crimson')
        plt.show()
        fig.savefig(Folder + 'ResidualsZ_t.pdf', bbox_inches='tight')

    print(r"Mean of residuals $Z_t$ : " + str(Slice['Z_t'].mean()))
    print(r"Std of residuals $Z_t : " + str(Slice['Z_t'].std()))

    #Analysis assuming Brownian Motion as Lévy process

```

```python
    Slice['SigmaDeltaW_t'] = Slice['Z_t']/np.sqrt((1-np.exp(-2*Slice['kappa_t']))/(2*Slice['kappa_t']))
    Slice['SigmaDeltaW_tSq'] = Slice['SigmaDeltaW_t']**2

    MeanDayW_t = np.zeros(365)
    SigmaSquaredDayW_t = np.zeros(365)
    for i in np.arange(365):
        temp = Slice[Slice.N.mod(365) == i]
        MeanDayW_t[i] = temp['SigmaDeltaW_t'].mean()
        SigmaSquaredDayW_t[i] = temp['SigmaDeltaW_tSq'].mean()

    FSigma = lambda t, Lambda, phi1, varphi1, phi2, varphi2, phi3, varphi3, phi4, varphi4 : Lambda + phi1*np.sin(omega*t+varphi1) + phi2*np.sin(2*omega*t+varphi2)          + phi3*np.sin(3*omega*t+varphi3) + phi4*np.sin(4*omega*t+varphi4)
    ParamSigmaW, _ = curve_fit(FSigma, np.arange(365), SigmaSquaredDayW_t, [0.2, 0, 0, 0, 0, 0, 0, 0, 0])
    print('\n\nParameters sigma(t) for W_t \n')
    ParamSigmaWStr = ['Lambda', 'phi1', 'varphi1', 'phi2', 'varphi2', 'phi3', 'varphi3', 'phi4', 'varphi4']
    for name, param in zip(ParamSigmaWStr, ParamSigmaW):
        print(name + ' = ' + str(param))

    SigmasW_t = FSigma(np.arange(365), ParamSigmaW[0], ParamSigmaW[1], ParamSigmaW[2], ParamSigmaW[3], ParamSigmaW[4], ParamSigmaW[5], ParamSigmaW[6], ParamSigmaW[7], ParamSigmaW[8])

    fig, ax = plt.subplots(figsize=(10,5))
    plt.title(r"Mean squared residuals by day and $\sigma^2(t)$ for $W_t$", fontdict = font, color='darkred')
    ax.set_xlabel('Day', rotation='horizontal', size = 13, labelpad=10, color='darkred')
    plt.plot(np.arange(365), SigmaSquaredDayW_t, label=r'Squared residuals')
    plt.plot(np.arange(365), SigmasW_t, color='crimson', linewidth = 2.4, label=r'$\sigma^2(t)$')
    plt.legend()
    plt.show()

    fig.savefig(Folder + 'SigmaSquaredresidualsW_t.pdf', bbox_inches='tight')

    Slice['Sigma_tW_t'] = FSigma(Slice.N, ParamSigmaW[0], ParamSigmaW[1], ParamSigmaW[2], ParamSigmaW[3], ParamSigmaW[4], ParamSigmaW[5], ParamSigmaW[6], ParamSigmaW[7], ParamSigmaW[8])
    Slice['Sigma_tW_t'] = np.sqrt(Slice['Sigma_tW_t'])
    Slice['DeltaW_t'] = Slice['SigmaDeltaW_t']/Slice['Sigma_tW_t']
    Slice['DeltaW_tSq'] = Slice['DeltaW_t']**2

    fig, ax = plt.subplots(figsize=(10,5))
    plt.title(r"Brownian increments $\Delta W_t$", fontdict = font)
    ax.set_ylabel('°C', rotation='horizontal', color='darkred', size = 13, labelpad=10)
    plt.plot(Slice.Date, Slice['DeltaW_t'], color='crimson', label = r"$\Delta W_t$", linewidth = 2.2)
    plt.show()
```

```python
    fig.savefig(Folder + 'BrownianIncrements.pdf', bbox_inches='tight')
    print(r"Mean of $\Delta L_t$ : " + str(Slice['DeltaW_t'].mean()))
    print(r"Std of $\Delta L_t$ : " + str(Slice['DeltaW_t'].std()))

    fig, ax = plt.subplots(figsize=(10,5))
    plt.title(r"Residuals $Z^{\diamond}_{t}$ and Brownian increments $\Delta W_t$", fontdict = font)
    ax.set_ylabel('°C', rotation='horizontal', color='darkred', size = 13, labelpad=10)
    ax.set_xlabel('Day', rotation='horizontal', size = 13, labelpad=10, color='darkred')
    plt.plot(Slice.Date, Slice['Z_t'], label =  r"$Z^{\diamond}_{t}$")
    plt.plot(Slice.Date, Slice['DeltaW_t'], color='crimson', label = r"$\Delta W_t$")
    plt.legend(fontsize = 'large', loc = 'upper right')
    plt.show()
    fig.savefig(Folder + 'ResidualsBrownianincrements.png', bbox_inches='tight')

    NewMeanDayW_t = np.zeros(365)
    NewSquaredDayW_t = np.zeros(365)
    for i in np.arange(365):
        temp = Slice[Slice.N.mod(365) == i]
        NewMeanDayW_t[i] = temp['DeltaW_t'].mean()
        NewSquaredDayW_t[i] = temp['DeltaW_tSq'].mean()

    fig, ax = plt.subplots(figsize=(10,5))
    plt.title(r"Mean of the squared Brownian increments by day", fontdict = font)
    ax.set_xlabel('Day', rotation='horizontal', size = 13, labelpad=10, color='darkred')
    plt.plot(np.arange(365), NewSquaredDayW_t, color = 'crimson', linewidth = 2.2)
    plt.show()
    fig.savefig(Folder + 'MeanSquaredDeltaW_t.pdf', bbox_inches='tight')

    fig, ax = plt.subplots(figsize=(10,5))
    plt.title(r"Mean of the Brownian increments by day", fontdict = font)
    ax.set_xlabel('Day', rotation='horizontal', size = 13, labelpad=10, color='darkred')
    plt.plot(np.arange(365), NewMeanDayW_t, color = 'crimson', linewidth = 2.2)
    plt.show()
    fig.savefig(Folder + 'MeanDeltaW_t.pdf', bbox_inches='tight')

    #Analysis assuming a general Lévy process
    Slice['SigmaDeltaL_t'] = Slice['Detrended+1']-Slice['Detrended'] + Slice['kappa_t']*(Slice['Detrended+1']-Slice['Detrended'])/2
    Slice['SigmaDeltaL_tSq'] = Slice['SigmaDeltaL_t']**2

    MeanDaySigmaL_t = np.zeros(365)
    SigmaSquaredDayL_t = np.zeros(365)
    for i in np.arange(365):
        temp = Slice[Slice.N.mod(365) == i]
```

```
        MeanDaySigmaL_t[i] = temp['SigmaDeltaL_t'].mean()
        SigmaSquaredDayL_t[i] = temp['SigmaDeltaL_tSq'].mean()

    fig, ax = plt.subplots(figsize=(10,5))
    plt.title(r"Daily Mean of $\sigma(t) \Delta L_t$", fontdict = font, color='darkred')
    ax.set_xlabel('Day', rotation='horizontal', size = 13, labelpad=10, color='darkred')
    plt.plot(np.arange(365), MeanDaySigmaL_t, color='crimson', linewidth = 2.4, label=r'$\sigma^2(t)$')
    plt.show()
    fig.savefig(Folder + 'MeanDaySigmaL_t.pdf', bbox_inches='tight')

    ParamSigmaL, _ = curve_fit(FSigma, np.arange(365), SigmaSquaredDayL_t, [0.2, 0, 0, 0, 0, 0, 0, 0, 0])
    print('\n\nParameters sigma(t) for L_t \n')
    ParamSigmaLStr = ['Lambda', 'phi1', 'varphi1', 'phi2', 'varphi2', 'phi3', 'varphi3', 'phi4', 'varphi4']
    for name, param in zip(ParamSigmaLStr, ParamSigmaL):
        print(name + ' = ' + str(param))

    SigmasL_t = FSigma(np.arange(365), ParamSigmaL[0], ParamSigmaL[1], ParamSigmaL[2], ParamSigmaL[3], ParamSigmaL[4], ParamSigmaL[5], ParamSigmaL[6], ParamSigmaL[7], ParamSigmaL[8])
    fig, ax = plt.subplots(figsize=(10,5))
    plt.title(r"Mean squared residuals by day and $\sigma^2(t)$ for $L_t$", fontdict = font, color='darkred')
    ax.set_xlabel('Day', rotation='horizontal', size = 13, labelpad=10, color='darkred')
    plt.plot(np.arange(365), SigmaSquaredDayL_t, label=r'Squared residuals')
    plt.plot(np.arange(365), SigmasL_t, color='crimson', linewidth = 2.4, label=r'$\sigma^2(t)$')
    plt.legend()
    plt.show()
    fig.savefig(Folder + 'SigmasL_tSquaredresiduals.pdf', bbox_inches='tight')

