NTNU
Norwegian University of
Science and Technology

Lévy Processes and Path Integral Methods with Applications in the Energy Markets

Christian A. J. Oshaug

Master of Science in Physics and Mathematics
Submission date: July 2011
Supervisor: Arvid Næss, MATH

Norwegian University of Science and Technology
Department of Mathematical Sciences

Lévy processes and Path Integral Methods with Applications in the Energy Markets

Christian Aleksander Jansen Oshaug

July 29, 2011

Problem description

Lévy processes may be well suited for modelling changes in energy prices. We will look into possible one-factor models driven by Lévy processes, and see how they can be calibrated to historical price series. We will use numerical Path Integrals to obtain price distributions and to valuate derivatives, based on energy price models.

Abstract

The objective of this thesis was to explore methods for valuation of derivatives in energy markets. One aim was to determine whether the Normal inverse Gaussian distributions would be better suited for modelling energy prices than normal distributions. Another aim was to develop working implementations of Path Integral methods for valuing derivatives, based on some one-factor model of the underlying spot price.

Energy prices are known to display properties like mean-reversion, periodicity, volatility clustering and extreme jumps. Periodicity and trend are modelled as a deterministic function of time, while mean-reversion effects are modelled with auto-regressive dynamics. It is established that the Normal inverse Gaussian distributions are superior to the normal distributions for modelling the residuals of an auto-regressive energy price model. Volatility clustering and spike behaviour are not reproduced with the models considered here.

After calibrating a model to fit real energy data, valuation of derivatives is achieved by propagating probability densities forward in time, applying the Path Integral methodology. It is shown how this can be implemented for European options and barrier options, under the assumptions of a deterministic mean function, mean-reversion dynamics and Normal inverse Gaussian distributed residuals.

The Path Integral methods developed compares favourably to Monte Carlo simulations in terms of execution time. The derivative values obtained by Path Integrals are sometimes outside of the Monte Carlo confidence intervals, and the relative error may thus be too large for practical applications. Improvements of the implementations, with a view to minimizing errors, can be subject to further research.

Contents

Acknowledgements

I want to thank my supervisor, Arvid Næss, for giving me advice along the way. Thanks to Sjur Westgaard for advice and for providing me with the data I needed. Thanks to Eskil K. Dahl for reading through the thesis and offering helpful comments.

Til Ada Oline og Linus Aleksander; beklager at vi mistet en halv sommer. Jeg skal gjøre det godt igjen.

Til Marit, for at du holder ut med meg.

1 Introduction

In this thesis we are concerned with the stochastic properties of energy price time series, and how to valuate derivatives based on energy spot price models.

In Section 2 we review some basic finance theory, and we look at Brownian motion and the model of Black and Scholes (1973). In Section 3 we look at Lévy processes, and the Normal inverse Gaussian process in particular, as an alternative to Brownian motion in financial modelling. Section 4 is devoted to the Path Integral methodology for valuation of derivatives, as an alternative to Monte Carlo methods, in cases where no analytical solutions are known.

In Section 5 we consider one-factor stochastic models for daily energy spot prices. We discuss how known properties of energy price data can be modelled, and we establish that Lévy processes are better suited than Brownian motion for driving the stochastics of the considered models.

In Section 6 we show how to implement Path Integral methods specifically for one of the models discussed in Section 5. We compare results from the Path Integral methods with results obtained by Monte Carlo simulations. The Path Integral methods compare favourably to Monte Carlo simulations in terms of execution time. All results are reasonable, suggesting that the implementations are working as intended, but some of the error can not be accounted for without further research.

2 Stochastic models for price changes in financial markets

2.1 Randomness of price processes

It's common to assume some version of the "Efficient Market Hypothesis" when modelling price changes in financial markets, i.e. the market responds instantly to new information about an asset, such that the price p_t of an asset "fully reflects" some set Φ_t of information that is available at time t (Fama, 1970). Assuming no arbitrage possibilities, an efficient market leads to random changes in the asset price (Samuelson, 1965). The resulting price process is a *Markov process*, which means that distributions of future states depends solely on the present state of the process.

Definition 1. The stochastic process $X = \{X_\tau, \tau \geq 0\}$ is said to have the *Markov property* if

$$Pr(X_t = x|\mathcal{F}_s) = Pr(X_t = x|x_s)$$

where $s < t$ and $\mathcal{F}_s$ represents the history of X up to time $\tau = s$.

Definition 2. The stochastic process $X = \{X_\tau, \tau \geq 0\}$ is a *martingale* if

(i) X_t is known at time t

(ii) $E[|X_t|] < \infty$ for all $t \geq 0$

(iii) $E[X_t|\mathcal{F}_s] = X_s \quad (0 \leq s \leq t)$

where $\mathcal{F}_s$ represents the history of X up to time $\tau = s$

A martingale is a process that is 'constant on average', and models an investment prospect with zero risk premium, a so-called 'fair game' (Schoutens, 2003, p. 14).

2.2 Brownian motion

Brownian motion is a Markov process that is commonly used for modelling in finance. We follow Schoutens (2003) in the following exposition.

Definition 3. A stochastic process $X = \{X_t, t \geq 0\}$ is a *standard Brownian motion* on some probability space $(\Omega, \mathcal{F}, P)$ if

(i) $X_0 = 0$ almost surely,

(ii) X has independent increments,

(iii) X has stationary increments,

(iv) $X_t - X_s \sim N(0, t-s), \quad dt > 0.$

The standard Brownian motion is often called the *Wiener process* after Norbert Wiener, and we will use the notation $W = \{W_t, t \geq 0\}$ for this process.

We see from condition (ii) that the standard Brownian motion is a Markov process. From the Markov property and the zero-mean distribution of time increments, we have

$$E[W_t|\mathcal{F}_s] = E[W_t|W_s] = W_s. \tag{1}$$

Thus, the standard Brownian motion is a martingale. It can be proved that the brownian motion has continuous paths. However, the paths are not differentiable at any point and the variation of the paths is infinite on any interval.

2.3 Geometric Brownian Motion and the Black-Scholes model

The *return* of an asset over a given time interval is defined to be the change in the asset price divided by the original value of that asset. Let S denote the value of an asset, then the return over a time interval dt can be written as $\frac{dS}{S}$. This quantity is of more interest to us than the absolute change in the asset price, and a common model for the return of an asset is given by the following stocastic differential equation (see Wilmott et al., 1995; Schoutens, 2003):

$$\frac{dS}{S} = \sigma dW + \mu dt \tag{2}$$

where $-\infty < \mu < \infty$ and $\sigma \geq 0$. In this model the term μ is called the *drift* parameter. It is a measure of the average rate of growth in the asset price and it contributes to the return in a deterministic way, like the interest rate (spot rate) on an investment in a risk-free bank. The term dW is the stochastic element of the model and represents a Wiener process, i.e. $dW \sim N(0, dt)$. The term σ is a scale parameter for the Wiener process. It is called the *volatility* and is a measure of the magnitude of variation in the asset price. The drift and volatility are often treated as constants, in which case there are maximum likelihood estimators available for calibration with historical data.

When $\sigma = 0$ the model is purely deterministic and (2) has an exact solution, yielding exponential growth in the value of the asset (Wilmott et al., 1995, p. 21):

$$S_t = S_0 \exp\left(\mu(t - t_0)\right). \tag{3}$$

When $\sigma > 0$, however, the unique solution of (2) becomes (Schoutens, 2003, p. 28):

$$S_t = S_0 \exp\left((\mu - \frac{1}{2}\sigma^2)t + \sigma W_t\right). \tag{4}$$

This exponential functional of Brownian motion is called *geometric Brownian motion*. We can write (4) as:

$$S_t = \exp\left(\log S_0 + (\mu - \frac{1}{2}\sigma^2)t + \sigma W_t\right). \tag{5}$$

Since W_t is normally distributed and the class of normally distributed variables are closed under linear transformation, the exponent has a normal distribution. Thus S_t has a *lognormal* distribution.

That assets prices follow a lognormal distribution is one of the key assumptions in the Black-Scholes model (Black and Scholes, 1973), which has been widely used for valuing European call and put options on the stock markets.

2.4 Empirical shortcomings of the Black-Scholes framework

The Black-Scholes model is attractive from a theoretical point of view, as it leads to analytical solutions for the price of European options, but empirical analysis reveal that the log-returns of real assets may differ significantly from the normal distribution. In particular, historical log-returns may have significant skewness and excess kurtosis.

Definition 4. The *skewness* of a distribution is denoted by γ_1 and is defined to be the third moment around the mean divided by the cube of the standard deviation:

$$\gamma_1 = \frac{\mu_3}{\sigma^3} = \frac{E[(X - \mu)^3]}{E[(X - \mu)^2]^{\frac{3}{2}}}$$

Skewness is a measure of the assymetry in the distribution. For symmetric distributions, such as the normal, the skewness is zero. Positive skewness indicates that the right tail of the distribution is longer than the left tail, and a negative skewness indicates the opposite. (Schoutens, 2003, p. 34)

Definition 5. The *excess kurtosis* of a distribution is denoted γ_2 and is given by

$$\gamma_2 = \frac{\mu_4}{\sigma^4} - 3 = \frac{E[(X-\mu)^4]}{E[(X-\mu)^2]^2} - 3$$

where μ_4 is the fourth moment around the mean.

We will refer to the excess kurtosis as simply the *kurtosis*. The correction of -3 in the definition is chosen in order to make the kurtosis of the normal distribution equal to zero. A distribution with zero kurtosis is called *mesokurtic*. Distributions with positive kurtosis are called *leptokurtic*, and distributions with negative kurtosis are called *platykurtic*. The kurtosis can be viewed as a measure of the 'pointyness' of the distribution or of the fatness of the tails. Leptokurtic distributions are more pointy and have fatter tails than the normal distribution. (Schoutens, 2003, p. 35)

Fama (1965) mentions several sources of research that show the inadequacy of the normality hypothesis and document leptokurtic behaviour in the stock market. Schoutens (2003, p. 34) provides skewness and kurtosis estimates for some major stock indices, which indicates leptokurtic behaviour and negative skewness. Similar behaviour may be observed in energy price series.

3 Lévy processes

The Lévy processes (named after Paul Lévy) is a class of stochastic processes, of which Brownian motion is a special case. We use the definition of Applebaum (2004).

Definition 6. A stochastic process $X = \{X(t), t \geq 0\}$ is a *Lévy process* if

(i) X has independent and stationary increments,

(ii) $X(0) = 0$ almost surely,

(iii) X is stochastically continuous (see Appendix A).

Alternatively (Schoutens, 2003, p. 44-45), the Lévy process may be defined in terms of infinite divisibility (see Appendix A). Let $\phi(u)$ be a characteristic function of any infinitely divisible distribution. A Lévy process is a stochastic process $X = \{X_t, t \geq 0\}$ which has $X_0 = 0$ and has independent, stationary increments, such that the distribution of $X_{t+s} - X_s$ has $(\phi(u))^t$ as it's characteristic function

It can be shown that the cumulant characteristic function $\psi(u) = \log \phi(u)$ of the Lévy increments satisfies the *Lévy-Khintchine formula*

$$\psi(u) = \mathrm{i}\gamma u - \frac{1}{2}\sigma^2 u^2 + \int_{-\infty}^{+\infty} (\exp(\mathrm{i}ux - 1 - \mathrm{i}ux 1_{\{|x|<1\}})\nu(\mathrm{d}x) \qquad (6)$$

where $\gamma \in \mathbb{R}$, $\sigma^2 \geq 0$ and ν is a Lévy measure (see Appendix A). The infinitely divisible distribution is thus determined by the *Lévy triplet* $[\gamma, \sigma, \nu(\mathrm{d}x)]$.

A Lévy process consists of a linear deterministic part, a Brownian part and a pure jump part, corresponding to each term in the Lévy triplet. The jumps are determined by the Lévy measure $\nu(\mathrm{d}x)$. If $\sigma = 0$, then the Lévy process has no Brownian part and is called a pure jump Lévy process.

By using Lévy processes in modelling financial time series, we are able to reproduce behaviours that can not be accounted for with Brownian motion. Examples of models based on Lévy processes found in the litterature include the Variance Gamma model (Madan and Seneta, 1990), the Hyperbolic model (Eberlein and Keller, 1995) and the the Normal inverse Gaussian (NIG) model (Barndorff-Nielsen, 1995), all of which are special cases of the Generalized Hyperbolic model (Eberlein and Prause, 1998).

The NIG market model has been used by Rydberg (1997), and more recently been by Næss et al. (2010), for analyzing stock market returns. We will restrict our attention to models based on the NIG distribution throughout this thesis.

3.1 NIG distributions

The class of Normal inverse Gaussian (NIG) distributions was introduced by Barndorff-Nielsen (1995). The NIG distribution with parameters α, β and δ, denoted NIG(α,β,δ), has a known density function (Schoutens, 2003, p. 60):

$$f_{\text{NIG}}(x;\alpha,\beta,\delta) = \frac{\alpha\delta}{\pi}\exp(\delta\sqrt{\alpha^2-\beta^2}+\beta x)\frac{K_1(\alpha\sqrt{\delta^2+x^2})}{\sqrt{\delta^2+x^2}} \tag{7}$$

where K_1 denotes the modified Bessel function of third order with index 1 (see e.g. Abramowitz and Stegun, 1968, a reference work with a chapter devoted to Bessel functions).

We can add a location parameter μ to this distribution such that if $X \sim NIG(\alpha,\beta,\delta)$, then

$$\overline{X} = X + \mu \sim NIG(\alpha,\beta,\delta,\mu) \tag{8}$$

with density function

$$f_{\overline{X}}(x) = f_X(x-\mu). \tag{9}$$

The α and β parameters determines the steepness and asymmetry of the distribution, while δ and μ are scale and location parameters (Rydberg, 1997). By using the four parameter NIG distribution, the variance, skewness and kurtosis can be calibrated independently of the mean when fitting the distribution to a data set.

In Figure 1 we compare a normal density function to a NIG density. Both distributions have zero mean and unit variance, but the NIG density has an additional $skewness = 1$ and $kurtosis = 2$. The skewness is clearly seen as the right tail of the NIG density is thicker than the left tail. The positive excess kurtosis is seen more clearly when we plot the log densities of the distributions, as in Figure 2. We see that the density in the tails decrease more rapidly for the normal distribution than for the NIG distribution.

3.2 The NIG process

The NIG process is a Lévy process defined by

$$X^{\text{NIG}} = \{X_t^{\text{NIG}}, t \geq 0\} \tag{10}$$

with increments $X_{t+s}^{\text{NIG}} - X_s^{\text{NIG}}$ distributed as $NIG(\alpha,\beta,\delta t)$ (see Rydberg (1997) or Schoutens (2003)). An additional drift term μ can be added, such that the increments are distributed according to a $NIG(\alpha,\beta,\delta t,\mu t)$ law.

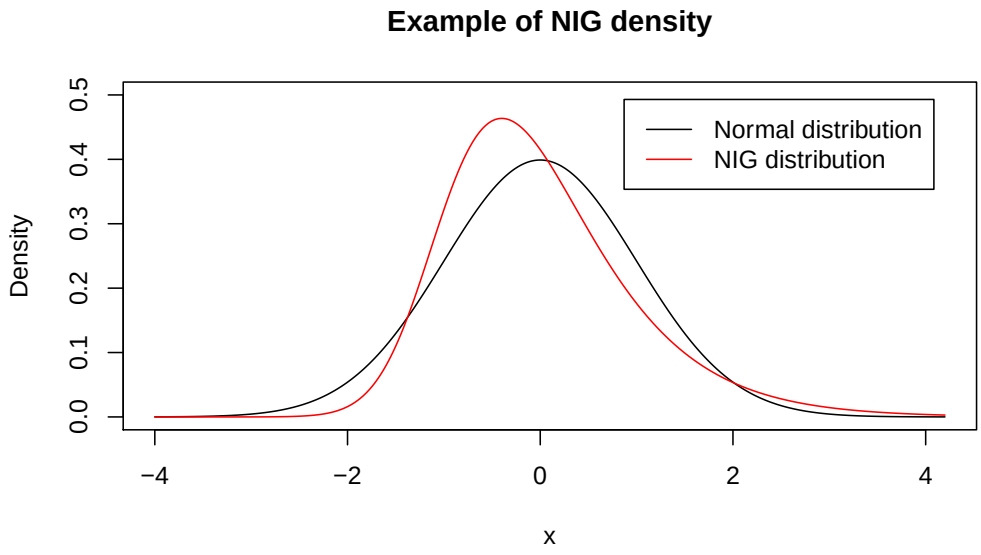

Figure 1: Example densities of normal and NIG distributions. Both distributions have $mean = 0$ and $variance = 1$, but the NIG distribution has $skew = 1$ and $kurtosis = 2$.

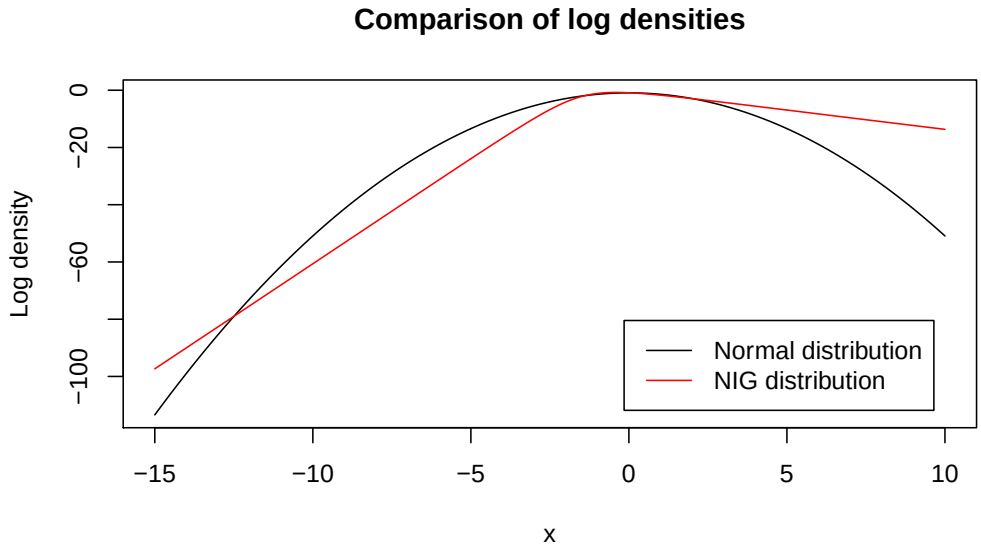

Figure 2: Log densities of normal and NIG distributions, where the first two moments are equal and the NIG distribution have positive skewness and kurtosis.