    Slice['Sigma_tL_t'] = FSigma(Slice.N, ParamSigmaL[0], ParamSigmaL[1], ParamSigmaL[2], ParamSigmaL[3], ParamSigmaL[4], ParamSigmaL[5], ParamSigmaL[6], ParamSigmaL[7], ParamSigmaL[8])
    Slice['Sigma_tL_t'] = np.sqrt(Slice['Sigma_tL_t'])
    Slice['DeltaL_t'] = Slice['SigmaDeltaL_t']/Slice['Sigma_tL_t']
    Slice['DeltaL_tSq'] = Slice['DeltaL_t']**2

    print(r"Mean of Deseasonalized residuals DeltaL_t: " + str(Slice['DeltaL_t'].mean()))
    print(r"Std of Deseasonalized residuals DeltaL_t: " + str(Slice['DeltaL_t'].std()))

    fig, ax = plt.subplots(figsize=(10,5))
    plt.title(r"Lévy increments $\Delta L_t$", fontdict = font)
    ax.set_ylabel('°C', rotation='horizontal', color='darkred', size = 13, labelpad=10)
    plt.plot(Slice.Date, Slice['DeltaL_t'], color='crimson', label = r"$\Delta L_t$", linewidth = 2.2)
```

```python
    plt.show()
    print(r"Mean of $\Delta L_t$ : " + str(Slice['DeltaL_t'].mean()))
    print(r"Std of $\Delta L_t$ : " + str(Slice['DeltaL_t'].std()))

    fig, ax = plt.subplots(figsize=(10,5))
    plt.title(r"Residuals $Z^{\diamond}_{t}$ and Lévy increments $\Delta L_t$", fontdict = font)
    ax.set_ylabel('°C', rotation='horizontal', color='darkred', size = 13, labelpad=10)
    ax.set_xlabel('Day', rotation='horizontal', size = 13, labelpad=10, color='darkred')
    plt.plot(Slice.Date, Slice['Z_t'], label =  r"$Z^{\diamond}_{t}$")
    plt.plot(Slice.Date, Slice['DeltaL_t'], color='crimson', label = r"$\Delta W_t$")
    plt.legend(fontsize = 'large', loc = 'upper right')
    plt.show()
    fig.savefig(Folder + 'ResidualsLévyincrements.png', bbox_inches='tight')

    NewMeanDayL_t = np.zeros(365)
    NewSquaredDayL_t = np.zeros(365)
    for i in np.arange(365):
        temp = Slice[Slice.N.mod(365) == i]
        NewMeanDayL_t[i] = temp['DeltaL_t'].mean()
        NewSquaredDayL_t[i] = temp['DeltaL_tSq'].mean()

    fig, ax = plt.subplots(figsize=(10,5))
    plt.title(r"Mean of the squared Lévy increments $\Delta L_t^2$ by day", fontdict = font)
    ax.set_xlabel('Day', rotation='horizontal', size = 13, labelpad=10, color='darkred')
    plt.plot(np.arange(365), NewSquaredDayL_t, color = 'crimson', linewidth = 2.2)
    plt.show()
    fig.savefig(Folder + 'MeanSquaredDeltaL_t.pdf', bbox_inches='tight')

    fig, ax = plt.subplots(figsize=(10,5))
    plt.title(r"Mean of the Lévy increments $\Delta L_t$ by day", fontdict = font)
    ax.set_xlabel('Day', rotation='horizontal', size = 13, labelpad=10, color='darkred')
    plt.plot(np.arange(365), NewMeanDayL_t, color = 'crimson', linewidth = 2.2)
    plt.show()
    fig.savefig(Folder + 'MeanDeltaL_t.pdf', bbox_inches='tight')

    #Autocorrelation
    fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12,3.3))
    plot_acf(Slice['DeltaW_t'][:-1].values, ax = ax1, lags=50)
    ax1.set_title(r"Autocorrelation  $\Delta W_t$", fontdict = font2)
    ax1.set_xlabel('Lag', rotation='horizontal', size = 13, labelpad=10, color='navy')
    plot_acf(Slice['DeltaL_t'][:-1].values, ax = ax2, lags=50) #, zero = False
    ax2.set_title(r"Autocorrelation $\Delta L_t$", fontdict = font2)
```

```
        ax2.set_xlabel('Lag', rotation='horizontal', size = 13, labelpad=10, color='navy')
        fig.savefig(Folder + 'Autocorrelation.pdf', bbox_inches='tight')
        plt.show()
        plt.close()

        L_t = Slice['DeltaL_t'][:-1]
        print("Fit Lévy : \n\n")
        FitLevy_rWrap(L_t, Folder)
        w = Slice['DeltaW_t'][:-1]
        Gaussian_fit(w, Folder)
        return Slice

    def FFT(self, period, freq):
        FromYear, ToYear = period
        Slice = self.Cut(FromYear, ToYear).copy()
        fft = np.fft.fft(Slice.Temp)
        delta_T = 1/365  # sampling interval
        N = Slice.Temp.size
        f = np.linspace(0, 1 / delta_T, N)
        fig, ax = plt.subplots(figsize=(10,5))
        plt.title("Discrete Fourier Transform", fontdict = font)
        ax.set_xlabel('Frequency [Hz]', color='darkred', size = 13, labelpad=10)
        ax.set_ylabel('Magnitude', color='darkred', size = 13, labelpad=10)
        delta_v = 1/(N*delta_T)
        lim = int(freq/delta_v)
        plt.bar(f[:lim], np.abs(fft)[:lim] * 1 / N, color ='r', width = 0.04)  # 1 / N is a normalization factor
        plt.show()
        fig.savefig(self.Folder + 'DFT.pdf', bbox_inches='tight')

Stockholm = TempSerie(StockholmData, "Stockholm")

Model2 = Stockholm.Fit((1960,2019), (2000,2019), False)

Model3 = Stockholm.Fit((1960,2019), (2000,2019), True)
```

B.2 Python code for the pricing of weather derivatives

```python
def GenPricingParam(Model, StartDate, EndDate):
    StartDate = pd.to_datetime(StartDate, format='%d/%m/%Y')
    PreviousDate = StartDate - pd.Timedelta('1 days')
    EndDate = pd.to_datetime(EndDate, format='%d/%m/%Y')
    t = Model.loc[StartDate, 'N']
    PrintStr = "The first day of the contract correspond to t = {}\n"
    print(PrintStr.format(str(t)))
    InitialDiff = Model.loc[PreviousDate, 'Detrended']
    PrintStr = "On the previous day, T_t - S(t) = {} \n"
    print(PrintStr.format(str(InitialDiff)))
    TimeDiff = EndDate - StartDate
    NumberOfDays = TimeDiff.days + 1
    PrintStr = "The maturity of the contract is {} days.\n"
    print(PrintStr.format(str(NumberOfDays)))
    return t, InitialDiff, NumberOfDays

@njit
def PricingVG(t, Maturity, InitialDiff, nDailySteps, nSim):
    dT = 1/nDailySteps
    def Strend(t):
        omega = 2*np.pi/365.0
        alpha = 6.1785162890049445
        beta = 9.842711993555477e-05
        theta1 = 10.17684885383261
        phi1 = -1.9358915022293823
        theta2 = -0.7672813386695645
        phi2 =  29.623776422246006
        value = alpha + beta*t + theta1*np.sin(omega*t+phi1) + theta2*np.sin(2*omega*t+phi2)
        return value
    def Kappa(t):
        omega = 2*np.pi/365.0
        Lambda = 0.23523671259171378
        phi1 = -0.029629918020637566
        varphi1 = 0.06496097376065202
        phi2 = -0.036865297056646366
        varphi2 = 0.26830186964048086
        phi3 = 0.01633859377487671
        varphi3 = 1.6017102936767609
        phi4 = -0.0007695873912343463
        varphi4 = 0.24609306152333735
        return Lambda + phi1*np.sin(omega*t+varphi1) + phi2*np.sin(2*omega*t+varphi2) + phi3*np.sin(3*omega*t+varphi3) + phi4*np.sin(4*omega*t+varphi4)
    def Sigma(t):
        omega = 2*np.pi/365.0
        Lambda = 6.137261912748916
        phi1 = 2.4610598949596327
        varphi1 = 1.2648519932150053
        phi2 = -1.6420730958940173
        varphi2 = -1.1105401910897663
        phi3 = 0.6901859440955523
        varphi3 = 0.8159967383683336
        phi4 = -0.0859968526532975
```

```
        varphi4 = -0.7976238695231654
        return np.sqrt(Lambda + phi1*np.sin(omega*t+varphi1) +
        ↪ phi2*np.sin(2*omega*t+varphi2) + phi3*np.sin(3*omega*t+varphi3)
        ↪ + phi4*np.sin(4*omega*t+varphi4))
    def Sim():
        #param
        Lambda = 3.021098
        alpha = 2.459503
        beta = -0.000062
        N = Maturity*nDailySteps
        #final value of the index for each sim
        CatValues = np.zeros(nSim)
        HDDValues = np.zeros(nSim)
        CDDValues = np.zeros(nSim)
        #Constant for each sim --> compute them only once
        vS = np.zeros(shape = N + 1)
        vKappa = np.zeros(shape = N + 1)
        vSigma = np.zeros(shape = N + 1)
        for i in prange(N + 1):
            vS[i] = Strend(t + i*dT)
            vKappa[i] = Kappa(t + i*dT)
            vSigma[i] = Sigma(t + i*dT)
        #Individual simulations.
        for i in prange(nSim):
            TildeT = np.zeros(shape = N + 1)
            TildeT[0] = InitialDiff
            Norm = np.random.normal(loc=0.0, scale=1.0, size=N)
            Gam = np.random.gamma(shape = Lambda*dT, scale = 2/(alpha**2 -
            ↪ beta**2), size=N)
            for z in range(N):
                TildeT[z + 1] =  TildeT[z] - vKappa[z]*TildeT[z]*dT +
                ↪ vSigma[z] * (beta * Gam[z] + np.sqrt(Gam[z]) * Norm[z])
            Tt = TildeT + vS
            print(Tt)
            print(TildeT)
            print(vS)
            #We have the temperature path for that sim
            #Compute DAT for each day
            #then compute value of the index
            CAT = np.zeros(shape = Maturity)
            HDD = np.zeros(shape = Maturity)
            CDD = np.zeros(shape = Maturity)
            for z in range(0, Maturity):
                position = z*nDailySteps
                DailyTemp =
                ↪ (Tt[(1+position):(1+position+nDailySteps)].max() +
                ↪ Tt[(1+position):(1+position+nDailySteps)].min())/2
                CAT[z] = DailyTemp
                HDD[z] = max(18 - DailyTemp, 0)
                CDD[z] = max(DailyTemp - 18, 0)
            CatValues[i] = CAT.sum()
            HDDValues[i] = HDD.sum()
            CDDValues[i] = CDD.sum()
        Indexes = CatValues, HDDValues, CDDValues
        return Indexes
    return Sim()
```

```python

@njit
def PricingVGConst(t, Maturity, InitialDiff, nDailySteps, nSim):
    dT = 1/nDailySteps
    Kappa = 0.22207568513566173
    def Strend(t):
        omega = 2*np.pi/365.0
        alpha = 6.1785162890049445
        beta = 9.842711993555477e-05
        theta1 = 10.17684885383261
        phi1 = -1.9358915022293823
        theta2 = -0.7672813386695645
        phi2 =  29.623776422246006
        value = alpha + beta*t + theta1*np.sin(omega*t+phi1) + theta2*np.sin(2*omega*t+phi2)
        return value
    def Sigma(t):
        omega = 2*np.pi/365.0
        Lambda = 6.07340448175054
        phi1 = 2.5125963720919233
        varphi1 = 1.2048650771125153
        phi2 = -1.5673162602384785
        varphi2 = -1.249777056029442
        phi3 = 0.6747580293367428
        varphi3 = 0.6593518799507244
        phi4 = -0.06116340915424446
        varphi4 = -0.8975968382757149
        return np.sqrt(Lambda + phi1*np.sin(omega*t+varphi1) + phi2*np.sin(2*omega*t+varphi2) + phi3*np.sin(3*omega*t+varphi3) + phi4*np.sin(4*omega*t+varphi4))
    def Sim():
        #param
        Lambda = 3.003475
        alpha = 2.451575
        beta = -0.000556
        N = Maturity*nDailySteps
        #final value of the index for each sim
        CatValues = np.zeros(nSim)
        HDDValues = np.zeros(nSim)
        CDDValues = np.zeros(nSim)
        #Constant for each sim --> compute them only once
        vS = np.zeros(shape = N + 1)
        vSigma = np.zeros(shape = N + 1)
        for i in prange(N + 1):
            vS[i] = Strend(t + i*dT)
            vSigma[i] = Sigma(t + i*dT)
        #Individual simulations.
        for i in prange(nSim):
            TildeT = np.zeros(shape = N + 1)
            TildeT[0] = InitialDiff
            Norm = np.random.normal(loc=0.0, scale=1.0, size=N)
            Gam = np.random.gamma(shape = Lambda*dT, scale = 2/(alpha**2 - beta**2), size=N)
            for z in range(N):
                TildeT[z + 1] = TildeT[z] - Kappa*TildeT[z]*dT + vSigma[z] * (beta*Gam[z] + np.sqrt(Gam[z]) * Norm[z])
```

```
                Tt = TildeT + vS
                #We have the temperature path for that sim
                #Compute DAT for each day
                #then compute value of the index
                CAT = np.zeros(shape = Maturity)
                HDD = np.zeros(shape = Maturity)
                CDD = np.zeros(shape = Maturity)
                for z in range(0, Maturity):
                    position = z*nDailySteps
                    DailyTemp =
                     ↪  (Tt[(1+position):(1+position+nDailySteps)].max() +
                     ↪  Tt[(1+position):(1+position+nDailySteps)].min())/2
                    CAT[z] = DailyTemp
                    HDD[z] = max(18 - DailyTemp, 0)
                    CDD[z] = max(DailyTemp - 18, 0)
                CatValues[i] = CAT.sum()
                HDDValues[i] = HDD.sum()
                CDDValues[i] = CDD.sum()
            Indexes = CatValues, HDDValues, CDDValues
            return Indexes
        return Sim()

@njit
def PricingNIG(t, Maturity, InitialDiff, nDailySteps, nSim):
    dT = 1/nDailySteps
    def Strend(t):
        omega = 2*np.pi/365.0
        alpha = 6.1785162890049445
        beta = 9.842711993555477e-05
        theta1 = 10.17684885383261
        phi1 = -1.9358915022293823
        theta2 = -0.7672813386695645
        phi2 =  29.623776422246006
        value = alpha + beta*t + theta1*np.sin(omega*t+phi1) +
         ↪  theta2*np.sin(2*omega*t+phi2)
        return value
    def Kappa(t):
        omega = 2*np.pi/365.0
        Lambda = 0.23523671259171378
        phi1 = -0.029629918020637566
        varphi1 = 0.06496097376065202
        phi2 = -0.036865297056646366
        varphi2 = 0.26830186964048086
        phi3 = 0.016338593774876 71
        varphi3 = 1.6017102936767609
        phi4 = -0.0007695873912343463
        varphi4 = 0.24609306152333735
        return Lambda + phi1*np.sin(omega*t+varphi1) +
         ↪  phi2*np.sin(2*omega*t+varphi2) + phi3*np.sin(3*omega*t+varphi3)
         ↪  + phi4*np.sin(4*omega*t+varphi4)
    def Sigma(t):
        omega = 2*np.pi/365.0
        Lambda = 6.137261912748916
        phi1 = 2.4610598949596327
        varphi1 = 1.2648519932150053
        phi2 = -1.6420730958940173
```

```
        varphi2 = -1.1105401910897663
        phi3 = 0.6901859440955523
        varphi3 = 0.8159967383683336
        phi4 = -0.0859968526532975
        varphi4 = -0.7976238695231654
        return np.sqrt(Lambda + phi1*np.sin(omega*t+varphi1) + phi2*np.sin(2*omega*t+varphi2) + phi3*np.sin(3*omega*t+varphi3) + phi4*np.sin(4*omega*t+varphi4))
    def Sim():
        #param
        alpha = 1.618664
        beta = -0.000062
        delta =  1.621384
        #Want to run
        N = Maturity*nDailySteps
        #final value of the index for each sim
        CatValues = np.zeros(nSim)
        HDDValues = np.zeros(nSim)
        CDDValues = np.zeros(nSim)
        #Constant for each sim --> compute them only once
        vS = np.zeros(shape = N + 1)
        vKappa = np.zeros(shape = N + 1)
        vSigma = np.zeros(shape = N + 1)
        for i in prange(N + 1):
            vS[i] = Strend(t + i*dT)
            vKappa[i] = Kappa(t + i*dT)
            vSigma[i] = Sigma(t + i*dT)
        #Individual simulations.
        for i in prange(nSim):
            TildeT = np.zeros(shape = N + 1)
            TildeT[0] = InitialDiff
            Norm = np.random.normal(loc=0.0, scale=1.0, size=N)
            #note conversion to numpy implementation(wald)
            #scale = lambda = delta**2 --> (delta*dt)**2
            #mean = mu = delta/gamma ----> delta*dt/sqrt(alpha**2 - beta**2)
            NIG = np.random.wald(mean = (delta*dT)**2, scale = delta*dT/np.sqrt(alpha**2 - beta**2), size=N)
            for z in range(N):
                TildeT[z + 1] =  TildeT[z] - vKappa[z]*TildeT[z]*dT + vSigma[z] * (beta * NIG[z] + np.sqrt(NIG[z]) * Norm[z])
            Tt = TildeT + vS
            print(Tt)
            print(TildeT)
            print(vS)
            #We have the temperature path for that sim
            #Compute DAT for each day
            #then compute value of the index
            CAT = np.zeros(shape = Maturity)
            HDD = np.zeros(shape = Maturity)
            CDD = np.zeros(shape = Maturity)
            for z in range(0, Maturity):
                position = z*nDailySteps
                DailyTemp = (Tt[(1+position):(1+position+nDailySteps)].max() + Tt[(1+position):(1+position+nDailySteps)].min())/2
```