The location and scale of the increments are proportional to t, due to the convolution properties of the NIG distribution (Rydberg, 1997):

$$NIG(\alpha, \beta, \delta_1, \mu_1) * NIG(\alpha, \beta, \delta_2, \mu_2) = NIG(\alpha, \beta, \delta_1 + \delta_2, \mu_1 + \mu_2) \quad (11)$$

The $NIG(\alpha, \beta, \delta)$ distribution can be written as a mixture of a normal and an Inverse Gaussian (IG) variable (Barndorff-Nielsen, 1997). Assuming we can simulate from the normal and the IG distributions (see e.g. Rydberg, 1997), we can use this relation to simulate from the NIG distribution. Following Schoutens (2003) and adding the drift term μt, we can represent a drifted NIG process by

$$X_t = \beta\delta^2 I_t + \delta W_{I_t} + \mu t \quad (12)$$

where W_t is a standard Brownian motion. I_t is an $IG(\mu, \lambda)$ process with parameters $\mu = (\delta\sqrt{\alpha^2 - \beta^2})^{-1}$ and $\lambda = 1$, which has increments with the following probability distribution (see Næss et al., 2010):

$$f_{IG}(x; \mu, \lambda) = \left[\frac{\lambda}{2\pi x^3}\right]^{\frac{1}{2}} \exp\left(\frac{-\lambda(x-\mu)^2}{2\mu^2 x}\right). \quad (13)$$

Note that under this parameterization, which is different from that of Schoutens (2003), the IG distribution has the following scaling properties (see Appendix B):

$$X \sim IG(\mu, \lambda) \rightarrow tX \sim IG(t\mu, t\lambda). \quad (14)$$

The representation of the NIG process in Equation (12) can interpreted as a Brownian motion in "economic time", where the economic time follows an IG process in real time. A large jump in the IG process can then be interpreted as a period of high economic activity, which increases the probability of large price movements. In Figure 3 we show a NIG process obtained by means of an IG process and a Brownian motion. We see that large jumps in the IG process corresponds to large movements in the NIG process.

3.3 Estimating NIG parameters

Having a known probability density function, the four parameter NIG distribution can be fitted to a data set using Maximum Likelihood (ML) estimation. Numerical methods for doing ML estimation for the NIG distributions (see e.g. Karlis, 2002) will not be discussed here.

Alternatively, one can use the Method of Moments for parameter estimation. The first four moments of the $NIG(\alpha, \beta, \delta, \mu)$ distribution are known

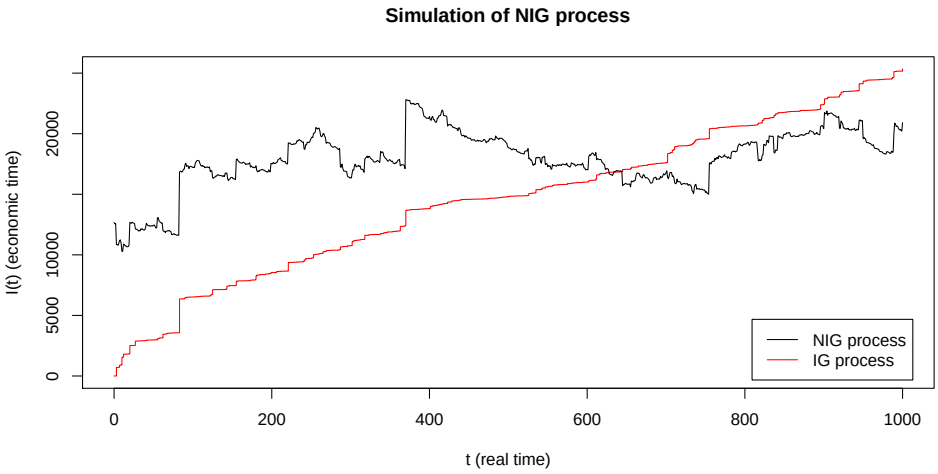

Figure 3: NIG process simulated by means of a IG process, with parameters $\alpha = 0.5$, $\beta = 0.15$, $\delta = 0.07$ and $\mu = -0.022$. The NIG process have been scaled and shifted in order to show it's characteristics in the plot. We clearly see that large jumps in the IG process corresponds to large jumps in the NIG process.

in terms of the four parameters, and these equations can be solved for the parameters (see Appendix B). Thus we have:

$$\alpha = \frac{\sqrt{3\gamma_2 - 4\gamma_1^2}}{\sqrt{m_2}(\gamma_2 - \frac{5}{3}\gamma_1^2)} \tag{15}$$

$$\beta = \frac{\gamma_1}{\sqrt{m_2}(\gamma_2 - \frac{5}{3}\gamma_1^2)} \tag{16}$$

$$\delta = \frac{\sqrt{m_2(3\gamma_2 - 5\gamma_1^2)}}{\gamma_2 - \frac{4}{3}\gamma_1^2} \tag{17}$$

$$\mu = m_1 - \frac{\gamma_1\sqrt{m_2}}{\gamma_2 - \frac{4}{3}\gamma_1^2} \tag{18}$$

where m_1 is the mean, m_2 the variance, γ_1 the skewness and γ_2 the excess kurtosis of the distribution. The parameters may be straightforwardly estimated from these equations by using sample moments.

Method of Moments estimation is, in general, less accurate than ML estimation and may be inadequate. In particular, it will fail whenever $\hat{\gamma}_2 < \frac{5}{3}\hat{\gamma}_1^2$. Method of Moments estimates are sometimes used as initial values for iterative ML algorithms. If the samples are large and the best possible estimates are not required, the Method of Moments may be preferred for it's simplicity,

and we will use it for implementations in this thesis.

4 The Path Integral approach to valuing derivatives

4.1 The Path Integral approach

A motivation for modelling asset prices is to estimate the value of derivatives, and the Path Integral (PI) method (for technical expositions, see Næss (2001) or Linetsky (1997)) can be used to calculate the value derivatives under a given model. It has been proposed as an alternative to Monte Carlo simulations, as MC techniques are too slow for some purposes.

Let $X = \{X_t, t \geq 0\}$ be a Markov process and let $0 = \tau_0 < \tau_1 < \cdots < \tau_{m-1} < \tau_m = T$, such that the density $p(x_{\tau_i}|x_{\tau_{i-1}})$ is known and we are able to sample from that density. We assume X_0 is known, and we want to know the density of X_T at some time $t = T$. A typical MC approach is to start at X_0 and simulate from the known density function to propagate the process forward in time. This way we obtain a sample of the stochastic process for $t = 0, \tau_1, \ldots, \tau_{m-1}, T$. If we repeat this procedure k times, then we obtain a size k sample of the process, from which we can estimate the probability density function at each time τ_i.

The Path Integral method involves propagating the whole density function itself forward in time, i.e. for each time step $\tau_i - \tau_{i-1}$ we approximate the distribution of X_{τ_i} given the distribution in $X_{\tau_{i-1}}$. By iterating over all the time steps $i = 0, \ldots, m$ we can approximate the distribution of X_T.

Assume we know the probability density function $p(x_{\tau_i}|x_{\tau_{i-1}})$ for all τ_i and the initial density $p(x_0)$. If the value of X_0 is known, then $p(x_0) = \delta(X_0)$ where δ denotes the Dirac delta function with $\int_{-\infty}^{\infty} \delta(x)\mathrm{d}x = 1$. The density $p(x_{\tau_i})$ can be found by the law of total probability

$$p(x_{\tau_i}) = \int_{-\infty}^{\infty} p(x_{\tau_i}|x_{\tau_{i-1}})p(x_{\tau_{i-1}})\mathrm{d}x_{\tau_{i-1}} \tag{19}$$

whenever $p(x_{\tau_{i-1}})$ is known. Iterating through $i = 0, \ldots, m$, we obtain

$$p(x_T) = p(x_{\tau_m}) = \int_{-\infty}^{\infty} \cdots \int_{-\infty}^{\infty} p(x_{\tau_m}|x_{\tau_{m-1}}) \ldots p(x_{\tau_1}|x_{\tau_0})p(x_{\tau_0})\mathrm{d}x_{\tau_0} \ldots \mathrm{d}x_{\tau_{m-1}}\,. \tag{20}$$

By numerically calculating the integral in Equation (20), we can obtain an approximation to the density of X_T, whenever X_t is a Markov process with known increments and initial distribution.

The PI approach described here enables us to approximate future densities of models different from that of Black and Scholes. For Brownian motion

and the NIG process the distribution of X_T can be derived analytically, due to the convolution properties of the increments. However, the PI method can be modified to solve valuation problems that are path dependent, and also to find the distribution of X_T in the case of a mean reverting model.

4.2 Valuation of path-independent derivatives

The most common derivatives in financial markets are European options, futures and forward contracts. The payoff functions of European options and forward contracts are dependent only on the price S_T of the underlying at the expiry $t = T$. There also exist a wide variety of more exotic derivatives, such as Asian, barrier and lookback options. These derivatives are path dependent, i.e. the payoff depends on the development of the underlying price process prior to expiry. We will do a short review of some derivatives of interest, mainly following Wilmott et al. (1995).

Note that we will only derive the values of the derivatives at expiry/maturity. We make no assumptions about risk-free interest rates, and accordingly the values are not discounted.

4.2.1 Forward contracts

A *forward contract* is an agreement between two parties to buy/sell an asset at some specified time in the future. The *underlying* asset is to be delivered at a predetermined price E, called the *forward price*, at the *maturity date* T. To the buyer, the value V of the forward contract at maturity is given by:

$$V(T, E) = S_T - E\,. \tag{21}$$

4.2.2 European options

A European call option is a contract that gives the holder a right to buy an asset, called the *underlying*, at a fixed price at some specified time T. We say that the option *expires* at time T, and the predetermined price E is called the *exercise price*. Since the contract gives the buyer a right, but not a duty, to exercise the option, the value C of a European call option at expiry is given by

$$C(T, E) = max(0, S_T - E)\,. \tag{22}$$

Similarly, the European put option gives the holder a right to sell the underlying asset, and the value P at expiry is given by

$$P(T, E) = max(0, E - S_T)\,. \tag{23}$$

If the asset price S_T at expiry is lower than the exercise price E, the European call option is worthless. The European put option is worthless if the opposite holds. Then we have the following relation between the European put and call options at expiry:

$$C(T,E) - P(T,E) = S_T - E\,. \tag{24}$$

This relation is called the *put-call parity*, and establishes that, for fixed T and E, buying a call and selling a put is equivalent to buying a forward contract.

The values of both European options and forward contracts are dependent only on the asset price at time T. If the probability density function $f_T(s)$ of S_T is known, the expected value of a European call option is given by

$$E[C(T,E)] = \int_E^\infty (s-E) f_T(s) ds\,. \tag{25}$$

The expected value of a European put option is given similarly by

$$E[P(T,E)] = \int_{-\infty}^E (E-s) f_T(s) ds\,. \tag{26}$$

The value of a forward contract $V(T,E)$ can be determined from the put-call parity and the expressions above, or from the integral

$$E[V(T,E)] = \int_{-\infty}^\infty (s-E) f_T(s) ds\,. \tag{27}$$

We note that having $-\infty$ as a lower limit of integration allows for, in principle, negative prices of the underlying. In many financial markets negative prices are not possible, but that is merely to say that $f(s) = 0$ for $s < 0$. In some power markets, however, negative prices have occurred in the past.

4.3 Exotic options

A barrier option is an option that either comes into existence or becomes worthless if the asset price S_t reaches some predetermined value (the *barrier*) before expiry. Asian options are options for which the payoff function depends on some form of average of the price process S_t. Lookback options have payoff functions that depend on the minimum or maximum asset price over the life time of the option. All these option types are path dependent, i.e. it is not enough to know the distribution of S_T in order to find their expected value.

A typical approach to valuing exotic options is to do repeated MC runs and calculate the average option value. The PI method may provide a fast

and accurate alternative to MC simulations for many valuation problems. Skaug and Naess (2005) have developed a PI approach for calculating the value of Asian options, which requires integration in two dimensions. PI methods for barrier options are treated in Skaug and Naess (2007) and Næss et al. (2010).

4.4 Barrier options

In the following we will look at how Path Integrals can be used to value barrier options in particular.

Barrier options can be either continuously monitored or discretely monitored. At each monitor time we check if the price has reached some predetermined barrier price B. The value of the option depends on whether the barrier is reached in the life time of the option.

There are two basic types of barrier options. *Knock-in* options are worthless until the barrier price is reached, while *knock-out* options are worthless from the moment the barrier is reached. The barrier may be above or below the initial price, and complex options may be constructed by combinations of upper and lower, knock-in and knock-out barriers.

4.4.1 Valuing knock-out barrier options by Path Integrals

Consider an *up-and-out* barrier option, i.e. a knock-out option with an upper barrier B. The underlying price process is discretely monitored at times τ_j, $j = 1, \ldots, m$, where $\tau_{j+1} > \tau_j > 0$ and $\tau_m = T$. Following Skaug and Naess (2007), we define the probability function:

$$H_m(s) = P\{s < S_{\tau_m} \leq B \cap S_{\tau_j} \leq B; j < m\}, \tag{28}$$

for $-\infty < s < B$. This is the probability that the price process S_t have not crossed the barrier at any monitoring up to and including $t = \tau_m$ and that it is larger than s at $t = \tau_m$. Now we can define the H-density

$$h_m(s) = -dH_m(s)/ds\,. \tag{29}$$

such that

$$H_m(s_1) - H_m(s_2) = \int_{s_1}^{s_2} h_m(s)ds \tag{30}$$

for $s_1 < s_2 < B$.

Assume that $h_j(s)$ and the conditional $p(s_{\tau_{j+1}}|s_{\tau_j})$ is known. The H-density can then be propagated forward in time by

$$h_{j+1}(s) = \int_{-\infty}^{B} p(s|\tilde{s})h_j(\tilde{s})d\tilde{s} \tag{31}$$

for $s < B$. If the initial distribution $p(s_{\tau_1})$ is known, then

$$h_1(s) = p(s_{\tau_1} = s) \tag{32}$$

for $s < B$. Thus, by Equations (31) and (32) we can write

$$h_m(s) = \int_{-\infty}^{B} \dots \int_{-\infty}^{B} p(s|s_{\tau_{m-1}}) \dots p(s_{\tau_2}|s_{\tau_1})\, p(s_{\tau_1})\, ds_{\tau_1}\, ds_{\tau_1} \dots ds_{\tau_{m-1}} \tag{33}$$

Once $h_m(s)$ is known, the value of a up-and-out call option $C(\tau_m, E, B)$ can be straightforwardly evaluated by

$$C(\tau_m, E, B) = \int_{E}^{B} (s - E) h_m(s) ds\,, \tag{34}$$

while the value up-and-out put option is given by

$$P(\tau_m, E, B) = \int_{-\infty}^{E} (E - s) h_m(s) ds\,. \tag{35}$$

For *down-and-out* barrier options, i.e. knock-out options with a lower we have to redefine $H_m(s)$ and change the limits of the integrals, but for the most part the procedure is the same.

4.4.2 Valuing knock-in barrier options by Path Integrals

We consider an *up-and-in* option, i.e. a knock-in barrier option with an upper barrier B, exercise price E and expiry T. As before, the option is discretely monitored at times τ_j, $j = 1, \dots, m$, with $T = \tau_m$.

The value of an up-and-in option can be derived from a parity of knock-in and knock-out options (see Appendix B.3):

$$C_{\mathrm{E}}(T, E) = C_{\mathrm{UAO}}(T, E, B) + C_{\mathrm{UAI}}(T, E, B)\,. \tag{36}$$

Thus, if the values of a European call option and a up-and-out barrier option are known (at fixed E and T), we can find the value of the corresponding up-and-in barrier option directly. The same result holds for put options.

If none of these values are known, we can use the Path Integral approach to find C_{UAI} and P_{UAI}. In the following we show how this can be done. The method described here is computationally as expensive as finding C_{E} and C_{UAO} separately, in fact we do compute the equivalent of a knock-out density and a full probability density. The following method can be modified and extended to evaluate more complex barrier structures.

We define two probability functions

$$H_m(s) = P\{S_{\tau_m} > s \cap [\cup_{j=1}^{m} S_{\tau_j} > B]\} \tag{37}$$

and

$$G_m(s) = P(S_{\tau_m} > s \cap [S_{\tau_j} < B; j \leq m]), \quad s < B, \tag{38}$$

and we have the relation

$$H_m(s) + G_m(s) = P\{S_{\tau_m} > s\}. \tag{39}$$

$H_m(s)$ is the probability that S_t have crossed the barrier at some monitoring $j \leq m$ and is greater than s at time τ_m. G_m is the probability that S_t have not yet crossed the barrier and is greater than s at time τ_m. From these probabilities, we can define the H-density and the G-density by

$$h_m(s) = -dH_m(s)/ds \tag{40}$$

and

$$g_m(s) = -dG_m(s)/ds, \quad s < B, \tag{41}$$

and we denote the full probability density function of S_{τ_j} by $p_j(s)$.

When propagating these functions forward in time, $g_{j+1}(s)$ can only receive contributions from $g_j(s)$, because only those price processes that have never crossed the barrier at τ_j can still be in the set of non-crossing processes at τ_{j+1}. Thus, we have

$$g_{j+1}(s) = \int_{-\infty}^{B} p(s|\tilde{s}) g_j(\tilde{s}) d\tilde{s}, . \tag{42}$$

The function $h_{j+1}(s)$, $s < B$, only receives contributions from $h_j(s)$, however for $s > B$ it receives contributions from the full probability density function $p_j(s)$. This is because whatever the value of S_{τ_j}, there is a positive probability of achieving $S_{\tau_{j+1}} > B$. But for a process to be in the set of barrier-crossing processes and to be smaller than B at $t = \tau_{j+1}$, it would have to already be in that set at $t = \tau_j$. From these considerations, we get

$$h_{j+1}(s) = \begin{cases} \int_{-\infty}^{\infty} p(s|\tilde{s}) h_j(\tilde{s}) d\tilde{s} & \text{if } s < B \\ \\ \int_{-\infty}^{\infty} p(s|\tilde{s}) p_j(\tilde{s}) d\tilde{s} & \text{if } s > B \end{cases}. \tag{43}$$

The expressions in Equations (42) and (43) can be evaluated numerically, and by iteration we can find $h_m(s)$. The value of an up-and-in call option can then be found by evaluating

$$C(\tau_m, E, B) = \int_{E}^{\infty} (s - E) h_m(s) ds \tag{44}$$

and the value of the corresponding put option is given by

$$P(\tau_m, E, B) = \int_{-\infty}^{E} (E - s) h_m(s) ds \,. \tag{45}$$

5 Modelling energy prices

5.1 Energy data

In this part we will analyse three historical energy related time series. We will look at spot prices for Brent Crude oil, a natural gas price index from ICE and the Nord Pool system spot price for electricity in Norway. All price series where obtained from the Reuter Ecowin reporting tool.

The oil price data go back to 1985 and are given in USD per barrel. The ICE natural gas index data go back to 1998, and they are given in GBP per Therm, where 1 Therm is the equivalent of 100 cubic feet. The electricity prices go back to 1996 and are given in NOK per MWh. In Figure 4 we display the log values of these time series. The prices are monitored at a daily basis, with weekends removed. This leaves approximately 260 trading days each year. The oil, gas and electricity data contains 177, 215 and 139 missing values respectively. We replace missing values by the mean value of the adjacent non-missing data.

Energy commodities are known to display interesting price dynamics. Such commodities have limited storeability, which makes it difficult to compensate for shocks in supply or demand. Price shocks are frequently observed in gas and electricity markets. Other typical characteristics of electricity prices include seasonal and weekly periodicity, mean reversion and volatility clustering (see e.g. Byström (2005), Koopman et al. (2007) and Weron et al. (2004)). Log returns on electricity prices are not normally distributed and display significant skewness and excess kurtosis (see e.g. Cartea and Figueroa (2005) and Lucia and Schwartz (2002)). Some of these characteristics may be idiosyncratic to electricity, while others may apply to oil and gas prices as well.

In modelling these energy spot prices, we will focus on features like periodicity, mean reversion and the distribution of driving processes. We will assume the volatility to be constant throughout this thesis.

5.2 A one factor model for energy spot prices

In this section we will look at a one factor model, similar to those used in Lucia and Schwartz (2002) and Benth et al. (2008). In modelling these energy price series we will assume that they share the same basic dynamics. We assume that general price growth and seasonal variations can be modelled in a deterministic way. We will use a single factor geometric model for the price series:

$$d \ln S(t) = d \ln \Lambda(t) + dX(t) \tag{46}$$

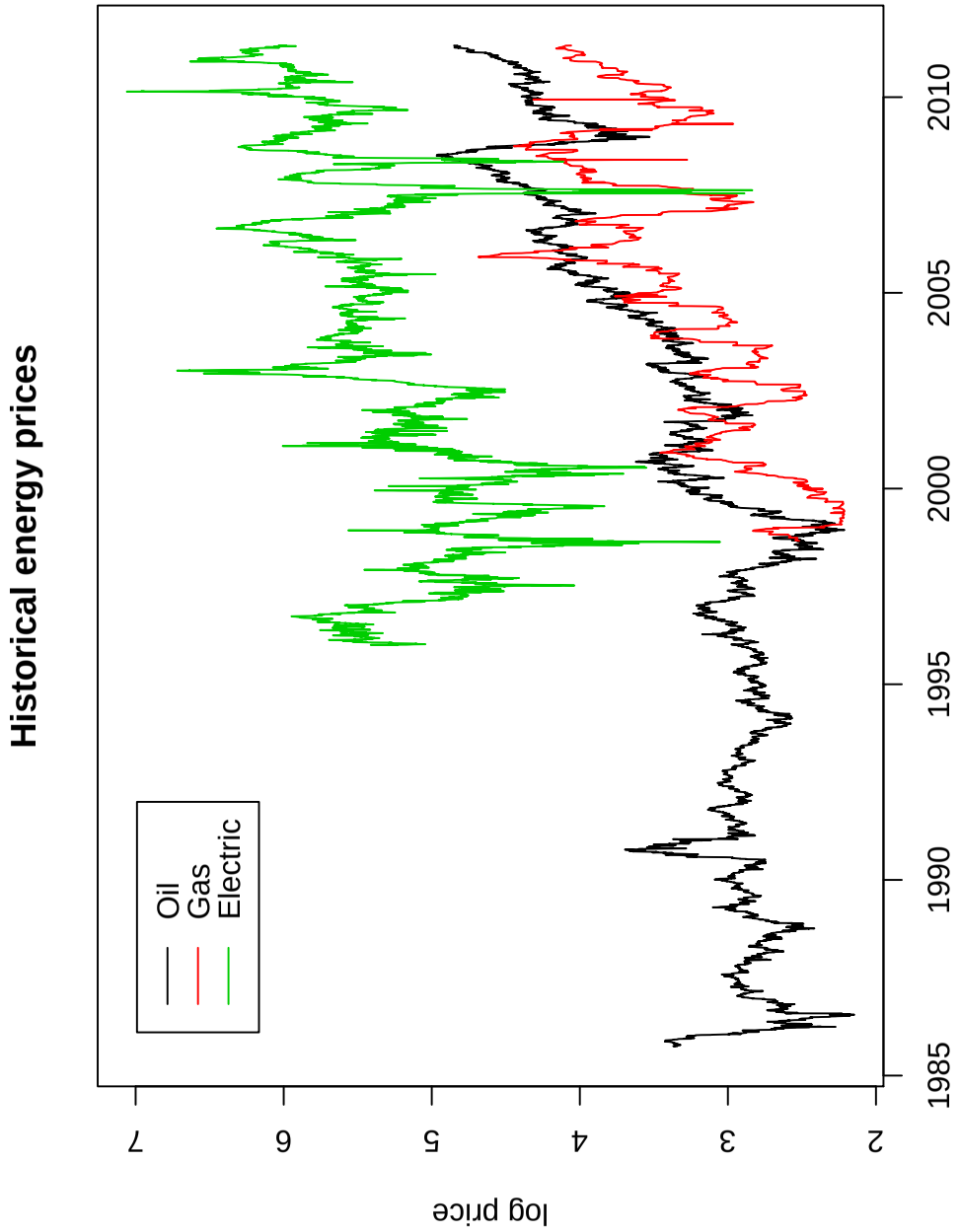

Figure 4: Historical log spot prices of Brent Crude oil, natural gas and electric power.

where $\Lambda(t)$ is a deterministic function modelling trend and seasonal variations. $X(t)$ is assumed to be a mean reverting process satisfying

$$dX(t) = -\alpha X(t)\,dt + dL(t). \tag{47}$$

Here α is a mean reversion parameter and $L(t)$ is the Lévy process driving the stochastic dynamics of the model. The discrete X_t then becomes an autoregressive process of order 1, an AR(1) process (see e.g. Shumway and Stoffer, 2000), which can be written

$$X_t = \varphi X_{t-1} + \varepsilon_t\,. \tag{48}$$

Thus ε_t is a Lévy increment, and the mean reversion parameter is $\alpha = 1 - \varphi$.

5.2.1 Modelling seasonal variations

In order to analyze the stochastic properties of $X(t)$ we need to have a model for $\Lambda(t)$. We assume that there is some trend in the log price because of general price growth. It is also known that energy prices tend to display seasonal variations as well as weekly variations. Thus a resonable model for $\Lambda(t)$ can be

$$\ln\Lambda(t) = \beta_0 + \beta_1 \cos\left(\frac{\tau_1 + 2\pi t}{260}\right) + \beta_2 \cos\left(\frac{\tau_2 + 2\pi t}{5}\right) + \beta_3 t\,. \tag{49}$$

The typical way to fit a function to a discrete series is to use a Least Squares (LS) approach. A possible drawback of using LS here is that extreme values will be more influential than the normal valued data points. We want $\ln\Lambda(t)$ to model only trend and periodic variations, not the random variations of the stochastic process.

Benth et al. (2008, chap. 5.1.1) propose to remove "outliers" in the data before fitting the model with LS. They look at the daily changes in the logaritmically transformed spot prices. Sorting these data we find the lower and upper quartiles Q_1 and Q_3, then we define the interquartile range (IQR) to be the difference $Q_3 - Q_1$. An outlier is then defined to be any value that falls outside the interval $[Q_1 - 3 \times IQR, Q_3 + 3 \times IQR]$. The number of outliers found are shown in Table 1. For each outlying jump in the log price, we replace the resulting value with the mean of the adjacent log prices (treating it as we would a missing value). Then an ordinary LS procedure[1] is employed in order to fit $\ln\Lambda(t)$ to the resulting time series. The estimated parameters are shown in Table 2.

[1]We used the *nlm* function in R, with analytical gradient input. The results were obtained in a few seconds.

	# of outliers detected
Oil	69
Gas	252
Electric	149

Table 1: Number of outliers found in the daily differences of log prices.

	β_0	β_1	β_2	β_3	τ_1	τ_2
Oil	2.4247	0.0352	0	0.0003	0	0
Gas	2.5350	-0.0331	0	0.0005	0	0
Electricity	4.7697	0.2026	-0.0243	0.0003	-0.0024	0.0001

Table 2: Trend and seasonal parameters from ordinary LS with outliers removed.

Another way to handle extreme variations in the data is to use robust least squares estimation (see e.g. Klüppelberg et al., 2010). This algorithm works by iteratively fitting the model and manipulating influential data points. In each iteration, after fitting the model, we estimate the standard deviation s of the sample. Then for each residual r_i, if $|r_i| > \delta s$ we move the data point δs closer to the fitted function. The value of δ is a matter of choice. We stop the iterations when the change in the fitted function is below some tolerance level of our choice.

Let $\hat{g}_k(t)$ be the fitted seasonality function at the k-th iteration. We choose a tolerance level of 0.1, i.e. we stop the iterations whenever $\sum_{j=1}^{N} |\hat{g}_k(j) - \hat{g}_{k-1}(j)| < 0.1$, and we set $\delta = 1.5$. The estimated parameters are shown in Table 3.

	β_0	β_1	β_2	β_3	τ_1	τ_2
Oil	2.4373	0.0142	0	0.0003	0	0
Gas	2.5339	0.0307	0	0.0005	0	0
Electricity	4.6955	0.1400	-0.0255	0.0003	-0.0024	0.0001

Table 3: Robust least squares estimates of trend and seasonal parameters.

In Figures 5, 6 and 7 we have plotted the fitted trend and seasonality functions over the logaritmic price series. For the oil and electricity data there is clearly a difference in the seasonal amplitudes, depending on the methods of estimation. For the gas data there is almost no difference between the two methods. The electricity data seems to have significant weekly variations,

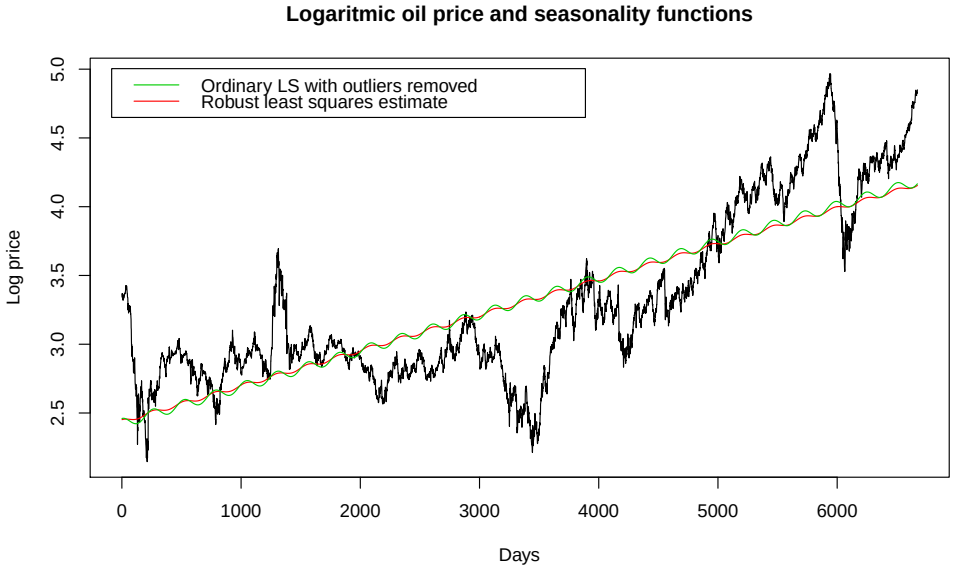

Figure 5: Logaritmic oil prices with seasonality and trend functions.

as we can see from the "thickness" of the plotted functions. We also note that the electricity data seems to be more affected by seasonal variations. The trend parameter β_3 is in the same order of magnitude for all price series, which seems reasonable as it models general price growth in the markets.

Which one of the methods to use is a matter of deliberate choice. The robust least squares approach admits less influence to extreme values, and the seasonal amplitudes thus tends to be smaller than for the other method. However, an advantage of just removing outlying log price jumps is that this method admits more influence to those extreme values that didn't arise from just a few extreme jumps. Thus periods of persistently high/low prices are given more weight. This is perhaps a reasonable compromise, and following Benth et al. (2008) we will use the parameters from ordinary LS with outliers removed.

5.2.2 Modelling detrended and deseasonalized data.

After obtaining the parameters for trend and seasonal variations, we can subtract $\ln \Lambda(t)$ from $\ln S$ in order to obtain a detrended and deseasonalized time series.

$$dX(t) = d \ln S(t) - d \ln \Lambda(t) \tag{50}$$

The resulting $X(t)$ series are displayed in Figure 8. All three time series have zero means, but they do not vary irregulary around zero. This leads us

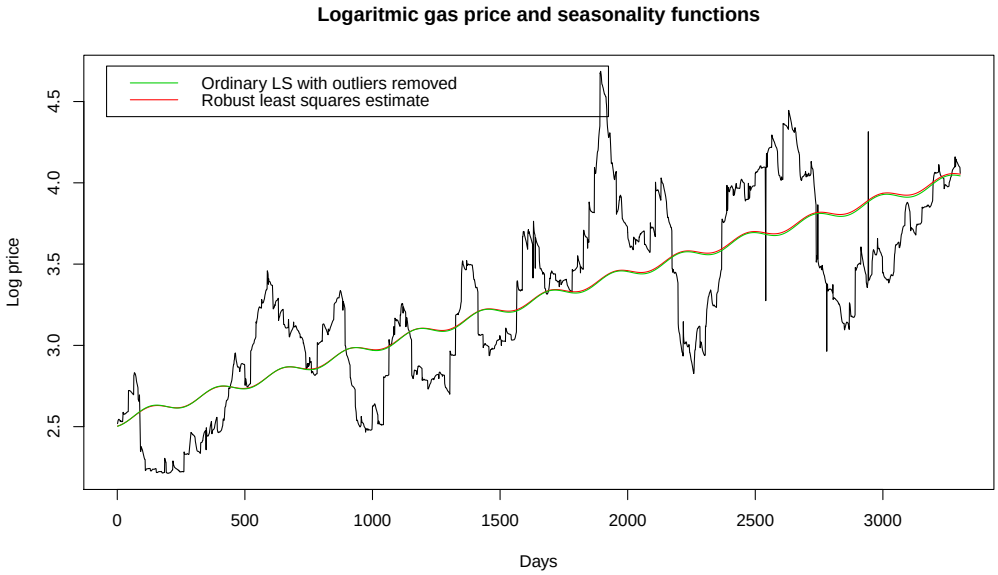

Figure 6: Logaritmic gas prices with seasonality and trend functions.

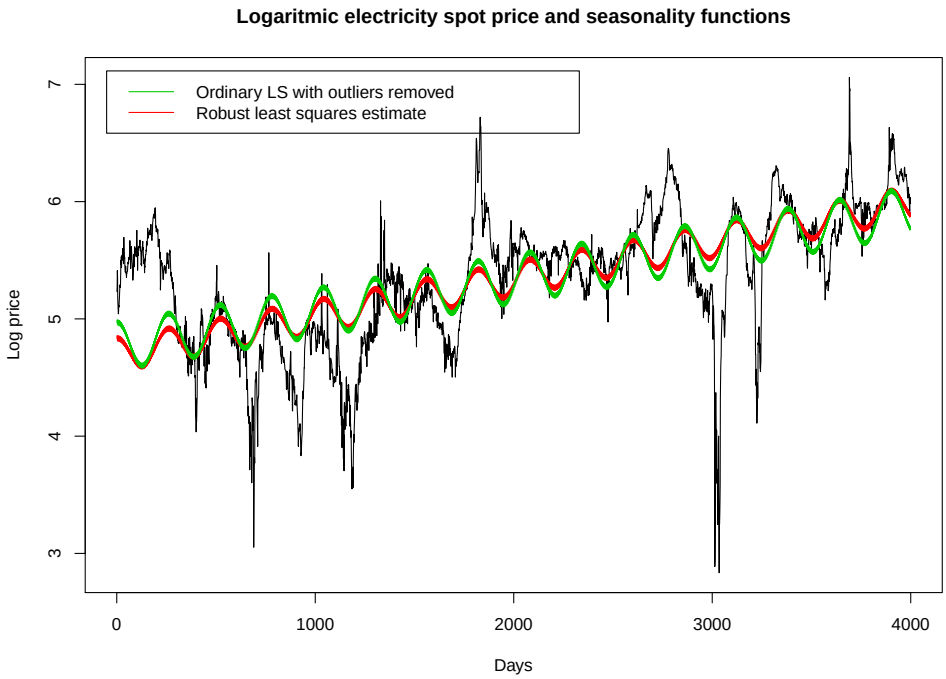

Figure 7: Logaritmic electricity prices with seasonality and trend functions.

to expect strong autocorrelation effects in the data, which is verified by the empirical autocorrelation functions displayed in Figure 9. When we fit an AR(1) model to the data, the autoregressive parameter φ is close to 1 for all three series. This suggests that there is only a weak reversion effect in the data.

We may assume $\alpha = 0$ in Equation (47) and look at the $dX(t)$ processes, i.e. the changes in the log price processes after trends and seasonal variations are removed. We approximate the sample density of these changes with a kernel smoothing function[2]. We also estimate gaussian parameters and NIG-parameters for these series. The results are compared in Figure 10. The sample densities show signs of leptokurtic behaviour, which can not be modelled with the gaussian distribution.

Normal Q-Q plots of these series (see Figure 11) confirm that the data are not gaussian. However, it is evident from Figure 12 that the NIG distribution fit the data quite well. When we fit an AR(1) model to each series and take out the (small) mean reversion effect, we get the same results.

We should note that the assumption of NIG distributed residuals is quite ad hoc. We have no argument to support such an assumption, except that the NIG distributions have favourable modelling properties in this case.

5.2.3 Simulation

Once we have obtained estimates of the $\Lambda(t)$, AR and NIG parameters, it is easy to simulate from this model. We have

$$dL_t \sim NIG(\alpha, \beta, \delta, \mu) \tag{51}$$

and in each time step we obtain dX_t from Equation (47), with reversion parameter $\alpha = 1 - \varphi$. Then $\ln S_t$ is obtained by adding the seasonality function to X_t.

We simulate 4000 days from this model, using the Nord Pool data, and the result is shown together with historical data in Figure 13. The simulated data clearly have a trend and seasonal variations. Compared to the driving NIG process, we also see the effect of mean reversion.

From visual inspection the simulated data seem reasonable, but there are too few extreme jumps and too many medium sized jumps. We also note that we don't get price spikes with our model. This behaviour may be a result of our assumption of constant volatility and constant mean reversion. Thus the model underestimates the probability of extreme jumps, and overestimate the volatility in between extreme events. Neither can our model reproduce the fast mean reversion often seen after extreme jumps.

[2]We applied the *density()* function in R with a gaussian kernel smoother.

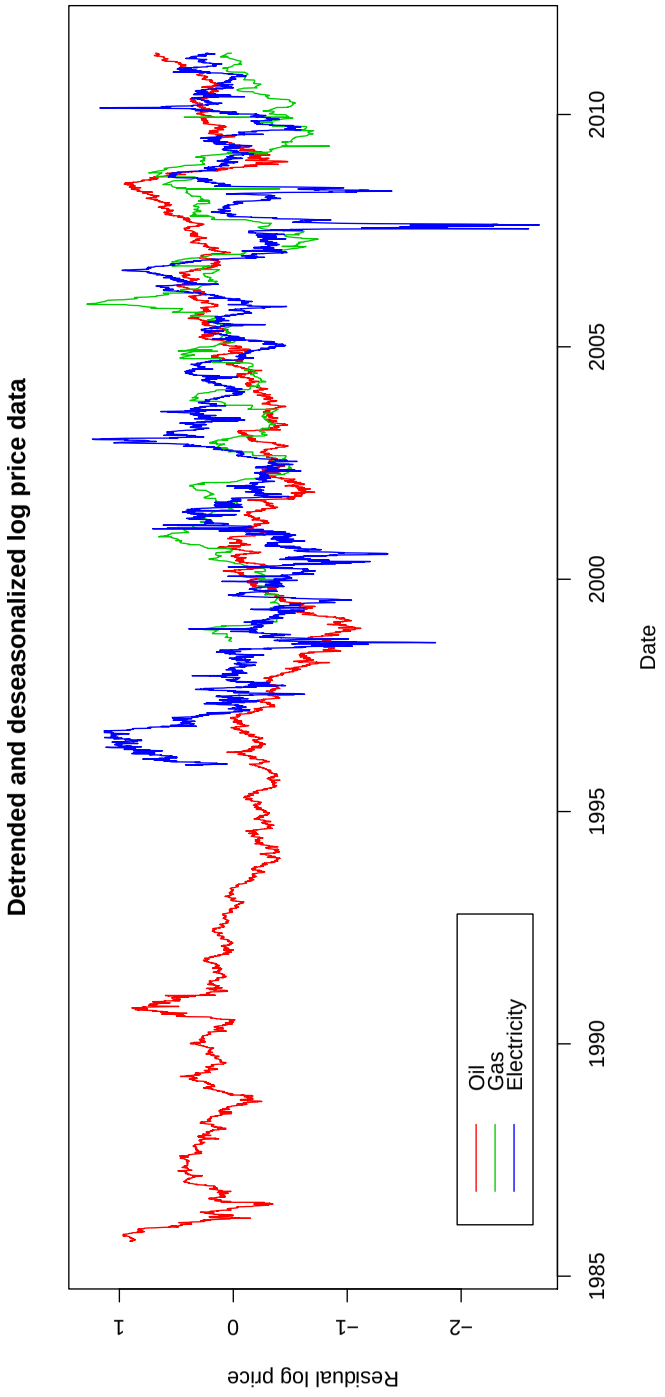

Figure 8: Detrended and deseasonalized log prices for oil, gas and electricity.