```
                CAT[z] = DailyTemp
                HDD[z] = max(18 - DailyTemp, 0)
                CDD[z] = max(DailyTemp - 18, 0)
            CatValues[i] = CAT.sum()
            HDDValues[i] = HDD.sum()
            CDDValues[i] = CDD.sum()
        Indexes = CatValues, HDDValues, CDDValues
        return Indexes
    return Sim()

@njit
def PricingNIGConst(t, Maturity, InitialDiff, nDailySteps, nSim):
    dT = 1/nDailySteps
    Kappa = 0.22207568513566173
    def Strend(t):
        omega = 2*np.pi/365.0
        alpha = 6.1785162890049445
        beta = 9.842711993555477e-05
        theta1 = 10.17684885383261
        phi1 = -1.9358915022293823
        theta2 = -0.7672813386695645
        phi2 =  29.623776422246006
        value = alpha + beta*t + theta1*np.sin(omega*t+phi1) + theta2*np.sin(2*omega*t+phi2)
        return value
    def Sigma(t):
        omega = 2*np.pi/365.0
        Lambda = 6.07340448175054
        phi1 = 2.5125963720919233
        varphi1 = 1.2048650771125153
        phi2 = -1.5673162602384785
        varphi2 = -1.249777056029442
        phi3 = 0.6747580293367428
        varphi3 = 0.6593518799507244
        phi4 = -0.06116340915424446
        varphi4 = -0.8975968382757149
        return np.sqrt(Lambda + phi1*np.sin(omega*t+varphi1) + phi2*np.sin(2*omega*t+varphi2) + phi3*np.sin(3*omega*t+varphi3) + phi4*np.sin(4*omega*t+varphi4))
    def Sim():
        #param
        alpha = 1.618144
        beta = 0.000028
        delta =  1.620796
        N = Maturity*nDailySteps
        #final value of the index for each sim
        CatValues = np.zeros(nSim)
        HDDValues = np.zeros(nSim)
        CDDValues = np.zeros(nSim)
        #Constant for each sim --> compute them only once
        vS = np.zeros(shape = N + 1)
        vSigma = np.zeros(shape = N + 1)
        for i in prange(N + 1):
            vS[i] = Strend(t + i*dT)
            vSigma[i] = Sigma(t + i*dT)
        #Individual simulations.
```

```python
        for i in prange(nSim):
            TildeT = np.zeros(shape = N + 1)
            TildeT[0] = InitialDiff
            Norm = np.random.normal(loc=0.0, scale=1.0, size=N)
            #note conversion to numpy implementation(wald)
            #scale = lambda = delta**2 --> (delta*dt)**2
            #mean = mu = delta/gamma ----> delta*dt/sqrt(alpha**2 - beta**2)
            NIG = np.random.wald(mean = (delta*dT)**2, scale = delta*dT/np.sqrt(alpha**2 - beta**2), size=N)
            for z in range(N):
                TildeT[z + 1] = TildeT[z] - Kappa*TildeT[z]*dT + vSigma[z] * (beta * NIG[z] + np.sqrt(NIG[z]) * Norm[z])
            Tt = TildeT + vS
            #We have the temperature path for that sim
            #Compute DAT for each day
            #then compute value of the index
            CAT = np.zeros(shape = Maturity)
            HDD = np.zeros(shape = Maturity)
            CDD = np.zeros(shape = Maturity)
            for z in range(0, Maturity):
                position = z*nDailySteps
                DailyTemp = (Tt[(1+position):(1+position+nDailySteps)].max() + Tt[(1+position):(1+position+nDailySteps)].min())/2
                CAT[z] = DailyTemp
                HDD[z] = max(18 - DailyTemp, 0)
                CDD[z] = max(DailyTemp - 18, 0)
            CatValues[i] = CAT.sum()
            HDDValues[i] = HDD.sum()
            CDDValues[i] = CDD.sum()
        Indexes = CatValues, HDDValues, CDDValues
        return Indexes
    return Sim()

@njit
def PricingBM(t, Maturity, InitialDiff, nDailySteps, nSim):
    dT = 1/nDailySteps
    def Strend(t):
        omega = 2*np.pi/365.0
        alpha = 6.1785162890049445
        beta = 9.842711993555477e-05
        theta1 = 10.17684885383261
        phi1 = -1.9358915022293823
        theta2 = -0.7672813386695645
        phi2 =  29.623776422246006
        value = alpha + beta*t + theta1*np.sin(omega*t+phi1) + theta2*np.sin(2*omega*t+phi2)
        return value
    def Kappa(t):
        omega = 2*np.pi/365.0
        Lambda = 0.23523671259171378
        phi1 = -0.029629918020637566
        varphi1 = 0.06496097376065202
        phi2 = -0.036865297056646366
        varphi2 = 0.26830186964048086
```

```
        phi3 = 0.016338593774876710
        varphi3 = 1.6017102936767609
        phi4 = -0.00076958739123434463
        varphi4 = 0.24609306152333735
        return Lambda + phi1*np.sin(omega*t+varphi1) +
        ↪  phi2*np.sin(2*omega*t+varphi2) + phi3*np.sin(3*omega*t+varphi3)
        ↪  + phi4*np.sin(4*omega*t+varphi4)
    def Sigma(t):
        omega = 2*np.pi/365.0
        Lambda = 5.536528121723934
        phi1 = 2.267267625887284
        varphi1 = 1.2505497831125316
        phi2 = -1.4801537100366213
        varphi2 = -1.1533860842826675
        phi3 = 0.6459907702278879
        varphi3 = 0.8227225422652261
        phi4 = -0.05130775413724097
        varphi4 = -0.970003080373328
        return np.sqrt(Lambda + phi1*np.sin(omega*t+varphi1) +
        ↪  phi2*np.sin(2*omega*t+varphi2) + phi3*np.sin(3*omega*t+varphi3)
        ↪  + phi4*np.sin(4*omega*t+varphi4))
    def Sim():
        #param
        N = Maturity*nDailySteps
        #final value of the index for each sim
        CatValues = np.zeros(nSim)
        HDDValues = np.zeros(nSim)
        CDDValues = np.zeros(nSim)
        #Constant for each sim --> compute them only once
        vS = np.zeros(shape = N + 1)
        vKappa = np.zeros(shape = N + 1)
        vSigma = np.zeros(shape = N + 1)
        for i in prange(N + 1):
            vS[i] = Strend(t + i*dT)
            vKappa[i] = Kappa(t + i*dT)
            vSigma[i] = Sigma(t + i*dT)
        #Individual simulations.
        for i in prange(nSim):
            TildeT = np.zeros(shape = N + 1)
            TildeT[0] = InitialDiff
            Norm = np.random.normal(loc=0.0, scale=1.0, size=N)
            for z in range(N):
                TildeT[z + 1] = TildeT[z] - vKappa[z]*TildeT[z] * dT +
                ↪  vSigma[z] * np.sqrt(dT) * Norm[z]
            Tt = TildeT + vS
            #We have the temperature path for that sim
            #Compute DAT for each day
            #then compute value of the index
            CAT = np.zeros(shape = Maturity)
            HDD = np.zeros(shape = Maturity)
            CDD = np.zeros(shape = Maturity)
            for z in range(0, Maturity):
                position = z*nDailySteps
                DailyTemp =
                ↪  (Tt[(1+position):(1+position+nDailySteps)].max() +
                ↪  Tt[(1+position):(1+position+nDailySteps)].min())/2
```

```
                CAT[z] = DailyTemp
                HDD[z] = max(18 - DailyTemp, 0)
                CDD[z] = max(DailyTemp - 18, 0)
            CatValues[i] = CAT.sum()
            HDDValues[i] = HDD.sum()
            CDDValues[i] = CDD.sum()
        Indexes = CatValues, HDDValues, CDDValues
        return Indexes
    return Sim()


@njit
def PricingBMConst(t, Maturity, InitialDiff, nDailySteps, nSim):
    dT = 1/nDailySteps
    Kappa = 0.22207568513566173
    def Strend(t):
        omega = 2*np.pi/365.0
        alpha = 6.1785162890049445
        beta = 9.842711993555477e-05
        theta1 = 10.17684885383261
        phi1 = -1.9358915022293823
        theta2 = -0.7672813386695645
        phi2 =  29.623776422246006
        value = alpha + beta*t + theta1*np.sin(omega*t+phi1) + theta2*np.sin(2*omega*t+phi2)
        return value
    def Sigma(t):
        omega = 2*np.pi/365.0
        Lambda = 5.486248146862906
        phi1 = 2.3236689385407727
        varphi1 = 1.19130990938617
        phi2 = -1.4143798156323266
        varphi2 = -1.3008889365029273
        phi3 = 0.6257781999463782
        varphi3 = 0.6458663557462555
        phi4 = -0.02470285541993388
        varphi4 = -0.9210284018231684
        return np.sqrt(Lambda + phi1*np.sin(omega*t+varphi1) + phi2*np.sin(2*omega*t+varphi2) + phi3*np.sin(3*omega*t+varphi3) + phi4*np.sin(4*omega*t+varphi4))
    def Sim():
        #param
        N = Maturity*nDailySteps
        #final value of the index for each sim
        CatValues = np.zeros(nSim)
        HDDValues = np.zeros(nSim)
        CDDValues = np.zeros(nSim)
        #Constant for each sim --> compute them only once
        vS = np.zeros(shape = N + 1)
        vSigma = np.zeros(shape = N + 1)
        for i in prange(N + 1):
            vS[i] = Strend(t + i*dT)
            vSigma[i] = Sigma(t + i*dT)
        #Individual simulations.
        for i in prange(nSim):
            TildeT = np.zeros(shape = N + 1)
```

```
            TildeT[0] = InitialDiff
            Norm = np.random.normal(loc=0.0, scale=1.0, size=N)
            Gam = np.random.gamma(shape = Lambda*dT, scale = 2/(alpha**2 - beta**2), size=N)
            for z in range(N):
                TildeT[z + 1] = TildeT[z] - Kappa*TildeT[z]*dT + vSigma[z] * np.sqrt(dT) * Norm[z]
            Tt = TildeT + vS
            #We have the temperature path for that sim
            #Compute DAT for each day
            #then compute value of the index
            CAT = np.zeros(shape = Maturity)
            HDD = np.zeros(shape = Maturity)
            CDD = np.zeros(shape = Maturity)
            for z in range(0, Maturity):
                position = z*nDailySteps
                DailyTemp = (Tt[(1+position):(1+position+nDailySteps)].max() + Tt[(1+position):(1+position+nDailySteps)].min())/2
                CAT[z] = DailyTemp
                HDD[z] = max(18 - DailyTemp, 0)
                CDD[z] = max(DailyTemp - 18, 0)
            CatValues[i] = CAT.sum()
            HDDValues[i] = HDD.sum()
            CDDValues[i] = CDD.sum()
        Indexes = CatValues, HDDValues, CDDValues
        return Indexes
    return Sim()

def Results(Arrs, Type, period):
    #Plot CAT
    g = plt.figure(figsize=(12,6))
    sns.kdeplot(Arrs[0], shade=False, label = 'Model1', linewidth = 2, color = 'C2')
    sns.kdeplot(Arrs[1], shade=False, label = 'Model2', linewidth = 2, color = 'C3')
    sns.kdeplot(Arrs[2], shade=False, label = 'Model3', linewidth = 2, color = 'C4')
    sns.kdeplot(Arrs[3], shade=False, label = 'Model4', linewidth = 2, color = 'C1')
    plt.title('Density Plot ' + Type + ' ' + period, fontdict = font2)
    plt.xlabel(Type, color='Navy', size = 13, labelpad=10)
    plt.ylabel('Density', color='Navy', size = 13, labelpad=10)
    plt.legend(fontsize = 'x-large')
    plt.savefig('C:/Users/nicol/Dropbox/Mémoire/Tempfig/Density'+ Type + period + '.pdf', bbox_inches='tight')
    plt.show()
    plt.close()
    #Value of the Future
    print('Value of the Future for Model1 : ' + str(Arrs[0].mean()))
    print('Value of the Future for Model2 : ' + str(Arrs[1].mean()))
    print('Value of the Future for Model3 : ' + str(Arrs[2].mean()))
    print('Value of the Future for Model4 : ' + str(Arrs[3].mean()))
    #Value of the Option
    MaxVal = max(np.max(Arrs[0]), np.max(Arrs[1]), np.max(Arrs[2]), np.max(Arrs[3]))
```

```
    Strikes = np.linspace(0, MaxVal, 201)
    Call1 = np.zeros(shape = 201)
    Call2 = np.zeros(shape = 201)
    Call3 = np.zeros(shape = 201)
    Call4 = np.zeros(shape = 201)
    Put1 = np.zeros(shape = 201)
    Put2 = np.zeros(shape = 201)
    Put3 = np.zeros(shape = 201)
    Put4 = np.zeros(shape = 201)
    for i in range(201):
        Strike = Strikes[i]
        Call1[i] = np.maximum(Arrs[0] - Strike, 0).mean()
        Put1[i] = np.maximum(Strike - Arrs[0], 0).mean()
        Call2[i] = np.maximum(Arrs[1] - Strike, 0).mean()
        Put2[i] = np.maximum(Strike - Arrs[1], 0).mean()
        Call3[i] = np.maximum(Arrs[2] - Strike, 0).mean()
        Put3[i] = np.maximum(Strike - Arrs[2], 0).mean()
        Call4[i] = np.maximum(Arrs[3] - Strike, 0).mean()
        Put4[i] = np.maximum(Strike - Arrs[3], 0).mean()
    #Value of the call option plot
    g = plt.figure(figsize=(12,6))
    plt.plot(Strikes, Call1, label = 'Model1', linewidth = 2, color = 'C2')
    plt.plot(Strikes, Call2, label = 'Model2', linewidth = 2, color = 'C3')
    plt.plot(Strikes, Call3, label = 'Model3', linewidth = 2, color = 'C4')
    plt.plot(Strikes, Call4, label = 'Model4', linewidth = 2, color = 'C1')
    plt.title('Value of the Call as a function of the Strike', fontdict = font2)
    plt.xlabel('Strike', color='Navy', size = 13, labelpad=10)
    plt.ylabel('Value', color='Navy', size = 13, labelpad=10)
    plt.legend(fontsize = 'x-large')
    plt.savefig('C:/Users/nicol/Dropbox/Mémoire/Tempfig/Call'+ Type + period + '.pdf', bbox_inches='tight')
    plt.show()
    plt.close()
    #Value of the Put option plot
    g = plt.figure(figsize=(12,6))
    plt.plot(Strikes, Put1, label = 'Model1', linewidth = 2, color = 'C2')
    plt.plot(Strikes, Put2, label = 'Model2', linewidth = 2, color = 'C3')
    plt.plot(Strikes, Put3, label = 'Model3', linewidth = 2, color = 'C4')
    plt.plot(Strikes, Put4, label = 'Model4', linewidth = 2, color = 'C1')
    plt.title('Value of the Put as a function of the Strike', fontdict = font2)
    plt.xlabel('Strike', color='Navy', size = 13, labelpad=10)
    plt.ylabel('Value', color='Navy', size = 13, labelpad=10)
    plt.legend(fontsize = 'x-large')
    plt.savefig('C:/Users/nicol/Dropbox/Mémoire/Tempfig/Put'+ Type + period + '.pdf', bbox_inches='tight')
    plt.show()
    plt.close()
    #Quantiles
    print('Value of the quantiles [0.001, 0.005, 0.01, 0.99, 0.995, 0.999] : \n')
    print('Model 1 : ')
    print(np.quantile(a = Arrs[0], q = [0.001, 0.005, 0.01, 0.99, 0.995, 0.999]))
    print('Model 2 : ')
```

```
    print(np.quantile(a = Arrs[1], q =[0.001, 0.005, 0.01, 0.99, 0.995, 0.999]))
    print('Model 3 : ')
    print(np.quantile(a = Arrs[2], q = [0.001, 0.005, 0.01, 0.99, 0.995, 0.999]))
    print('Model 4 : ')
    print(np.quantile(a = Arrs[3], q =[0.001, 0.005, 0.01, 0.99, 0.995, 0.999]))
    return None