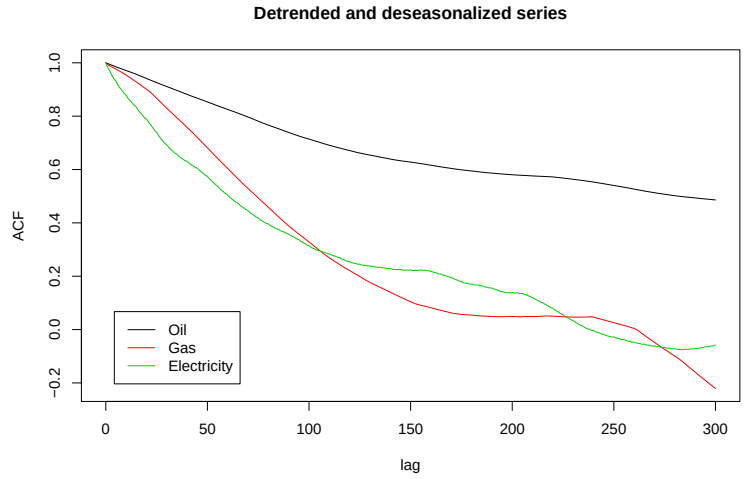

Figure 9: Empirical autocorrelation functions for detrended and deseasonalized time series. The ACF's are computed by the *acf*-function in R.

5.3 Modelling spot prices with periodic autoregressive models.

Another approach to seasonality modelling is to discard $\Lambda(t)$ and use a periodic autoregressive model directly. Byström (2005) uses an AR-GARCH model for modelling hourly spot prices at Nord Pool. In his model the hourly arithmetic return (or yield), r, is modelled as

$$r_t = \varphi_0 + \varphi_1 r_{t-1} + \varphi_2 r_{t-24} + \varphi_3 r_{t-168} + \varepsilon_t \,. \tag{52}$$

Here $\varepsilon_t = \sigma_t \eta_t$, and η_t is assumed to be i.i.d. variables from a gaussian or t-distribution with $mean = 0$ and $variance = 1$. The conditional variance σ_t^2 is modelled by

$$\sigma_t^2 = \beta_0 + \beta_1 \varepsilon_{t-1}^2 + \beta_2 \sigma_{t-1}^2 \tag{53}$$

in order to recreate the volatility clustering that are often observed in financial time series (see Figure 14). Modelling volatility clustering is outside the scope of this thesis, and we will consider a model where ε_t is assumed to be i.i.d. variables from a stable distribution.

5.3.1 A periodic autoregressive model for log returns

In Byström (2005) the arithmetic returns, $(S_t - S_{t-1})/S_{t-1}$, on electricity prices are modelled by Eq. (52) and (53) in order to study the behaviour of extreme changes in the upward direction. We are generally interested in

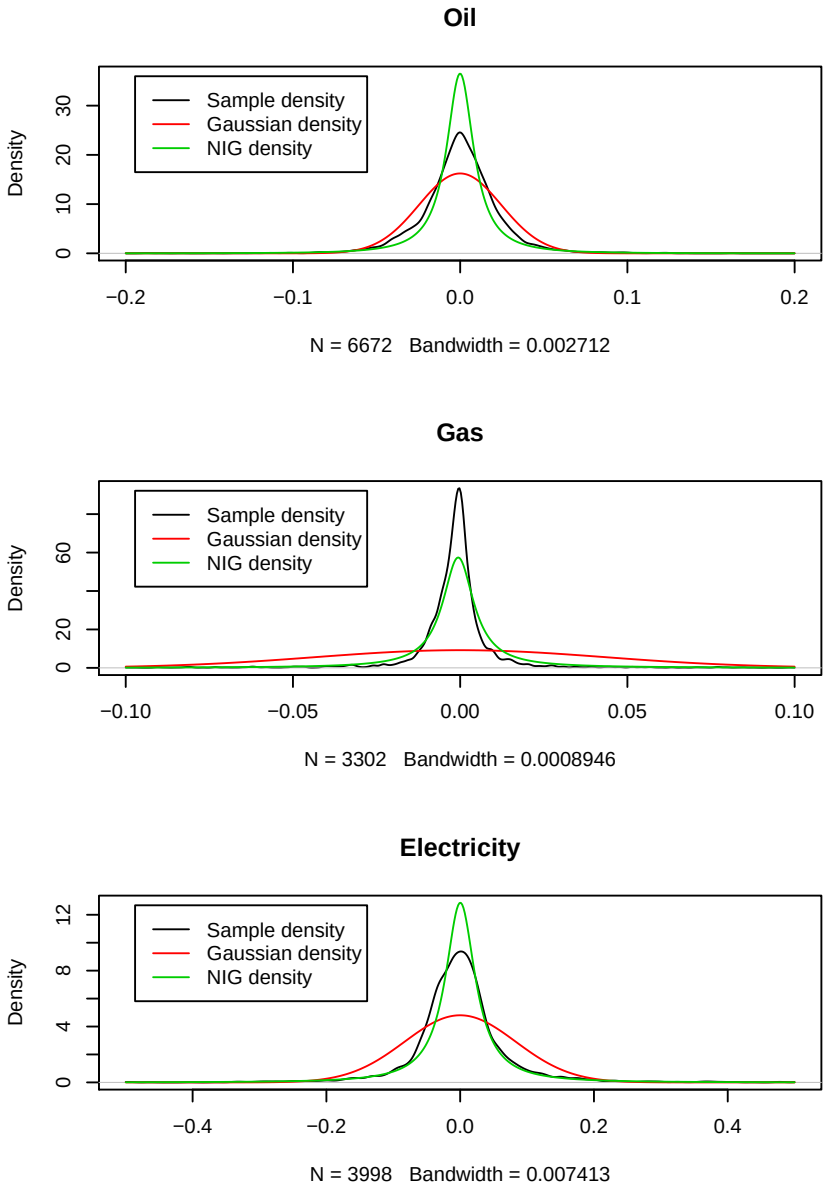

Figure 10: Sample densities and estimated gaussian and NIG densities for changes in detrended/deseasonalized log price series.

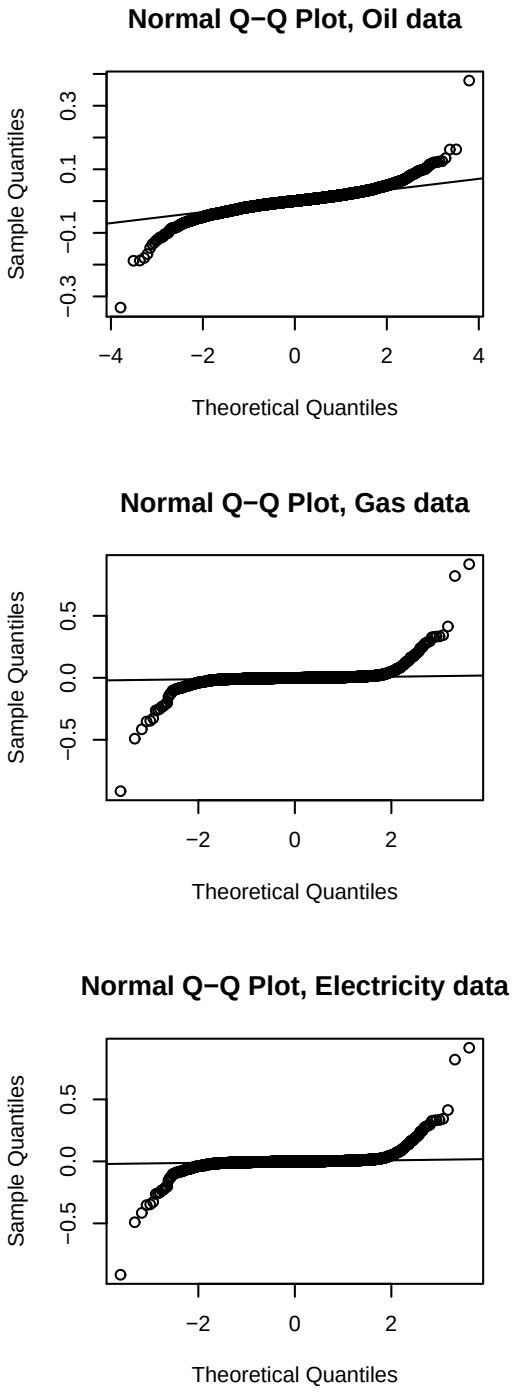

Figure 11: Normal Q-Q Plots of the changes in log prices after trend and seasonal effects are removed.

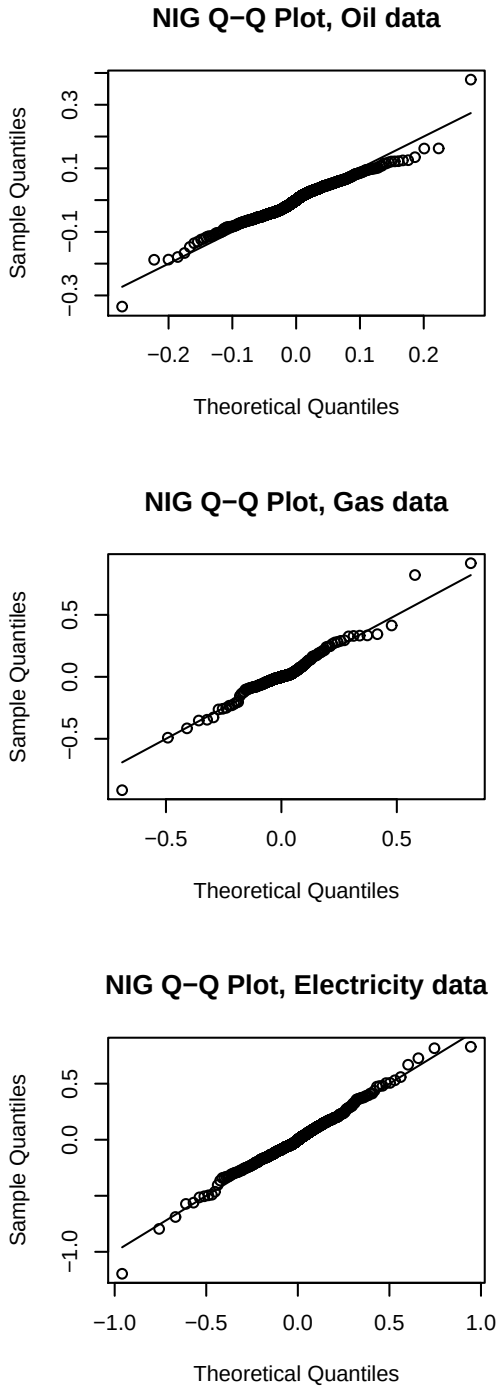

Figure 12: Q-Q plots of fitted NIG distributions against detrended/deseasonalized log prices.

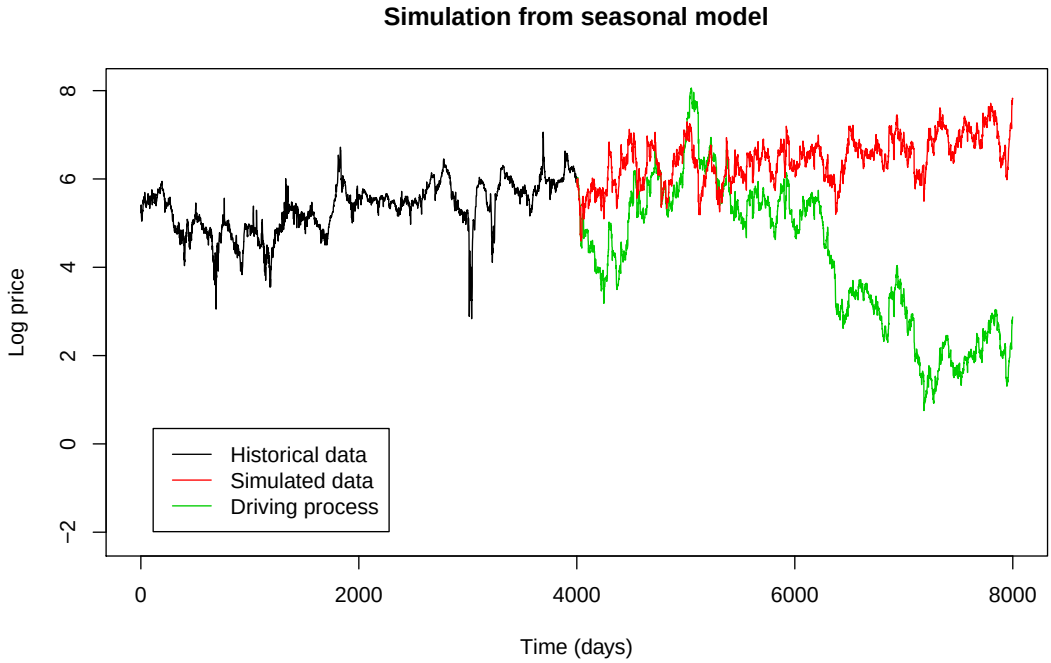

Figure 13: Historical data, simulated data and driving process for the simulations. Simulations are from one factor model with deterministic seasons and mean reversion, using the Nord Pool data.

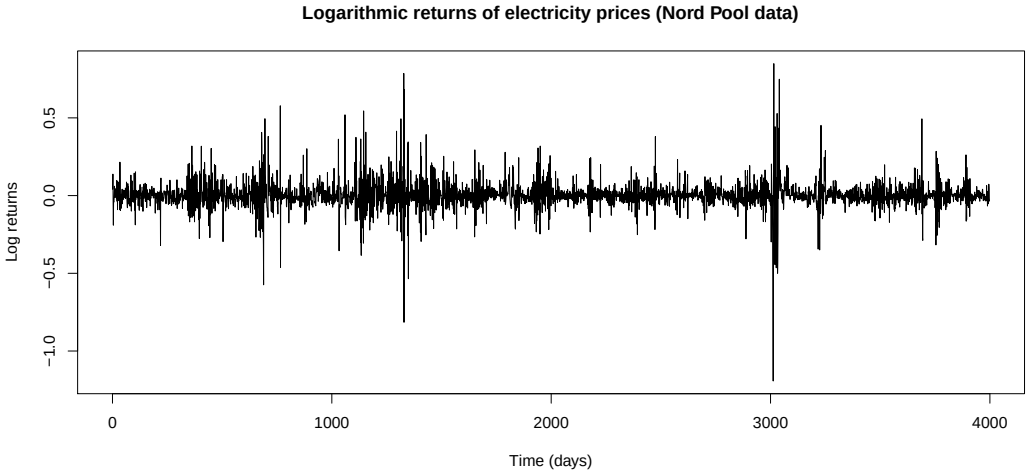

Figure 14: Logarithmic returns of electricity price series. We see periods of increased volatility, so called volatility clusters.

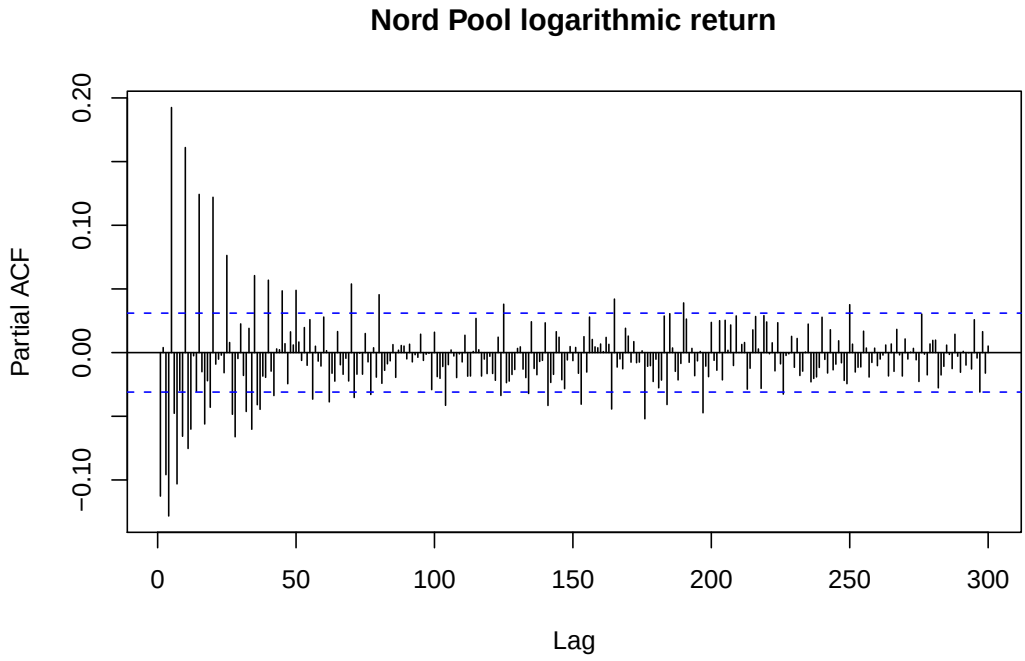

Figure 15: Partial autocorrelation structure for the logarithmic return on the Nord Pool system price.

forecasting both upward and downward movements, and we choose to look at the logarithmic returns $\log(S_t/S_{t-1})$. Under the logarithmic transformation the series of relative prices are balanced, i.e. the logarithmic returns sum to zero whenever $S_t = S_0$.

We anticipate that the log return of the energy prices will have autocorrelations at lags of one year and one week. The empirical partial autocorrelation of logarithmic return confirm that such effects are present. From the PACF plot of log returns in the Nord Pool data (see Figure 15) we see that there might be significant autocorrelations on many lags between 1 and 260 days. However, we should be suspicious of autocorrelations that we didn't anticipate or are unable to explain. Moreover, there is no specific reason why annual autocorrelations should be restricted to only one day, so a single positive correlation at $lag = 250$ should be interpreted with caution. With these considerations in mind, we might try fitting a periodic AR model with periods of 1, 5 and 250 days.

We apply the same procedure to the oil and gas series. For the log returns on oil prices, we are not able to find any AR parameters that are significantly different from zero. For the gas data we find a relatively strong

negative correlation on $lag = 1$ and some positive correlation at lags around 261 (i.e. about a year), but other correlation effects do not correspond to our anticipations and can not be accounted for. This means that under this model, the oil price log returns simply follow a Lévy process, while the gas price log returns follow a periodic AR model with periods 1 and 261 driven by a Lévy process. Thus the oil log return is taken to have no periodicity at all, while the gas log return is only dependent on the previous day and the previous year.

We proceed with the model for the Nord Pool system price. The model is given by:

$$r_t = \varphi_1 r_{t-1} + \varphi_2 r_{t-5} + \varphi_3 r_{t-250} + \varepsilon_t \,. \tag{54}$$

The gas price is modelled similarly. The LS estimates for the periodic autoregression parameters are shown in Table 4. We see that r_t is negatively correlated with r_{t-1}, while there are positive correlations at the week and year periods.

We note that the mean absolute value of the log returns are in the order of 10^{-2} for all three data sets. The mean effect of the AR terms in the model are thus in the order of 10^{-3}, i.e. an order of magnitude smaller than the standard deviation of the residual processes. Thus the residual process is prominent to the price development under the present model.

	φ_1	φ_2	φ_3	Residual SD
Electricity	-0.08725734	0.18678341	0.14764886	0.08381
Gas	-0.2336056		0.1212698	0.04360

Table 4: Least squares estimates of periodic AR parameters and standard deviation of the residual process, for log return models of gas and electricity prices.

We use Q-Q plots to investigate the distribution of the residual processes ε_t. From Figure 16 we see that these residuals are not from a gaussian distribution. NIG distribution seem to fit the residual processes quite well, as we can see from Figure 17 and 18. The NIG distribution also provides a reasonable fit to the log returns of each data set (see Figure 19). From visual inspection of the plots, however, the NIG distribution seems to be more appropriate for electricity data than for oil and gas data.

As expected, we obtain the same results if we omit the autoregressive terms and obtain Q-Q plots of the log returns directly.

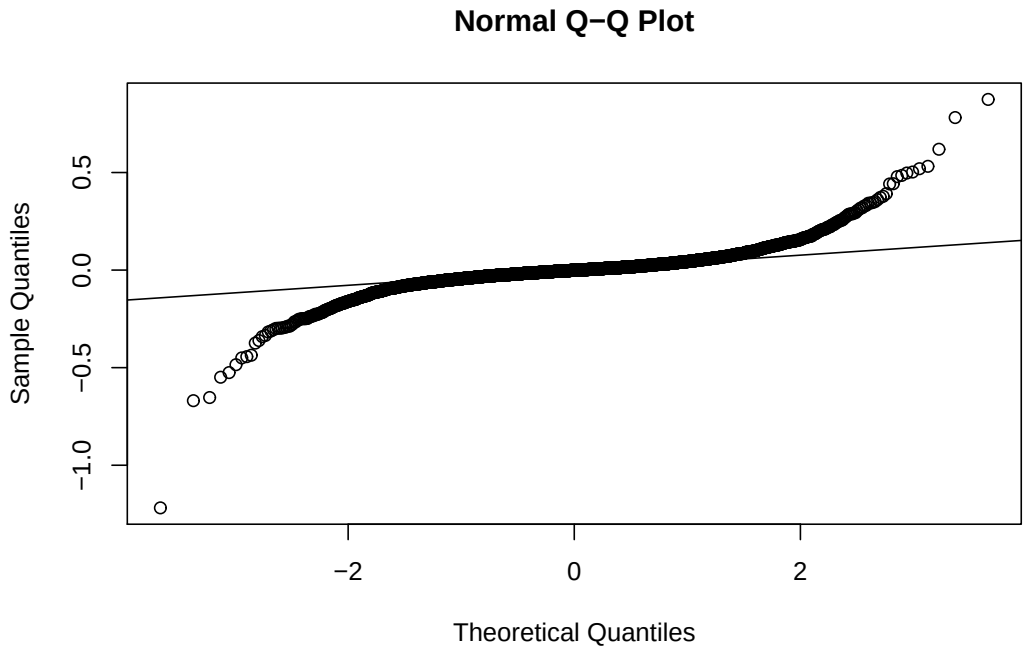

Figure 16: Normal Q-Q plot for the residual process ε_t in the periodic AR model (Eq. 54), using electricity price data.

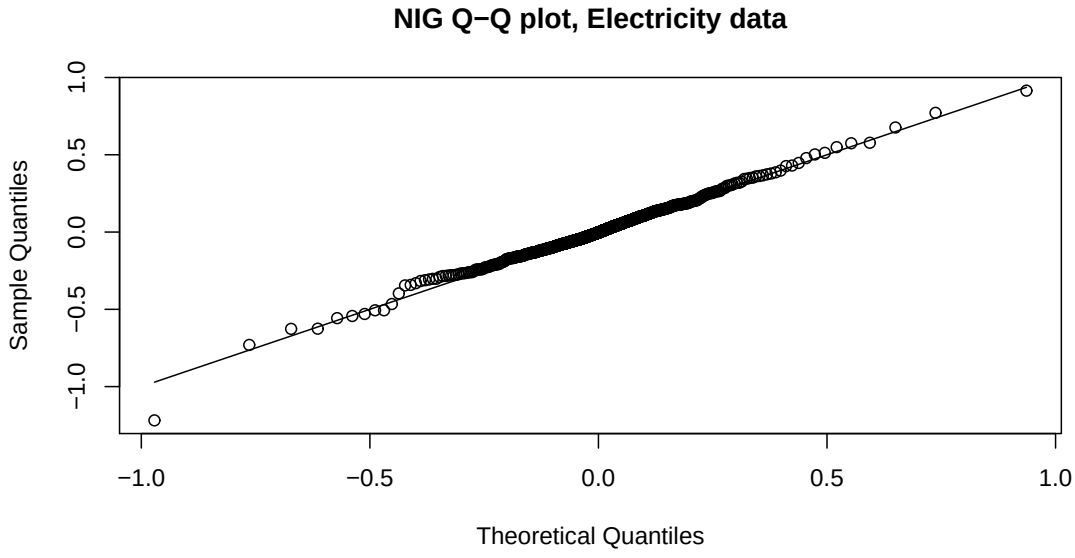

Figure 17: NIG Q-Q plot of the residual process ε_t in the periodic AR model (Eq. 54), using electricity data.

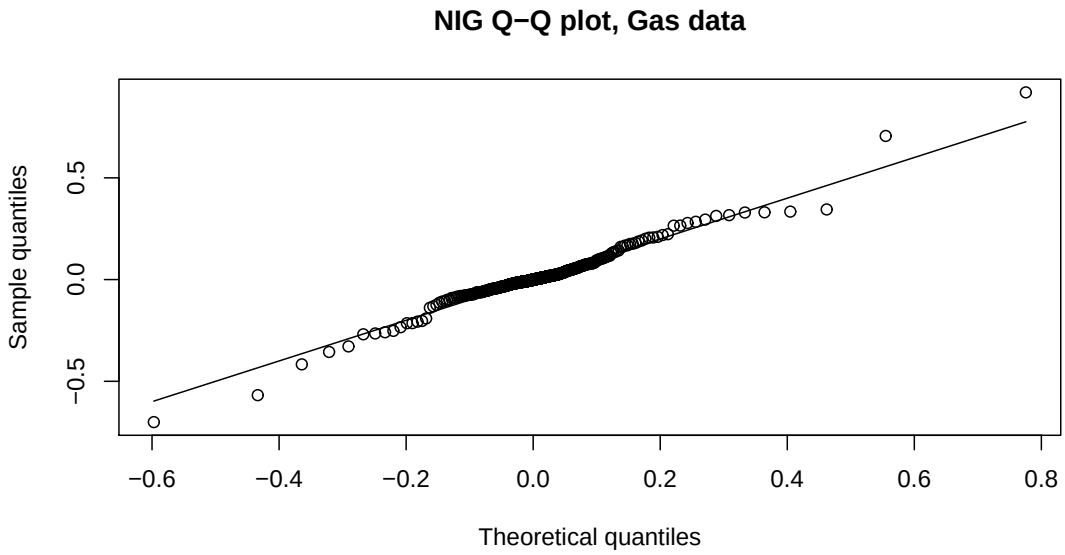

Figure 18: NIG Q-Q plot of the residual process ε_t in the periodic AR model, using gas data.

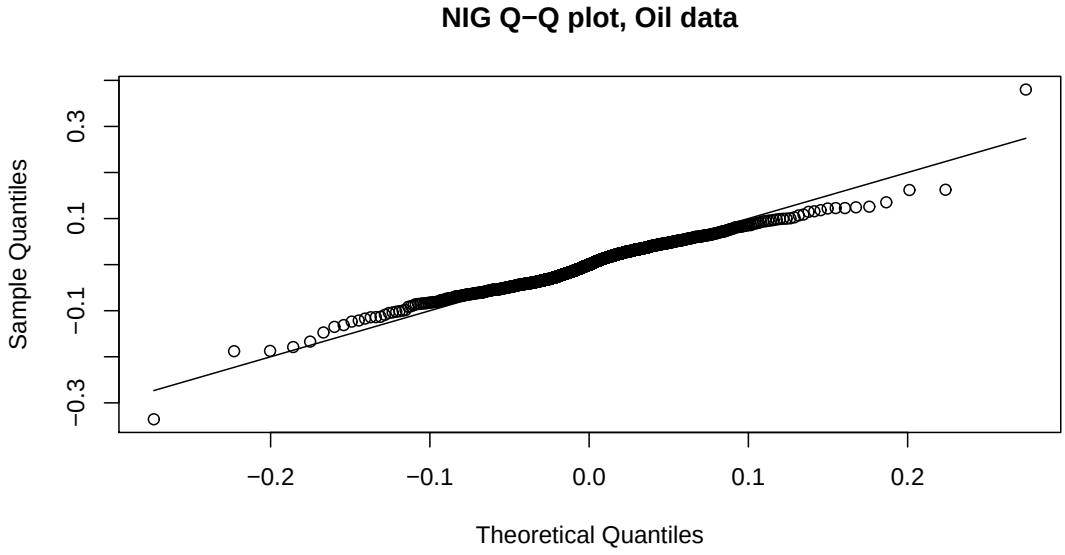

Figure 19: NIG Q-Q plot of the logreturns for oil data.

5.3.2 Simulation

After fitting a periodic AR model to a data set, we estimate the NIG parameters of the residual process ε_t. Knowing these parameters, we can simulate the residual process forward in time. The log return in each time step is then given by historical data and the residuals, using the relevant periodic AR model (e.g. Eq. (54) for electricity). After obtaining the log return r_t forward in time, the log price is given by the relation:

$$\ln S_t = \ln S_{t-1} + r_t \tag{55}$$

We simulate 2500 days forward from the periodic AR models, and the results are shown in Figures 20 and 21. From these plots we see that the periodic AR process differs little from the driving NIG process. Thus seasonal variations and trend are almost absent in the simulated data. Repeated simulations confirm that the model is likely to produce log prices that are uncharacteristic of energy spot prices. Neither are the model capable of producing price spikes with fast mean reversion or periods of extreme volatility.

It is not surprising that the fitted models produce uncharacteristic price developments in the long run. Since the models only depend on developments in the previous year, deviations from any expected outcome will propagate forward in time. There are no mechanisms for reversion to any historical developments beyond the previous year.

From Table 4 we see that the fitted models for electricity and gas are more influenced by short term developments than long term developments. As a result, even for simulations of less than 250 days ahead, the AR effects in the model will be more dependent on the driving process than on historical developments.

5.4 Discussion

We have looked at two different ways of modelling energy spot prices. In Section 5.2 we fitted a deterministic function for trend and seasonal effects of the log price, and used AR(1) dynamics for the mean reversion. In Section 5.3 we modelled the log returns as periodic AR models, attempting to capture seasonal effects in the AR model itself.

Both of the considered models have the drawback that by design they do not reproduce volatility clusters or price spikes (i.e. extreme jumps followed by fast mean reversion). In order to reproduce so-called regime changes, i.e. periods of increased volatility, we would have to use a two-factor model (see e.g. Lucia and Schwartz, 2002, section 3.3)) or a GARCH model for the volatility (see Byström, 2005).

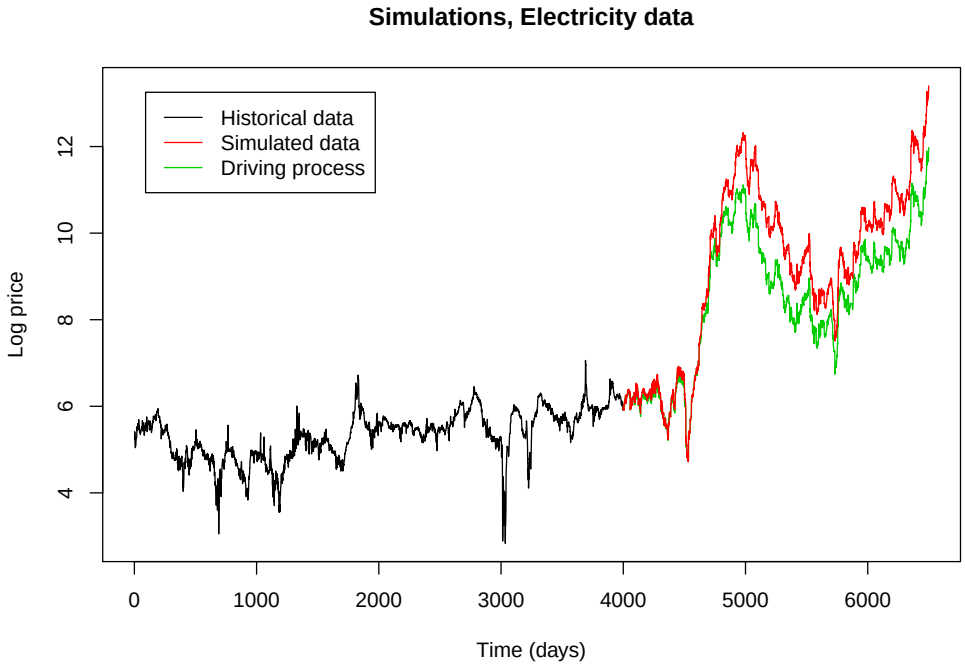

Figure 20: Simulation from periodic AR model with NIG distributed residuals, using electricity data.

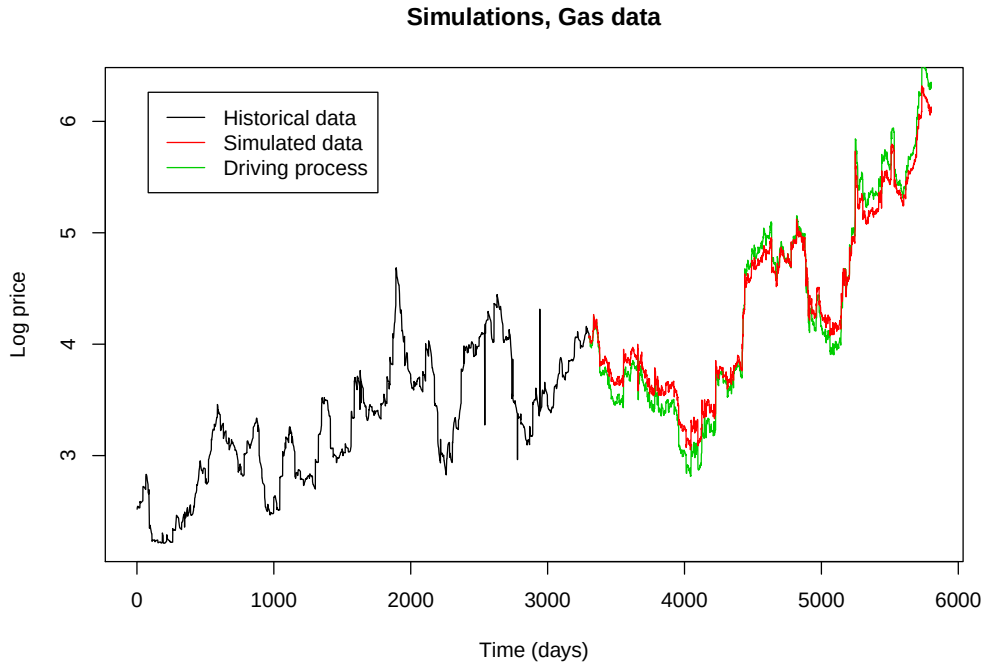

Figure 21: Simulation from periodic AR model with NIG distributed residuals, using gas data.

In order to introduce price spikes into the model, we could consider a two factor jump-diffusion model (see e.g. Cartea and Figueroa, 2005), where small movements are modelled as a Brownian motion and large jumps occur according to a Poisson process. Before estimating the parameters of such a model, one has to find a way of identifying and filtering out the jumps in order to separate the two simultaneous processes (see e.g. Johannes et al., 2009).

In both of models the residuals are modelled as variables from a NIG distribution. We have seen that the NIG distribution fits these residuals much better than the normal distribution does, mainly because of the leptokurtic behaviour of the residual process. There may be some inaccuracy in the tails, and we have seen that residuals from the oil data deviate more from the NIG distribution than residuals from gas and electricity data do.

The model with deterministic seasons and mean reversion produce what seems like reasonable price developments. However, because of the assumption of constant volatility, this model is likely to overestimate the probability of small jumps and underestimate the probability of extreme jumps. That trend, seasonal and weekly variations are all taken to be deterministic, in combination with the mean reversion dynamics, may also cause the model to be too conservative with respect to risk.

Simulations from the periodic AR model show that it behaves almost like a NIG process. The lack of trend, strong seasonal variations and mean reversion makes this model better suited for forecasting in the short run than in the long run. However, it is less prone to underestimating the risk of extreme prices (both upwards and downwards), and may thus be interesting for the purpose of risk management.

These conciderations are supported by simulation results. By performing 3000 Monte Carlo runs for each model and computing approximate forecasting distributions (see Figure 22), we see how the periodic AR model admits larger probabilities for unexpected price developments in the long run. The mean reversion effect makes the former model largely dependent on the deterministic component in the long run, to such an extent that the forecast variance barely increases with time, while the forecast variance of the periodic AR model increases linearly with time (see Figure 23).

For the purpose of applying the Path Integral method for forecasting and pricing of derivatives, there is a clear advantage in using an AR model that is only dependent on the previous time step, i.e. an AR(1) model. When we introduce more AR parameters, like in the periodic AR models, this will complicate the integration procedure. This poses an interesting problem, but it will not be pursued in this thesis.