#Contract Feb
t, InitialDiff, NumberOfDays = GenPricingParam(Model2, '01/02/2019', '28/02/2019')

#Feb
#GH + kappa(t)
Result5 = PricingVG(t = 21567, Maturity = 28, InitialDiff = -0.877290606124236 , nDailySteps = 100, nSim = 200000)
CatValues5, HDDValues5, CDDValues5 = Result5

#GH + kappa
Result6 = PricingVGConst(t = 21567, Maturity = 28, InitialDiff = -0.877290606124236 , nDailySteps = 100, nSim = 200000)
CatValues6, HDDValues6, CDDValues6 = Result6

#BM + kappa(t)
Result7 = PricingBM(t = 21567, Maturity = 28, InitialDiff =-0.877290606124236 , nDailySteps = 100, nSim = 200000)
CatValues7, HDDValues7, CDDValues7 = Result7

#BM + kappa
Result8 = PricingBMConst(t = 21567, Maturity = 28, InitialDiff = -0.877290606124236 , nDailySteps = 100, nSim = 200000)
CatValues8, HDDValues8, CDDValues8 = Result8

Results([CatValues5, CatValues6, CatValues7, CatValues8], 'CAT Index', 'Feb')


#Contract Sept - Nov
t, InitialDiff, NumberOfDays = GenPricingParam(Model2, '01/09/2019', '30/11/2019')


#Sept - Nov
#GH + kappa(t)
Result1 = PricingVG(t = 21779, Maturity = 91, InitialDiff = 3.0614559677297315, nDailySteps = 100, nSim = 200000)
CatValues1, HDDValues1, CDDValues1 = Result1

#GH + kappa
Result2 = PricingVGConst(t = 21779, Maturity = 91, InitialDiff = 3.0614559677297315, nDailySteps = 100, nSim = 200000)
CatValues2, HDDValues2, CDDValues2 = Result2

```

```
#BM + kappa(t)
Result3 = PricingBM(t = 21779, Maturity = 91, InitialDiff =
    3.0614559677297315, nDailySteps = 100, nSim = 200000)
CatValues3, HDDValues3, CDDValues3 = Result3

#BM + kappa
Result4 = PricingBMConst(t = 21779, Maturity = 91, InitialDiff =
    3.0614559677297315, nDailySteps = 100, nSim = 200000)
CatValues4, HDDValues4, CDDValues4 = Result4

Results([CatValues1, CatValues2, CatValues3, CatValues4], 'CAT Index', 'Sep
    - Nov')


#Contract Jun - Aug
t, InitialDiff, NumberOfDays = GenPricingParam(Model2, '01/06/2019',
    '31/08/2019')


#Jun - Aug
#GH + kappa(t)
Result9 = PricingVG(t = 21687, Maturity = 92, InitialDiff =
    -0.8812524168644469, nDailySteps = 100, nSim = 200000)
CatValues9, HDDValues9, CDDValues9 = Result9

#GH + kappa
Result10 = PricingVGConst(t = 21687, Maturity = 92, InitialDiff =
    -0.8812524168644469, nDailySteps = 100, nSim = 200000)
CatValues10, HDDValues10, CDDValues10 = Result10

#BM + kappa(t)
Result11 = PricingBM(t = 21687, Maturity = 92, InitialDiff =
    -0.8812524168644469, nDailySteps = 100, nSim = 200000)
CatValues11, HDDValues11, CDDValues11 = Result11

#BM + kappa
Result12 = PricingBMConst(t = 21687, Maturity = 92, InitialDiff =
    -0.8812524168644469, nDailySteps = 100, nSim = 200000)
CatValues12, HDDValues12, CDDValues12 = Result12

Results([HDDValues9, HDDValues10, HDDValues11, HDDValues12], 'HDD Index',
    'Jun - Aug')

Results([CDDValues9, CDDValues10, CDDValues11, CDDValues12], 'CDD Index',
    'Jun - Aug')

Results([CatValues9, CatValues10, CatValues11, CatValues12], 'CAT Index',
    'Jun - Aug')

@njit
def Test(t, Maturity, InitialDiff, nDailySteps, nSim):
    dT = 1/nDailySteps
    def Strend(t):
        omega = 2*np.pi/365.0
        alpha = 6.1785162890049445
        beta = 9.842711993555477e-05
```

```
        theta1 = 10.17684885383261
        phi1 = -1.9358915022293823
        theta2 = -0.7672813386695645
        phi2 =  29.623776422246006
        value = alpha + beta*t + theta1*np.sin(omega*t+phi1) + theta2*np.sin(2*omega*t+phi2)
        return value
    def Kappa(t):
        omega = 2*np.pi/365.0
        Lambda = 0.23523671259171378
        phi1 = -0.029629918020637566
        varphi1 = 0.06496097376065202
        phi2 = -0.036865297056646366
        varphi2 = 0.26830186964048086
        phi3 = 0.01633859377487671
        varphi3 = 1.6017102936767609
        phi4 = -0.0007695873912343463
        varphi4 = 0.24609306152333735
        return Lambda + phi1*np.sin(omega*t+varphi1) + phi2*np.sin(2*omega*t+varphi2) + phi3*np.sin(3*omega*t+varphi3) + phi4*np.sin(4*omega*t+varphi4)
    def SigmaSquarred(t):
        omega = 2*np.pi/365.0
        Lambda = 6.137261912748916
        phi1 = 2.4610598949596327
        varphi1 = 1.2648519932150053
        phi2 = -1.6420730958940173
        varphi2 = -1.1105401910897663
        phi3 = 0.6901859440955523
        varphi3 = 0.8159967383683336
        phi4 = -0.0859968526532975
        varphi4 = -0.7976238695231654
        return Lambda + phi1*np.sin(omega*t+varphi1) + phi2*np.sin(2*omega*t+varphi2) + phi3*np.sin(3*omega*t+varphi3) + phi4*np.sin(4*omega*t+varphi4)
    def path():
        N = Maturity*nDailySteps
        vS = np.zeros(shape = N + 1)
        vKappa = np.zeros(shape = N + 1)
        vSigma = np.zeros(shape = N + 1)
        for i in prange(N + 1):
            vS[i] = Strend(t + i*dT)
            vKappa[i] = Kappa(t + i*dT)
            vSigma[i] = SigmaSquarred(t + i*dT)
        test = vS, vKappa, vSigma
        return test
    return path()

test = Test(t = 18251, Maturity = 365, InitialDiff = -6.761597896470001, nDailySteps = 50, nSim = 100)
vS, vKappa, vSigma = test
#Plot S(t)
fig, ax = plt.subplots(figsize=(10,5))
plt.title(r"Test $S(t)$", fontdict = font)
ax.set_xlabel('°C', rotation='horizontal', color='darkred', size = 13, labelpad=10)
```

```
plt.plot(vS, color='crimson', label = r"$S(t)$")
plt.legend(fontsize = 'large', loc = 'upper right')
plt.show()
fig.savefig('C:/Users/nicol/Dropbox/Mémoire/Tempfig/TestSt.pdf', bbox_inches='tight')
#Plot Kappa(t)
fig, ax = plt.subplots(figsize=(10,5))
plt.title(r"Test $\kappa (t)$", fontdict = font)
plt.plot(vKappa, color='crimson', label = r"$\kappa (t)$")
plt.legend(fontsize = 'large', loc = 'upper right')
plt.ylim((0.0, 0.6))
plt.show()
fig.savefig('C:/Users/nicol/Dropbox/Mémoire/Tempfig/Kappa.pdf', bbox_inches='tight')
#Plot sigma(t)
fig, ax = plt.subplots(figsize=(10,5))
plt.title(r"Test $\sigma^2(t)$", fontdict = font)
plt.plot(vSigma, color='crimson', label = r"$\sigma^2(t)$")
plt.legend(fontsize = 'large', loc = 'upper right')
plt.ylim((2, 14))
plt.show()
fig.savefig('C:/Users/nicol/Dropbox/Mémoire/Tempfig/TestSigma.pdf', bbox_inches='tight')

def npTEST(Nsim, Delta_t, Lambda, alpha, beta):
    Varray = np.zeros(Nsim)
    start_time = time.time()
    N = np.random.normal(loc=0.0, scale=1.0, size=Nsim)
    print("Normal")
    print("Mean : " +str(N.mean()))
    print("Var : " +str(N.var()))
    X = scs.gamma.rvs(a = Lambda*Delta_t, loc = 0, scale = 2/(alpha**2 - beta**2), size=Nsim)
    #X = np.random.gamma(shape = Lambda*Delta_t, scale = 1/np.sqrt(alpha**2 - beta**2), size=Nsim)
    print("gamma")
    print("Mean : " +str(X.mean()))
    print("Var : " +str(X.var()))
    Varray = beta*X + np.sqrt(X) * N
    PrintStr = "Running time :  {}\n"
    print(PrintStr.format(str(time.time() - start_time)))
    #print("time.time() - start_time)
    return Varray

test5 = npTEST(Nsim = 10000000, Delta_t = 0.1, Lambda = 3.020783, alpha = 2.458418, beta = 1.681182e-05)

print(test5.mean())
print(test5.var())


Lambda = 3.020783 *0.1
alpha = 2.458418
beta = 1.681182e-05
gamma = np.sqrt(alpha**2 - beta**2)
GammaScale = 2/(alpha**2 - beta**2)
```

```python

#gamma
print("gamma test")
print("Mean : " +str(Lambda*GammaScale))
print("Var : " +str(Lambda*GammaScale**2))

#Variance gamma
print("VG test")
print("Mean : " +str(2*beta*Lambda/gamma))

#variance
print("var : " + str((2*Lambda/(gamma**2))*(1 + (2*(beta/gamma)**2))))
print("var : " + str((2*Lambda/gamma**2 ) + 4*Lambda*beta**2/(gamma**4)))
```

B.3 C++ code for the pricing of weather derivatives

```cpp
#define NOMINMAX
#define _USE_MATH_DEFINES
#ifndef __wtypes_h__
#include <wtypes.h>
#endif
#ifndef __WINDEF_
#include <windef.h>
#endif


#include <cmath>
#include <iostream>
#include <string>
#include <random>
#include <algorithm>
#include <fstream>
#include <iomanip>
#include <stdio.h>
#include <math.h>
#include <boost/random.hpp>
#include <ctime>
#include <cstdint>

CONST double omega = 2 * M_PI / 365.0;
typedef  boost::random::lagged_fibonacci_01_engine< double, 48, 607, 273 > lagged_fibonacci607;

class Sinusoid {
public:
    double alpha;
    double beta;
    double phi1, varphi1, phi2, varphi2, phi3, varphi3, phi4, varphi4;
    Sinusoid() = delete;
    Sinusoid(double a, double b, double p1, double v1, double p2, double v2, double p3, double v3, double p4, double v4) :
        alpha{ a }, beta{ b }, phi1{ p1 }, varphi1{ v1 }, phi2{ p2 }, varphi2{ v2 }, phi3{ p3 }, varphi3{ v3 }, phi4{ p4 }, varphi4{ v4 } {};
    double Value(double t) {
        return alpha + beta * t + phi1 * sin(omega * t + varphi1) + phi2 * sin(2 * omega * t + varphi2) + phi3 * sin(3 * omega * t + varphi3) + phi4 * sin(4 * omega * t + varphi4);
    }
};

class Levy {
public:
    virtual ~Levy() = default;
    virtual double gen() = 0;
    virtual void SetDelta_t(double Delta_t) = 0;
};

class NIG :public Levy
{
```

```
public:
    double alpha;
    double beta;
    double delta;
    double IgShape;
    double IgMean;
    lagged_fibonacci607 generator;
    boost::random::uniform_real_distribution<> UNI;
    boost::random::normal_distribution<> Norm;
    boost::variate_generator<lagged_fibonacci607&,
     ↪  boost::random::uniform_real_distribution<> > rUNI;
    boost::variate_generator<lagged_fibonacci607&,
     ↪  boost::random::normal_distribution<> > rNorm;
    NIG() = delete;
    NIG(double pAlpha, double pBeta, double pDelta) : alpha{ pAlpha },
     ↪  beta{ pBeta }, delta{ pDelta },
        IgShape{ pow(delta , 2.0) },
        IgMean{ delta / sqrt(pow(alpha, 2.0) - pow(beta, 2.0)) },
        generator{ lagged_fibonacci607() },
        UNI{ boost::random::uniform_real_distribution<>(0.0, 1.0) },
        Norm{ boost::random::normal_distribution<>(0.0, 1.0) },
        rUNI{ boost::variate_generator<lagged_fibonacci607&,
         ↪  boost::random::uniform_real_distribution<> >(generator, UNI) },
        rNorm{ boost::variate_generator<lagged_fibonacci607&,
         ↪  boost::random::normal_distribution<> >(generator, Norm) }
    {};
    void SetDelta_t(double Delta_t) override {
        IgShape = pow((delta * Delta_t), 2.0);
        IgMean = delta * Delta_t / sqrt(pow(alpha, 2.0) - pow(beta, 2.0));
    };

    double gen() override final {

        double nu = rNorm();
        double y = pow(nu, 2.0);
        double x = IgMean + ((IgMean * IgMean * y) / (2 * IgShape)) -
         ↪  (IgMean / (2 * IgShape)) * sqrt(4 * IgMean * IgShape * y +
         ↪  IgMean * IgMean * y * y);
        double z = rUNI();
        double I;
        if (z <= (IgMean / (IgMean + x))) {
            I = x;
        }
        else {
            I = (IgMean * IgMean) / x;
        }
        double n = rNorm();
        return beta * I + sqrt(I) * n;
    };
};

class VG :public Levy {
public:
    double lambda;
    double alpha;
    double beta;
```

```
    double GammaShape;
    double GammaScale;
    lagged_fibonacci607 generator;
    boost::random::gamma_distribution<> Gam;
    boost::random::normal_distribution<> Norm;
    boost::variate_generator<lagged_fibonacci607&,
     ↪  boost::random::gamma_distribution<> > rGam;
    boost::variate_generator<lagged_fibonacci607&,
     ↪  boost::random::normal_distribution<> > rNorm;
    VG() = delete;
    VG(double pLambda, double pAlpha, double pBeta) : lambda{ pLambda },
     ↪  alpha{ pAlpha }, beta{ pBeta },
        GammaScale{ lambda },
        GammaShape{ 2.0 / (pow(alpha, 2.0) - pow(pBeta, 2.0)) },
        generator{ lagged_fibonacci607() },
        Gam{ boost::random::gamma_distribution<>(GammaShape, GammaScale) },
        Norm{ boost::random::normal_distribution<>(0.0, 1.0) },
        rGam{ boost::variate_generator<lagged_fibonacci607&,
         ↪  boost::random::gamma_distribution<> >(generator, Gam) },
        rNorm{ boost::variate_generator<lagged_fibonacci607&,
         ↪  boost::random::normal_distribution<> >(generator, Norm) }
    {};
    void SetDelta_t(double Delta_t) override final {
        GammaScale = lambda * Delta_t;

         ↪  rGam.distribution().param(boost::random::gamma_distribution<>::param_type(Ga
         ↪  GammaShape));
    };

    double gen() override {
        double g = rGam();
        double n = rNorm();
        return beta * g + sqrt(g) * n;
    };
};

class WeatherDerivative {
public:
    Sinusoid S;
    Sinusoid Kappa;
    Sinusoid Sigma;
    Levy* L;
    WeatherDerivative() = delete;
    WeatherDerivative(Sinusoid s, Sinusoid K, Sinusoid Sig, Levy* l) : S{ s
     ↪  }, Kappa{ K }, Sigma{ Sig }  {
        L = l;
    };
    void SetTrend(Sinusoid s) {
        S = s;
    }
    void SetMeanReversion(Sinusoid K) {
        Kappa = K;
    }
    void SetVolatility(Sinusoid S) {
        Sigma = S;
    }
```

```
    void SetLevy(Levy* d) {
        L = d;
    }
    int Pricing(int t, int Maturity, double InitialDiff, int nStep, int nSim) {
        time_t start, end;
        time(&start);
        if (dynamic_cast<VG*>(this->L)) {
            std::cout << "Value of the indices under the Variance Gamma model: \n" << std::endl;
        }
        else if (dynamic_cast<NIG*>(this->L)) {
            std::cout << "Value of the indices under the Normal Inverse Gaussian model: \n" << std::endl;
        }
        SetThreadExecutionState(ES_CONTINUOUS | ES_SYSTEM_REQUIRED);
        std::cout << std::fixed;
        double Delta_t = 1.0 / nStep;
        const int N = Maturity * nStep;
        double* vS = new double[N + 1]();
        double* vKappa = new double[N + 1]();
        double* vSigma = new double[N + 1]();
        //auto vS = std::make_unique<double[]>(N + 1);
        L->SetDelta_t(Delta_t);
        for (int i = 0; i < N + 1; ++i) {
            vS[i] = S.Value(t + i * Delta_t);
            vKappa[i] = Kappa.Value(t + i * Delta_t);
            vSigma[i] = Sigma.Value(t + i * Delta_t);
        }
        double CAT = 0;
        double HDD = 0;
        double CDD = 0;
        double* Tilde_t = new double[N + 1]();
        double* T_t = new double[N + 1]();
        Tilde_t[0] = InitialDiff;

        for (int z = 0; z < nSim; ++z) {
            for (int u = 0; u < N; ++u) {
                Tilde_t[u + 1] = Tilde_t[u] - vKappa[u] * Tilde_t[u] * Delta_t + sqrt(vSigma[u]) * L->gen();
            }
            for (int u = 0; u < N + 1; ++u) {
                T_t[u] = Tilde_t[u] + vS[u];
            }
            for (int d = 0; d < Maturity; ++d) {
                int position = d * nStep;
                double DailyMax = *std::max_element(T_t + (int)(position + 1), T_t + (int)(position + nStep));
                double DailyMin = *std::min_element(T_t + (int)(position + 1), T_t + (int)(position + nStep));
                double DailyTemp = (DailyMin + DailyMax) / 2;
                CAT = CAT + DailyTemp;
                HDD = HDD + std::max(18.0 - DailyTemp, 0.0);
                CDD = CDD + std::max(DailyTemp - 18.0, 0.0);
            }
        }
```