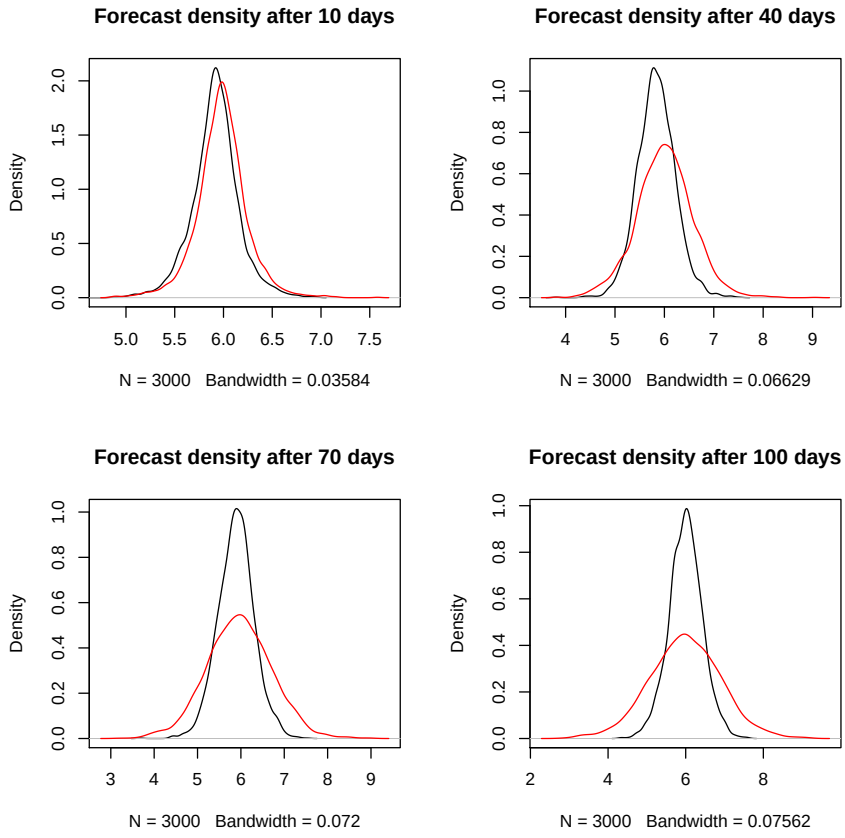

Figure 22: Forecasted probability density function at different times, obtained with 3000 Monte Carlo simulations. Results from the model with deterministic variations and mean reversion is shown in black, and the periodic AR model is shown in red.

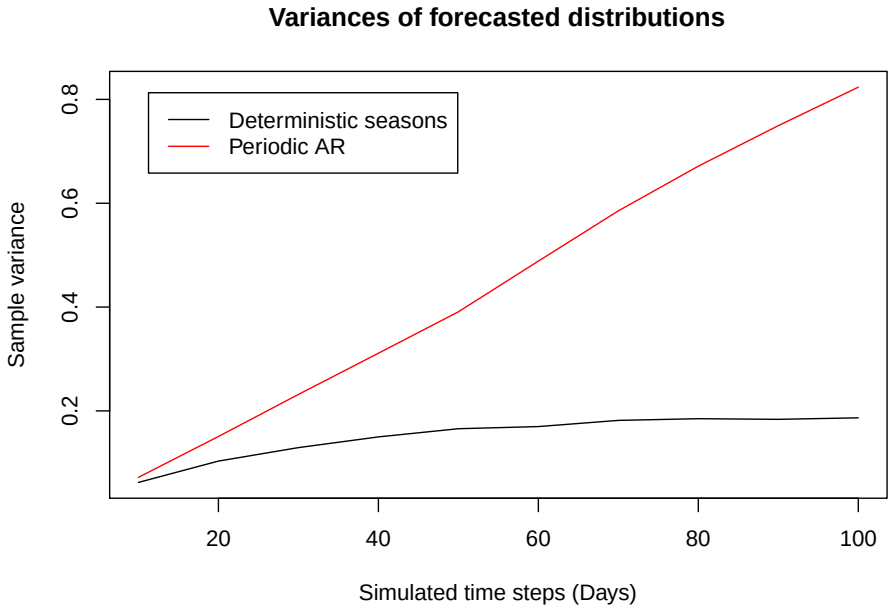

Figure 23: Variance of forecast density as a function of time for the two models considered.

6 Valuing energy derivatives with numerical path integrals

6.1 Implementation of density forecast

In order to find the forecast density $p(s_T)$ of the energy price model in Section 5.2, we need to solve Equation (20) numerically. Since all the stochastic behaviour takes place in the AR(1) model, we can simplify the implementation by working with the detrended/deseasonalized price process X_t.

When estimating the parameters of the model, there is the question of how much history to take into account. Price developments 10 years ago may not be relevant when forecasting one year ahead. The deterministic mean function $\Lambda(t)$ in Equation (49) will depend on the choice of data used in the estimation procedure. In Figure 24 we see how different choices will lead to different estimates. Not only do we get different amplitudes for the seasonal variations, we even get a negative trend in one of the four examples. We will not dwell on this problem at present, but rather adopt the convention that when we want to forecast m days into the future, then we will use the previous m days to estimate model parameters.

We start by fitting $\Lambda(t)$ to the appropriate series of log prices, and subtract the resulting function in order to obtain the AR process $\hat{X}_t$. From $\hat{X}_t$ we estimate the AR(1) parameter $\hat{\varphi}$, and from the residuals we estimate the NIG parameters $\hat{\alpha}$, $\hat{\beta}$, $\hat{\delta}$ and $\hat{\mu}$. Then we run 1000 simulations from X_t, $t = 1, \ldots, m$, to determine an interval $[x_{min}, x_{max}]$ where the probability density $p(x_t)$ is non-zero, and we expand the interval by 100% to include most of the probability mass.

In order to perform the numerical integration, we set up a uniform grid of size N over the interval $[x_{min}, x_{max}]$. The probability density in grid point i at time t is denoted $g_t^{(i)}$. We will only calculate the probability density in the gridpoints, and we approximate the full density function by a cubic spline interpolation with end conditions $g_t^{(1)} = g_t^{(N)} = 0$ for every t.

6.1.1 Cell mapping

A fairly simple way of propagating the $p(x_t)$ forward in time is by looking at the conditional probability densities for the grid points and treating them as if they were discrete conditional probabilities. This method is sometimes referred to as "cell mapping", but it is in fact numerical integration in its simplest form. Computationally it is exceptionally fast, because it relies only on matrix multiplication.

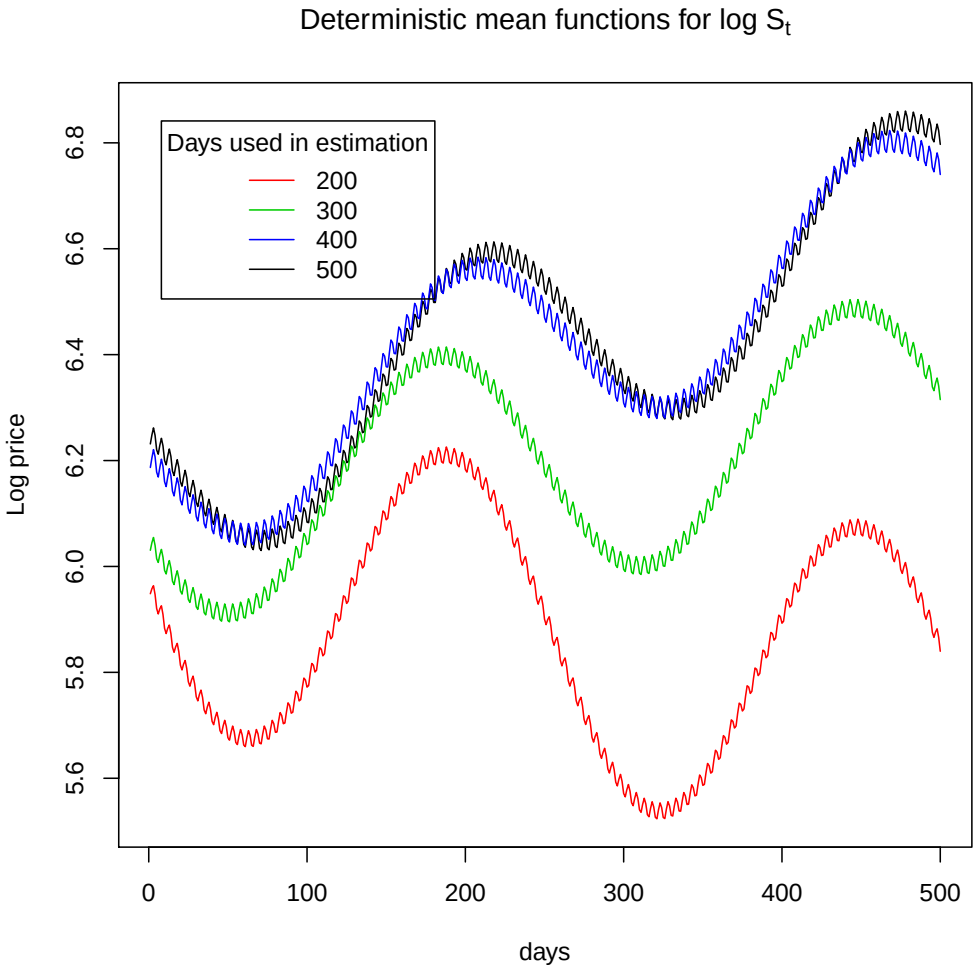

Figure 24: Different estimates of the deterministic mean log price, depending on how many days of history we use in the estimation. These functions are estimated from the Nord Pool daily spot prices.

Let $x^{(i)}$ be the position of the i'th grid point, and $p(x^{(i)}|x^{(j)})$ the probability density for getting to $x^{(i)}$ given $x^{(j)}$ in the previous time step. In our model these densities are from a NIG distribution with the location parameter adjusted by the AR dynamics:

$$p(x^{(i)}|x^{(j)}) = f_{\text{NIG}}(x^{(i)}; \hat{\alpha}, \hat{\beta}, \hat{\delta}, \hat{\mu} + \hat{\varphi}_1 x^{(j)}) \tag{56}$$

We can transform these densities into discrete probabilities by finding constants r_i such that the following relation is satisfied:

$$r_j \sum_{i=1}^{N} p(x^{(i)}|x^{(j)}) = 1 \quad \text{for} \quad j = 1, \ldots, N\,. \tag{57}$$

Then we can construct a transition matrix by:

$$P = \begin{bmatrix} r_1 p(x^{(1)}|x^{(1)}) & r_2 p(x^{(1)}|x^{(2)}) & \ldots & r_N p(x^{(1)}|x^{(N)}) \\ r_1 p(x^{(2)}|x^{(1)}) & r_2 p(x^{(2)}|x^{(2)}) & \ldots & r_N p(x^{(2)}|x^{(N)}) \\ \vdots & \vdots & \ddots & \vdots \\ r_1 p(x^{(N)}|x^{(1)}) & r_2 p(x^{(N)}|x^{(2)}) & \ldots & r_N p(x^{(N)}|x^{(N)}) \end{bmatrix}. \tag{58}$$

Now, if we let g_t be a normalized vector of grid probabilities at time t, we can propagate the probabilities forward in time by multiplication

$$g_{t+1} = P g_t \tag{59}$$

and since we know g_1, we have

$$g_m = P^{m-1} g_1\,. \tag{60}$$

Now we can use the discrete probabilities g_m to approximate the value of a European option.

If we need a continuous approximation, we can perform spline interpolatation on the probability vector g_m and integrate the spline once to find a normalization constant. (Alternatively we can let g_1 be a vector of densities and propagate on the densities directly, obtaining a density vector g_m.) The normalized spline function is our forcasted probability density $p(x_m)$.

6.1.2 Integration with Simpson's Rule

The method above only takes into account the conditional densities at the grid points $x^{(i)}$, $i = 1, \ldots, N$. A more precise result can be obtained by solving the integral in Equation (19) numerically. We perform the integration

by use of Simpson's Rule (see e.g. Burden and Faires, 2005), which is given by

$$\int_{x_0}^{x_2} f(x)dx = \frac{h}{3}[f(x_0) + 4f(x_1) + f(x_2)] \tag{61}$$

where $x_1 = x_0 + h$ and $h = (x_2 - x_0)/2$. The error of Simpson's rule is $\varepsilon = \frac{h^5}{90} f^{(4)}(\xi)$, with $x_0 < \xi < x_2$.

We could use Simpson's rule with only the grid points $x^{(i)}$, $i = 1, \ldots, N$. To improve accuracy we set up finer grids $x^{(i,j)}$, $j = 1, \ldots, K$ over limited intervals $[a_i, b_i]$, such that $p_{x^{(i)}|x'}(x')$ is below some tolerance level close to zero whenever x' is outside $[a_i, b_i]$. In this way we don't waste computer time on integrating where the conditional densities are very close to zero, and it allows us to obtain better precision on the interval where most of the probability mass is located.

For each grid point $x^{(i)}$, we calculate the conditional densities $p(x^{(i)}|x^{(i,j)})$ for $j = 1, \ldots, K$. The value of the integrand at the j'th grid point is

$$f_{i,j} = p(x^{(i)}|x^{(i,j)})p(x^{(i,j)})\,. \tag{62}$$

and h_i is the distance between adjacent points in the fine grid $x^{(i,j+1)} - x^{(i,j)}$. Now, let $k = 1, 3, 5, \ldots, K-2$. The probability density in $x_{t+1}^{(i)}$ approximated by Simpson's rule is then given by

$$g_{t+1}^{(i)} = \sum_{k=1}^{K-2} \frac{h_i}{3}[f_{i,k} + 4f_{i,k+1} + f_{i,k+2}] \tag{63}$$

Finally we set $g_{t+1}^{(0)} = g_{t+1}^{(N)} = 0$ and perform cubic spline interpolation on g_{t+1} to obtain $p(x_{t+1})$. This procedure can be done iteratively to obtain $p(x_m)$. Note that the grid points and conditional densities need only be calculated once, as they do not change with time.

6.1.3 Numerical results for density forecast

To assess the performance of the methods discussed above, we calculate the density of X_t after $t = 300$ days, using the Nord Pool daily system price data. We do the calculations with both the cell mapping technique and integration with Simpson's Rule. In addition, we perform repeated Monte Carlo simulations with the model for comparison with the PI calculations.

The model parameters is estimated from the last 300 prices in the data set. In Table 5 we show the results of the estimation procedure. We see from the value of $\hat{\varphi}$ that the estimated mean reversion is fairly strong compared to what we found in Section 5.2.

	$\hat{\varphi}$	$\hat{\alpha}$	$\hat{\beta}$	$\hat{\delta}$	$\hat{\mu}$
Estimate	0.90527	10.02191	-0.21844	0.03226	-0.00154

Table 5: Estimated parameters for detrended/deseasonalized process X_t, based on previous 300 daily electricity prices.

Because of the mean reversion dynamics, the distribution of X_t will be only temporarily dependent on the starting value. In the long run, we expect the distribution to converge, such that we have $X_{t+1} \overset{approx}{\sim} X_t$ for large t. From Figure 25 we see how the densities evolve for $t \leq 30$. After $t = 30$ the densities are nearly unchanged in each time step. The rate of convergence depends on the starting value x_0 and the speed of the mean reversion. Thus, for large t, the density of the model price will depend only on the model parameters as the starting value becomes insignificant.

We approximate $p(x_{300})$ for the Nord Pool system price model with cell mapping, Simpson's Rule and with MC simulations. We use $N = 100$ grid points to monitor the probability density, and $K = 1000$ grid points for the fine grid in the integration procedure with Simpson's Rule. The number of MC runs are 10^6. With our implementation running on a standard lap top computer the cell mapping procedure finished in about 0.1 seconds, while the integration procedure finished in about 250 seconds. The MC implementation finished in about 2500 seconds on the same computer. In Figure 26 we have plotted the approximate densities obtained. If we superimposed the plots we would hardly have been able to discriminate between them, except for a small "dent" in the MC density around $x = 0.02$, which we were not able to reproduce even with 10 times as many grid points. From Table 6 we see that the moment characteristics of the three density forecasts are almost the same.

	Monte Carlo	Cell mapping	Simpson's Rule
Mean	-0.02373	-0.02374	-0.02372
Variance	0.01787	0.01788	0.01784
Skewness	-0.02336	-0.03374	-0.03413
Kurtosis	0.92804	0.91277	0.91854

Table 6: Approximate mean, variance, skewness and kurtosis of forecasted distributions.

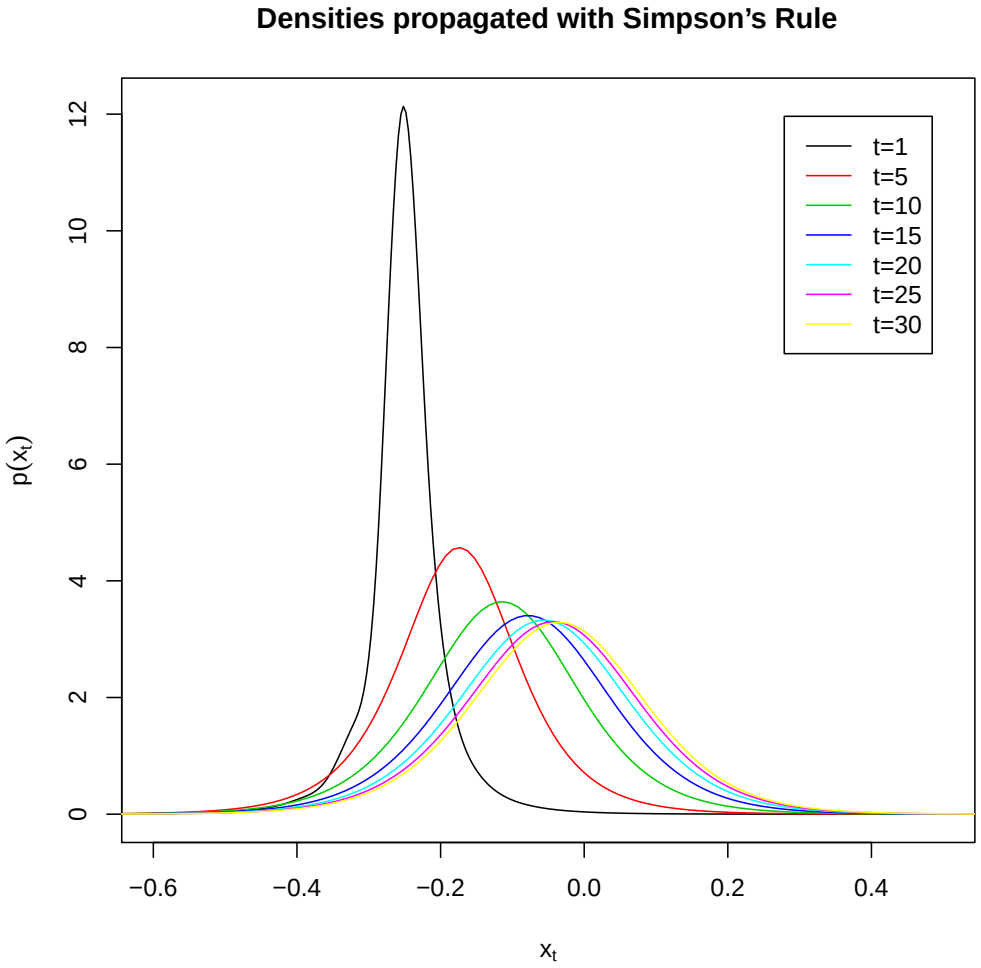

Figure 25: Densities $p(x_t)$ evolving with time, propagated by integration with Simpson's Rule.

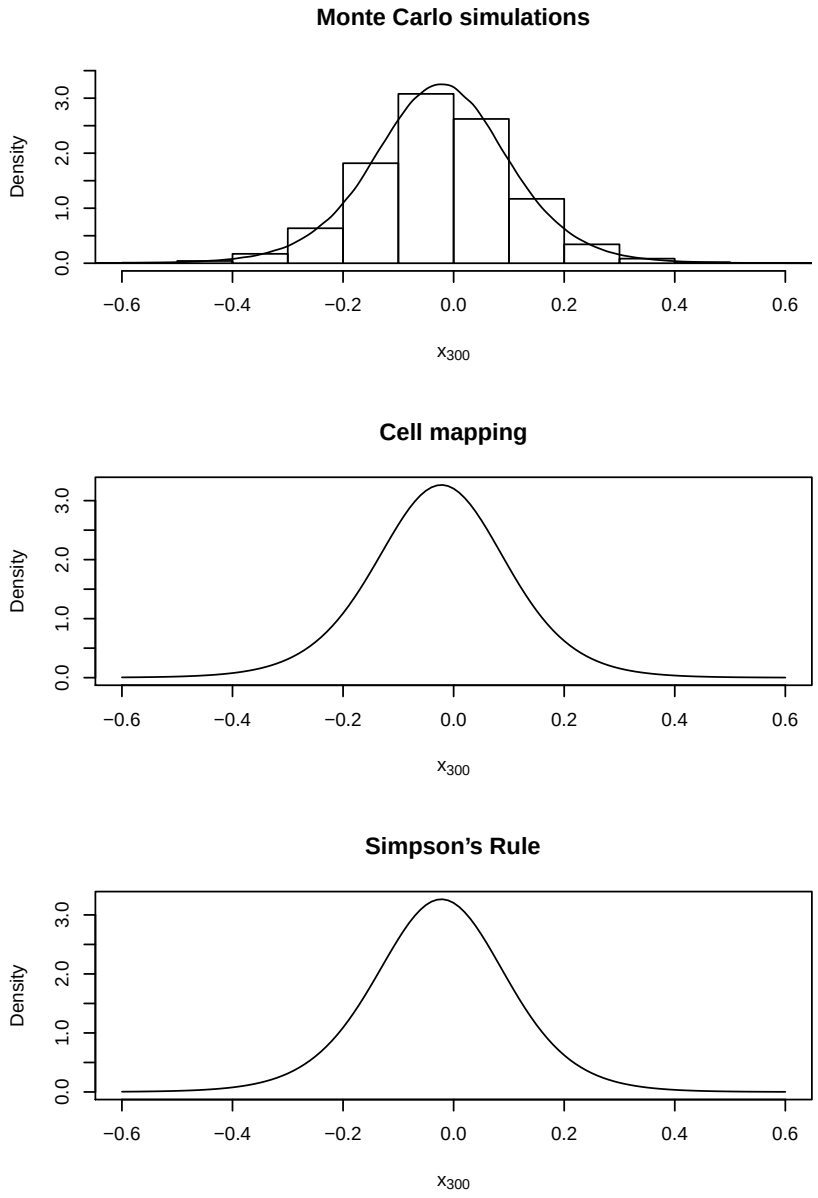

Figure 26: Approximations to $p(x_{300})$ with different methods.

6.2 Valuing path-independent derivatives based on forecasted densities

Once we have obtained some representation of the probability density function $p(x_T)$, we can derive the value of a European option or a forward contract at some specified exercise price by evaluating the respective integrals in Equations (25), (26) or (27).

In our Path Integral implementations we found the densities of the detrended/deseasonalized AR(1) process X_t at $t = T$, which we here denote $p_{X_T}(x)$. Now, let Y_t be the log price process $\ln(S_t)$. We have $Y_t = \ln \Lambda(t) + X_t$. Then, the density of Y_T is given by:

$$p_{Y_T}(y) = p_{X_T}(y - \ln \Lambda(T)) \,. \tag{64}$$

We can evaluate the price of a derivative by integrating in the log scale. Since all of the probability mass of our approximate $p_{X_T}(x)$ is located within the end points of a grid $x^{(i)}$, we can subsitute the infinite limits of integration by finite values. Thus, by changing variables, Equation (25) becomes

$$C(T, E) = \int_{\ln E}^{x^{(N)} + \ln \Lambda(T)} (\exp(y) - E) p_{X_T}(y - \ln \Lambda(T)) dy \,. \tag{65}$$

For put options we get

$$P(T, E) = \int_{x^{(1)} + \ln \Lambda(T)}^{\ln E} (E - \exp(y)) p_{X_T}(y - \ln \Lambda(T)) dy \tag{66}$$

and the value of a forward contract becomes

$$V(T, E) = \int_{x^{(1)} + \ln \Lambda(T)}^{x^{(N)} + \ln \Lambda(T)} (\exp(y) - E) p_{X_T}(y - \ln \Lambda(T)) dy \,. \tag{67}$$

These integrals can be evaluated numerically to obtain the respective derivative values under the assumed model.

6.2.1 Numerical results for path-independent derivatives

We compute derivative values for $T = 300$ days at different exercise prices. The results are shown in Table 7. We observe that the prices from the PI methods are very close to the prices obtained from MC simulations. We do not see any clear evidence that cell mapping is less accurate than integration by Simpson's Rule.

In Figure 27 we have plotted the derivative values as functions of exercise/forward price. We have $C(T,E) = P(T,E)$ when $E = \text{EXP}[S_T]$. The put-call parity is obeyed and the values behave as we would expect.

We also compute option prices for different values of T, as shown in Figure 6.2.1, and again we observe a close match between MC and PI results. The PI implementation with Simpson's Rule is about 10 times faster than the MC implementation.

We can construct confidence intervals for the option values from the MC results, using the test statistic $t_{n-1} = [\overline{x} - \mu]/[s/\sqrt{n}]$, where t_{n-1} is t-distributed with $n-1$ degrees of freedom. We detect errors in $C(250, 600)$ at the 0.05 level. The confidence interval for this option is $[170.6895, 171.1095]$, and both of the PI methods overestimate the option value. The other results are within the confidence interval.

To see how the input parameters affect the precision, we rerun the PI algorithms for different values of N and K. The results are shown in Figures 28 and 29. For cell mapping, we see that the errors are quite large when the grids are sparse, as one would expect. The behaviour of the Simpson's rule method is rather unexpected with small errors for sparse grids. Both methods stabilize at about $N = 150$ and $K = 200$. The option values converge to a level close to, but not inside, the confidence interval.

E	Method	$C(T,E)$	$P(T,E)$	$V(T,E)$
500	MC	138.5592	1.3698	137.1894
	Cell mapping	138.5803	1.3597	137.2216
	Simpson's Rule	138.5784	1.3575	137.2219
600	MC	54.0839	16.89451	37.18939
	Cell mapping	54.10426	16.88292	37.22197
	Simpson's Rule	54.06196	16.84004	37.22255
700	MC	12.33172	75.10708	-62.77536
	Cell mapping	12.28608	75.0634	-62.77732
	Simpson's Rule	12.28495	75.0626	-62.77758

Table 7: Model prices for European options and forward contracts at maturity $T = 300$, calculated from forecasted densities.

6.3 Valuing barrier options with Path Integrals

In this section we look at implementations of a Path Integral method for valuing knock-out and knock-in options with an upper barrier. We let m

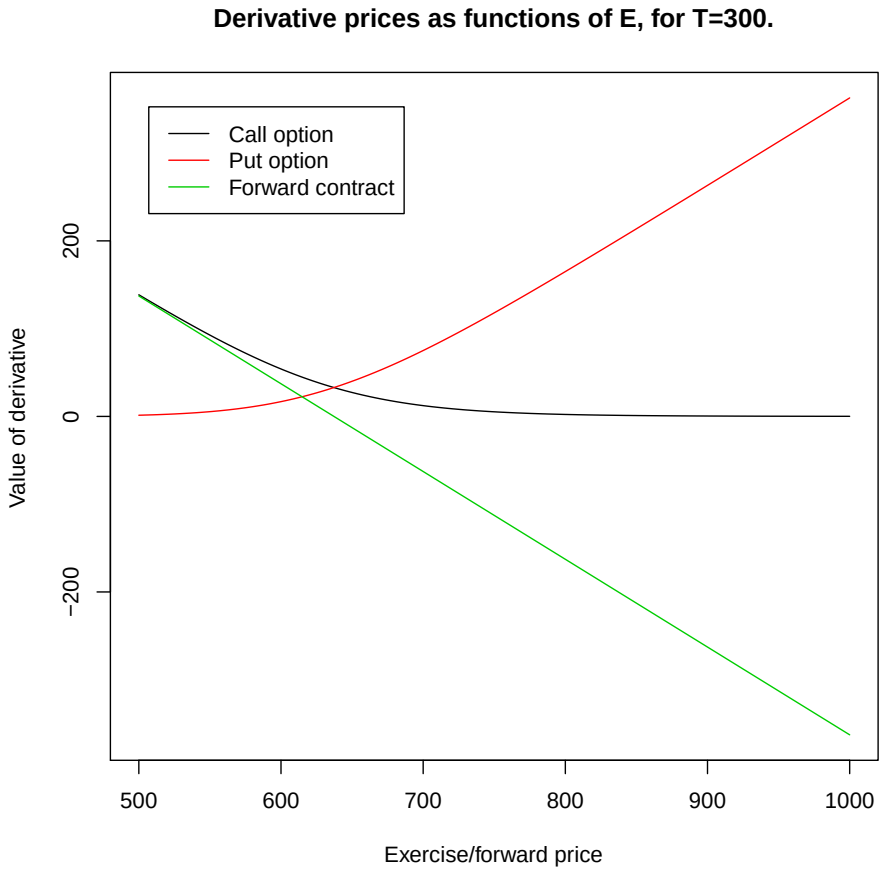

Figure 27: Values of European options and forward contract with expiry/maturity T=300, as a function of exercise/forward price. These prices are calculated from densities obtained by Path Integrals with Simpson's Rule.

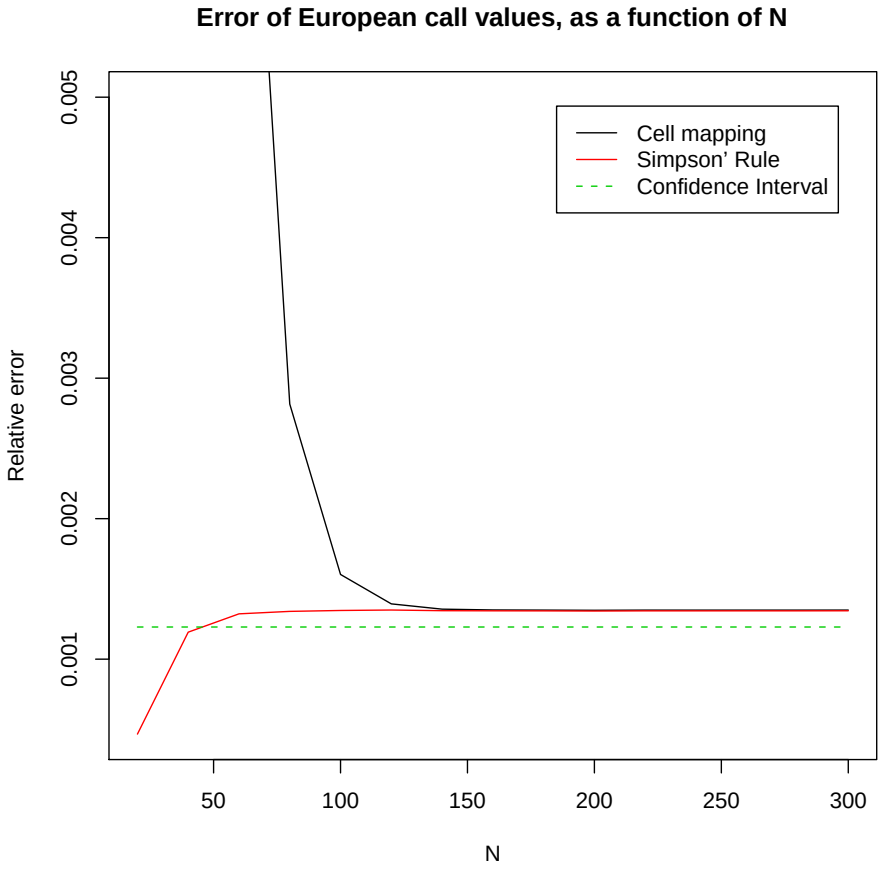

Figure 28: Relative deviation from MC result for European Call options with $T = 250$, $E = 600$ and $K = 1000$. The results converge at about $N = 150$.

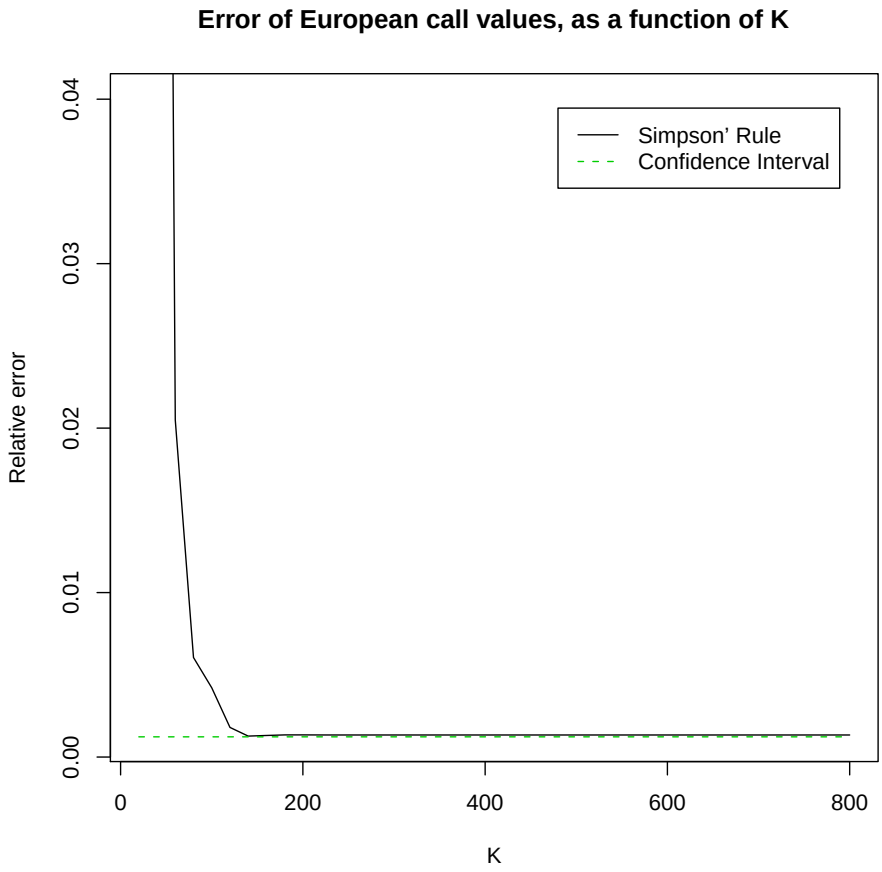

Figure 29: Relative deviation from MC result for European Call options with $T = 250$, $E = 600$ and $N = 150$. The results converge at about $K = 200$.

T	Method	CPU time	$C(T,E)$	$P(T,E)$
200	Monte Carlo	1885	1.2510	112.5134
	Cell mapping	0.14	1.2523	112.4547
	PI w/Simpson	164	1.2503	112.5319
250	Monte Carlo	2284	170.8995	1.7775
	Cell mapping	0.14	171.1734	1.7828
	PI w/Simpson	201	171.1296	1.7797
300	Monte Carlo	2692	54.0839	16.8945
	Cell mapping	0.11	54.1043	16.8829
	PI w/Simpson	243	54.0620	16.8400

Table 8: Numerical results for European options with $E = 600$. The CPU time is given in seconds. Estimates outside MC confidence interval are shown in red.

be the number of days until expiry, and we assume that the price of the underlying is monitored on a daily basis, i.e. at $t = 1, \ldots, m$. We show how to calculate the Path Integrals with Simpson's method, and verify results by use of MC simulations.

6.3.1 Implementation of Path Integral method for up-and-out barrier options

To propagate the H-density forward in time, we need to solve Equation (31) numerically. The procedure is basically the same as the one described in Section 6.1.2. We work in detrended/deseasonalized space, where the stochastic dynamics take place. We set up a grid $x^{(i)}$ to represent the density function on, and we let the upper end of the grid be slightly above the maximum barrier position. Then for each $i = 1, \ldots, N$ we set up a integration grid $x^{(i,j)}$, $j = 1, \ldots, K$, which is used to perform integration by Simpson's Rule. In each time step we interpolate the grid and corresponding densities by a cubic spline to approximate the full density. But now we cut off the spline at the barrier, such that the approximated density $h_t(x)$ is zero above the barrier. The resulting cut-off density is then used in the integrand at the next step.

Since we are doing the Path Integral procedure on the AR(1) process X_t, the barrier depends on $\ln \Lambda(t)$, so we have to use the correct x_t^{barrier} in each step. Moreover, to avoid approximation errors that might arise from the discontinuity at the barrier, we set up a fine grid $x^{(g)}$, $g = 1, \ldots, G$, and calculate the densities at these grid points as well. We place these

extra grid points in an interval where we expect barrier effects to occur, i.e. symmetrically around the previous cut-off value multiplied by φ. This grid and corresponding $G \times K$ propagator densities has to be calulated in every time step, as the barrier value changes. This slows the algorithm down quite a bit.

Because the density function is cut off in each step, some of the probability mass will disappear. The mass that disappear at $t = j$ corresponds to the fraction of the processes X_t that will cross the barrier at $t = j$. The total mass of $h_m(x)$, i.e. $H_m(-\infty)$, is the fraction of processes that do not cross the barrier in the life time of the option. We can use $h_m(x)$ directly to value the up-and-out call and put options, by rewriting Equations (65) and (66) to obtain:

$$C(m, E, B) = \int_{\ln E}^{\ln B} (exp(y) - E) h_m(y - \Lambda(m)) dy \tag{68}$$

and

$$P(m, E, B) = \int_{x^{(1)} + \ln \Lambda(m)}^{\ln E} (E - exp(y)) h_m(y - \Lambda(m)) dy \,. \tag{69}$$

In practice, since $h_m(y - \Lambda(m)) = 0$ whenever $y > \ln B$, we use the same implementation to evaluate these expressions as we did for the European option, substituting $p_m(x)$ by $h_m(x)$ in the argument.

6.3.2 Numerical results for up-and-out barrier options

We use the Nord Pool electricity spot prices as input to the algorithm, and we evaluate put and call options with different expiry dates and barrier prices. For the Path Integral method we use $N = 100$ grid points for the density representation, and $K = 1000$ points in the integration grid. To increase precision of the density function around the barrier, we add 20 grid points in an interval that is set to 5 times the length between the normal grid points. We also run the algorithm without this barrier grid. For comparison we run $n = 10^6$ MC simulations of the same barrier process.

In Figure 30 we have plotted the H-densities obtained from these three algorithms. The approximated densities are so close to each other that they can hardly be discriminated by visual inspection, even when we superimpose the plots. On the right tail we see that the density is cut off at the barrier.

In Table 6.3.2 we display some of the results from running the algorithms. Some of the option values falls outside the confidence interval, but all deviations from the MC results are relatively small. The speed of the MC algorithm depends on the parameters, because we stop the iteration each time a

process crosses the barrier. As expected, the PI implementation with extra grid points around the barrier is somewhat slower than if we use a uniform grid. However, in our test runs we found no significant loss of precision by using the uniform grid.

6.3.3 Implementation of PI method for up-and-in barrier options

We expand a grid with N points over the same interval as for the density forecasts. This grid is then extended by a finer grid of G points in an interval where we expect to see effects from the discontinuity at the barrier.

In each time iteration we propagate both $h_t(x)$ and $g_t(x)$ over the whole grid by applying Simpson's Rule with K integration points. The implementation will spend twice as much CPU time as the implementation for up-and-out barrier processes, as we have to do twice as many calculations. The part of $g_t(x)$ that flows over the barrier is cut off and added to $h_t(x)$ in each step. In Figure 31 we see how $h_t(x)$ develops around the barrier.