```
        std::cout << "CAT index : " << CAT / nSim << std::endl;
        std::cout << "HDD index : " << HDD / nSim << std::endl;
        std::cout << "CDD index : " << CDD / nSim << "\n" << std::endl;
        time(&end);
        auto execution_time = double(end - start);
        execution_time = double(end - start);
        std::cout << "Execution time " << execution_time << " sec \n " << std::endl;
        SetThreadExecutionState(ES_CONTINUOUS);
        delete[] vS;
        delete[] vKappa;
        delete[] vSigma;
        delete[] Tilde_t;
        delete[] T_t;
        return 1;
    }

};

int main() {
    VG vg = VG{ 3.021098, 2.459503, -0.000062 };
    NIG nig = NIG{ 1.618664, -0.000062, 1.621384 };
    WeatherDerivative StockholmDev = WeatherDerivative{ Sinusoid{6.1785162890049445, 9.842711993555477e-05, 10.17684885383261, -1.9358915022293823, -0.7672813386695645, 29.623776422246006, 0, 0, 0, 0},
                                        Sinusoid{ 0.2352367125917137, 0, -0.029629918020637566, 0.06496097376065202, -0.036865297056646366, 0.26830186964048086,
                                         0.01633859377487671, 1.6017102936767609, -0.0007695873912343463, 0.24609306152333735},
                                        Sinusoid{6.137261912748916, 0, 2.4610598949596327, 1.2648519932150053, -1.6420730958940173, -1.1105401910897663,
                                        0.6901859440955523, 0.8159967383683336, -0.0859968526532975, -0.7976238695231654}, &vg };

    StockholmDev.Pricing(21567, 28, -0.877290606124236, 100, 30000);
    StockholmDev.SetLevy(&nig);
    StockholmDev.Pricing(21567, 28, -0.877290606124236, 100, 30000);
    return 1;
};
```

References

[1] Eugene Lukacs. *Stochastic convergence*. 2. ed. Probability and mathematical statistics 30. New York: Academic Press, 1975. 200 pp.

[2] Ole E. Barndorff-Nielsen. “Processes of normal inverse Gaussian type”. In: *Finance and Stochastics* 2.1 (1997), pp. 41–68.

[3] Peter Alaton, Boualem Djehiche, and David Stillberger. “On modelling and pricing weather derivatives”. In: *Applied Mathematical Finance* 9.1 (2002), pp. 1–20.

[4] Dorje C Brody, Joanna Syroka, and Mihail Zervos. “Dynamical pricing of weather derivatives”. In: *Quantitative Finance* 2.3 (2002), pp. 189–198.

[5] Wim Schoutens. *Lévy processes in finance: pricing financial derivatives*. Wiley, 2003.

[6] David Applebaum. *Lévy processes and stochastic calculus*. Cambridge, UK; New York: Cambridge University Press, 2004.

[7] Rama Cont and Peter Tankov. *Financial modelling with jump processes*. Chapman & Hall/CRC, 2004.

[8] Ernst Eberlein and Ernst August v. Hammerstein. “Generalized Hyperbolic and Inverse Gaussian Distributions: Limiting Cases and Approximation of Processes”. In: *Seminar on Stochastic Analysis, Random Fields and Applications IV* (2004), pp. 221–264.

[9] Eckhard Platen and Jason West. “A Fair Pricing Approach to Weather Derivatives”. In: *Asia-Pacific Financial Markets* (2004).

[10] Fred Espen Benth and Jūratė Šaltytė-Benth. “Stochastic Modelling of Temperature Variations with a View Towards Weather Derivatives”. In: *Applied Mathematical Finance* 12.1 (2005), pp. 53–85.

[11] Fima C. Klebaner. *Introduction to stochastic calculus with applications*. London: Imperial College Press, 2006. 416 pp.

[12] Fred ESPEN Benth and Jūratė šaltytė Benth. “The volatility of temperature and pricing of weather derivatives”. In: *Quantitative Finance* 7.5 (2007), pp. 553–561.

[13] Fred Espen Benth, Jūratė Šaltytė-Benth, and Steen Koekebakker. “Putting a Price on Temperature”. In: *Scandinavian Journal of Statistics* (2007).

[14] Eric Jondeau, Ser-Huang Poon, and Michael Rockinger. *Financial modeling under non-Gaussian distributions*. Springer Finance Textbook. London: Springer, 2007.

[15] Ying Chen, Wolfgang Härdle, and Seok-Oh Jeong. “Nonparametric Risk Management With Generalized Hyperbolic Distributions”. In: *Journal of the American Statistical Association* 103.483 (2008), pp. 910–923.

[16] Monique Jeanblanc, Marc Yor, and Marc Chesney. *Mathematical Methods for Financial Markets*. London: Springer London, 2009.

[17] Luis Valdivieso, Wim Schoutens, and Francis Tuerlinckx. “Maximum likelihood estimation in processes of Ornstein-Uhlenbeck type”. In: *Statistical Inference for Stochastic Processes* 12.1 (2009), pp. 1–19.

[18] Ken-iti Sato. *Lévy processes and infinitely divisible distributions*. Nachdr. Cambridge studies in advanced mathematics 68. Cambridge: Cambridge Univ. Press, 2010, p. 486.

[19] Luigi Ambrosio, Giuseppe Da Prato, and Andrea Mennucci. *Introduction to Measure Theory and Integration*. 2011.

[20] Fred Espen Benth and Jūratė Šaltytė Benth. "Weather Derivatives and Stochastic Modelling of Temperature". In: *International Journal of Stochastic Analysis* (2011), pp. 1–21.

[21] Andrea Pascucci. *PDE and Martingale Methods in Option Pricing*. Milano: Springer Milan, 2011.

[22] David J. Scott et al. "Moments of the generalized hyperbolic distribution". In: *Computational Statistics* 26.3 (2011), pp. 459–476.

[23] Antonis Alexandridis and Achilleas D. Zapranis. *Weather Derivatives: Modeling and Pricing Weather-Related Risk*. Springer New York, 2013.

[24] Anatoliy Swishchuk and Kaijie Cui. "Weather Derivatives with Applications to Canadian Data". In: *Journal of Mathematical Finance* 03.1 (2013), pp. 81–95.

[25] Andreas E. Kyprianou. *Fluctuations of Lévy Processes with Applications*. Universitext. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014.

[26] Olivier Le Courtois and Christian Walter. *Extreme financial risks and asset allocation*. Imperial College Press, 2014.

[27] Krzysztof Podgórski and Jonas Wallin. "Convolution-invariant subclasses of generalized hyperbolic distributions". In: *Communications in Statistics - Theory and Methods* 45.1 (2016), pp. 98–103.

[28] Anthony W. Knapp. *Advanced Real Analysis: Digital Second Edition, Corrected version*. East Setauket, New York: Anthony W. Knapp, 2017.

[29] Ibrahim Abdelrazeq, B. Gail Ivanoff, and Rafal Kulik. "Goodness-of-fit tests for Lévy-driven Ornstein-Uhlenbeck processes". In: *Canadian Journal of Statistics* 46.2 (2018), pp. 355–376.

[30] Sheldon Axler. *Measure, Integration & Real Analysis*. Vol. 282. Graduate Texts in Mathematics. Cham: Springer International Publishing, 2020.

[31] Ole E Barndorff-Nielsen and Neil Shephard. *Basics of Lévy processes*.

[32] Peter J Brockwell, Richard A Davis, and Yu Yang. *Estimation for Non-negative Lévy-driven Ornstein-Uhlenbeck Processes*.

[33] Antonis Papapantoleon. *An introduction to Lévy processes with application in finance*.

[34] Karsten Prause. *Modelling Financial Data Using Generalized Hyperbolic Distributions*.

[35] Karsten Prause. *The Generalized Hyperbolic Model: Estimation, Financial Derivatives, and Risk Measures*.