After iterating through all the time steps, the value of an up-and-in barrier option can be calculated by the same procedure as before, using $h_m(x)$ instead of $p_m(x)$ in the integrand in Equations (65) and (66).

6.3.4 Numerical results for up-and-in barrier options

We test run the PI implementation for up-and-in barrier options with $N = 100$ and $K = 1000$, for some combinations of B, E and T. We also calculate the values by using the in-out parity of barrier options, using Cell mapping with $N = 150$ for the European option and Simpson's Rule without barrier grid for the up-and-out option. The results are shown in Table 6.3.4.

By dropping the barrier grid, the algorithm runs about twice as fast, but from our test runs we cannot conclude that there is any significant loss of precision. The results from using the in-out parity are obtained even faster, but the quality of the results vary from significantly better to significantly worse.

6.4 Discussion

We have shown how to implement PI methods for an AR(1) model with NIG distributed residuals, and we have shown how these methods can be used to value derivatives. The PI implementations are considerably faster than the corresponding MC implementations and they produce reasonable results. Some of the options values found by the PI methods are not inside the MC

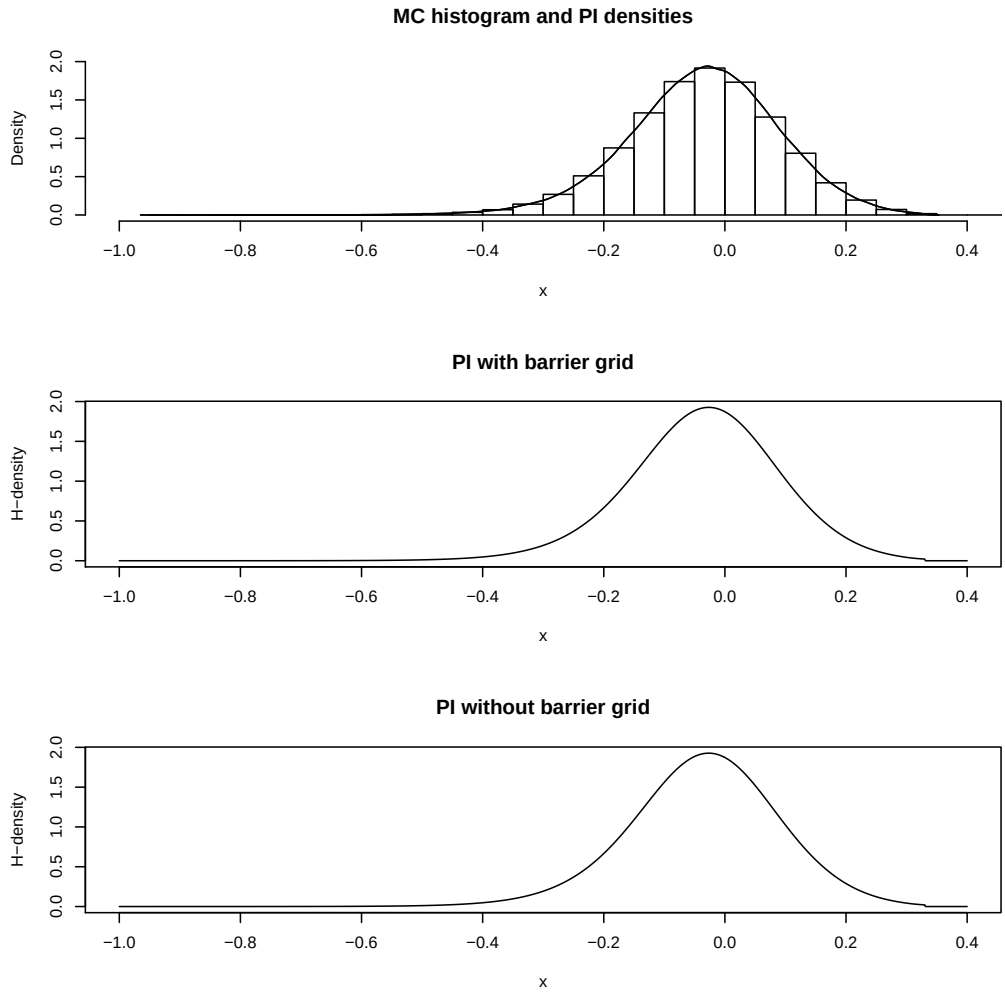

Figure 30: Histograms for MC simulations of X_{300}, and corresponding densities obtained with PI methods. The approximations are very close to eachother.

Table 9: Numerical results for up-and-out barrier processes with different expiry, exercise price and barrier values. The CPU time is given in seconds. Estimates outside of MC confidence interval are shown in red.

T	E	B	Method	CPU time	$H_m(-\infty)$	$C(T,E,B)$	$P(T,E,B)$
200	600	900	Monte Carlo	2807	0.998703	1.182144	112.4794
			PI with barrier grid	596	0.999403	1.196652	112.5342
			PI without barrier grid	176	0.995904	1.192350	112.1393
250	780	1300	Monte Carlo	2737	0.948202	34.29881	46.38103
			PI with barrier grid	747	0.948553	34.29263	46.39494
			PI without barrier grid	211	0.946785	34.22921	46.30963
300	600	900	Monte Carlo	3434	0.571642	26.79804	10.28791
			PI with barrier grid	880	0.570520	26.82146	10.27437
			PI without barrier grid	259	0.570682	26.83117	10.27706
350	500	850	Monte Carlo	3066	0.539866	15.27036	16.28538
			PI with barrier grid	1023	0.535164	15.09606	16.14556
			PI without barrier grid	296	0.543417	15.33142	16.39373

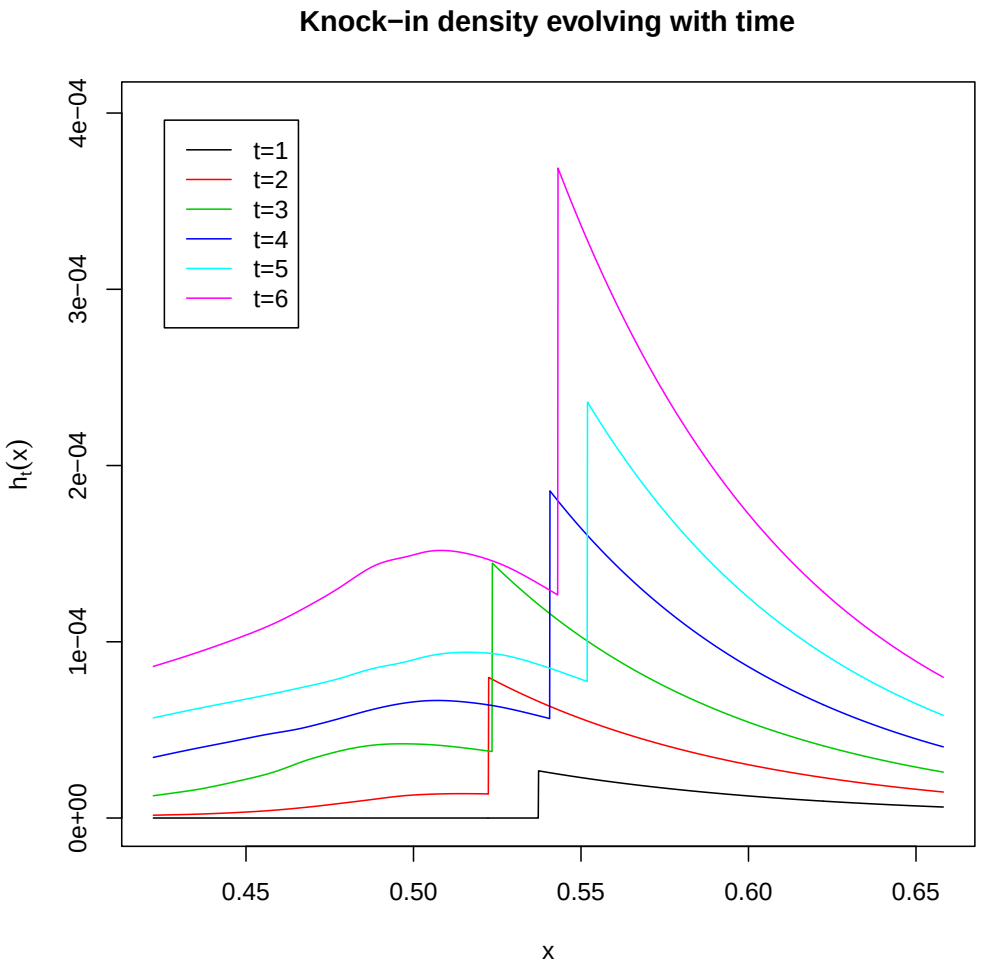

Figure 31: Development of the knock-in-density $h_t(x)$ around the barrier. In each time step some mass is added above the barrier and some of the density mass flows back below the barrier.

T	E	B	Method	CPU time	$H_m(-\infty)$	$C(T,E,B)$	$P(T,E,B)$
200	500	600	Monte Carlo	2320	0.352801	10.91859	6.019776
			PI with barrier grid	947	0.334996	11.13991	6.403226
			PI without barrier grid	573	0.416805	11.59898	6.761833
			In-Out Parity	180	0.356603	10.79003	6.196707
250	780	1200	Monte Carlo	2846	0.131563	8.166562	5.414192
			PI with barrier grid	1184	0.1301167	8.148202	5.473311
			PI without barrier grid	579	0.1299593	8.139486	5.466824
			In-Out Parity	220	0.1312831	8.136403	5.467996
300	600	900	Monte Carlo	3371	0.426954	27.1697	6.530322
			PI with barrier grid	1433	0.4251654	27.40383	6.60442
			PI without barrier grid	694	0.4249642	27.38936	6.601416
			In-Out Parity	264	0.4293247	27.24158	6.565924
350	500	850	Monte Carlo	3793	0.459681	16.68078	13.16960
			PI with barrier grid	1670	0.4582669	16.78025	13.24401
			PI without barrier grid	807	0.457325	16.74659	13.21731
			In-Out Parity	297	0.456579	17.01298	13.29620

Table 10: Numerical results for up-and-in barrier processes with different expiry, exercise price and barrier values. The CPU time is given in seconds. Estimates outside of MC confidence interval are shown in red.

confidence interval. The reason for these observations are not clear and could be explored further.

The performance of the PI algorithms depend on many variables; the number of grid points representing densities, the resolution of the numerical integration procedure, the size of the interval on which we approximate densities, and the parameters of the price model. The forecasted densities should in principle approach the true density as we increase the resolution of the representation and integration grids. Further study is needed to determine how variables should be chosen in order to achieve consistently good results, and to improve the implementations with a view to minimizing error.

For barrier option methods, one source of error is the discontinuity in the integrand that arise from cutting the the density function. At this point the error of Simpson's Rule diverges. The problem is partly contained by keeping K large, thus keeping the step size h small, but it could be completely remedied by splitting the integration grid at the cut-off point, and using the correct limit value depending on which side of the discontinuity we integrate.

Another idea is to use cell mapping for barrier options too. The speed of this method is so formidable that we can increase the grid resolution and still achieve fast results. However, matrix multiplication has a running time of about $O(n^3)$, so the speed is not without limitation. Additional grid points should be added in a way that optimizes precision, e.g. by having a denser grid around the barriers.

The results show that the PI approach can be a viable solution to valuation of derivatives under models of the kind considered here. For practical applications we would have to look into how the price model should be calibrated to achieve the best possible forecasts. For faster execution of the algorithms, we would implement them in a different language, e.g. Matlab or C.

A Terms and Definitions

Definition 7 (Stochastic continuity)**.** Let $X = \{X(t), t \geq 0\}$ be a stochastic process. If, for all $a > 0$ and for all $s \geq 0$, $\lim_{t \to s} P(|X(t) - X(s)| > a) = 0$, then X is *stocastically continuous*.

Definition 8 (Characteristic function)**.** The *characteristic functon* ϕ_X of a random variable X uniquely determines the density function $F(x) = P(X \leq x)$. ϕ_X is the Fourier-Stieltjes transform of $F(x)$ (Schoutens, 2003, p. 15), such that:

$$\phi_X(u) = E[\exp(\mathrm{i}uX)] = \int_{-\infty}^{+\infty} \exp(\mathrm{i}ux)\mathrm{d}F(x) \tag{70}$$

Definition 9 (Infinite divisibility)**.** A distribution defined by the characteristic function $\phi(u)$ is said to be *infinitely divisible* if, for every positive integer n, $\sqrt[n]{\phi(u)}$ is also a characteristic function (Schoutens, 2003, p. 44).

Definition 10 (Lévy measure)**.** A *Lévy measure* ν is a probability measure on $\mathbb{R} \setminus \{0\}$ with

$$\int_{-\infty}^{+\infty} \inf\{1, x^2\}\nu(\mathrm{d}x) = \int_{-\infty}^{+\infty} \inf\{1 \wedge x^2\}\nu(\mathrm{d}x) < \infty \tag{71}$$

(Schoutens, 2003, p. 45)

B Proofs and derivations

B.1 NIG parameters from moment characteristics

The mean, variance, skewness and kurtosis of the $NIG(\alpha, \beta, \delta)$ distribution are given by (Schoutens, 2003), and we add the location parameter μ to this parameterisation:

$$m_1 = \delta\beta/\sqrt{\alpha^2 - \beta^2} + \mu \tag{72}$$

$$m_2 = \alpha^2\delta(\alpha^2 - \beta^2)^{-3/2} \tag{73}$$

$$\gamma_1 = 3\beta\alpha^{-1}\delta^{-1/2}(\alpha^2 - \beta^2)^{-1/4} \tag{74}$$

$$\gamma_2 = 3\left(\frac{\alpha^2 + 4\beta^2}{\delta\alpha^2\sqrt{\alpha^2 - \beta^2}}\right) \tag{75}$$

From (73) and (74) we can write

$$\frac{\gamma_1}{\sqrt{m_2}} = \frac{3\beta\sqrt{\alpha^2 - \beta^2}}{\delta\alpha^2} \tag{76}$$

and (75) can be written as

$$\gamma_2 = \frac{3\sqrt{\alpha^2 - \beta^2}}{\delta\alpha^2} + \frac{5}{3}\left(\frac{9\beta^2}{\delta\alpha^2\sqrt{\alpha^2 - \beta^2}}\right). \tag{77}$$

By substituting (76) and the square of (74) into (77) we obtain

$$\gamma_2 = \frac{\gamma_1}{\sqrt{m_2}\beta} + \frac{5}{3}\gamma_1^2 \tag{78}$$

and thus we have

$$\beta = \frac{\gamma_1}{\sqrt{m_2}(\gamma_2 - \frac{5}{3}\gamma_1^2)}. \tag{79}$$

Now, from (73) and (74) we can write

$$\sqrt{m_2}\gamma_1 = \frac{3\beta}{\alpha^2 - \beta^2} \tag{80}$$

and (75) can be written as

$$\gamma_2 = 3\left(\frac{1}{\delta\sqrt{\alpha^2 - \beta^2}} + \frac{4\beta^2}{\alpha^2\delta\sqrt{\alpha^2 - \beta^2}}\right). \tag{81}$$

By substituting (80) into (81) we obtain

$$\gamma_2 = \gamma_1^2 \left(\frac{\alpha^2}{3\beta^2} + \frac{4}{3} \right) \tag{82}$$

$$\alpha = \beta \sqrt{\frac{3\gamma_2}{\gamma_1^2} - 4}. \tag{83}$$

Subtituting (79) into (83) we obtain

$$\alpha = \frac{\sqrt{3\gamma_2 - 4\gamma_1^2}}{\sqrt{m_2}(\gamma_2 - \frac{5}{3}\gamma_1^2)}. \tag{84}$$

Then, by substituting (79) and (84) into the square of (74), we can solve for δ, and we get

$$\delta = \frac{\sqrt{m_2(3\gamma_2 - 5\gamma_1^2)}}{\gamma_2 - \frac{4}{3}\gamma_1^2}. \tag{85}$$

Then, solving for μ in (72) we obtain

$$\mu = m_1 - \frac{\gamma_1 \sqrt{m_2}}{\gamma_2 - \frac{4}{3}\gamma_1^2}. \tag{86}$$

B.2 Alternative parameterizations of the IG distribution

Two different parameterizations of the IG distribution is commonly in use. Schoutens (2003) defines the distribution by the following PDF:

$$f_{IG}(x; a, b) = \frac{a}{\sqrt{2\pi x^3}} \exp(ab) \exp(-\frac{1}{2}(a^2 x^{-1} + b^2 x)) \,, \; x > 0 \tag{87}$$

which has $mean = a/b$ and $variance = a/b^3$. According to Schoutens (2003), the scaling properties of this distribution is given by

$$X \sim IG(a, b) \; \rightarrow \; tX \sim IG(\sqrt{t}a, b/\sqrt{t}) \,. \tag{88}$$

By comparing the PDF's in Equations (13) and (87) it's clear that the relation between the two parameterizations are given by

$$\lambda = a^2 \tag{89}$$

$$\mu = \frac{a}{b} \tag{90}$$

and thus we have $mean = \mu$ and $variance = \mu^3/\lambda$.

We can derive the scaling properties of this parameterization from Equation (88). Let $X \sim IG(\mu, \lambda) \sim IG(a^2, a/b)$, then $tX \sim IG(\tilde{\mu}, \tilde{\lambda})$ with:

$$\tilde{\mu} = (\sqrt{t}a)^2 = ta^2 = t\mu \tag{91}$$

and

$$\tilde{\lambda} = \frac{\sqrt{t}a}{b/\sqrt{t}} = t\frac{a}{b} = t\lambda\,. \tag{92}$$

The R library *mgcv* contains the function *rig()* which can be used to simulate from IG distributions. Note that this function assumes a parameterization with an inverse scale parameter, such that Equation (89) becomes $\lambda = a^{-2}$.

B.3 Proof of out-in parity for barrier options

Let A be the set of all processes S_t that cross the barrier B, and let $\overline{A}$ be the set of all processes that do not cross the barrier. We then have $A \cap \overline{A} = \emptyset$, and the union of A and $\overline{A}$ is the entire sample space of S_t. The value of a European call option is $C_E(S_t; T, E) = max(0, S_T - E)$. The value of a up-and-out call option is

$$C_{\mathrm{UAO}}(S_t; T, E, B) = \begin{cases} 0 & \text{if } S_t \in A \\ \\ C_E(S_t; T, E) & \text{if } S_t \notin A \end{cases} \tag{93}$$

and the value of a up-and-in call option is similarly

$$C_{\mathrm{UAI}}(S_t; T, E, B) = \begin{cases} C_E(S_t; T, E) & \text{if } S_t \in A \\ \\ 0 & \text{if } S_t \notin A \end{cases}\,. \tag{94}$$

It follows directly that

$$C_E(T, E) = C_{UAO}(T, E, B) + C_{UAI}(T, E, B)\,. \tag{95}$$

The same argument can be applied to put options.

C R code

In this section we include relevant [R] code that is developed and used to obtain results in this thesis. Some of the code relies on additional utility libraries; **mgcv** and **fBasics**.

C.1 Energy modelling

```
# Fit a NIG distribution to a
# vector of samples, using the
# Method of Moments
#
# Input:
# X  - data sample
#
estimateNIGparams = function(X)
{
  m1 = mean(X)
  m2 = var(X)
  g1 = sampleskew(X)
  g2 = samplekurtosis(X)

  c(nigalpha(m2,g1,g2),nigbeta(m2,g1,g2),
    nigdelta(m2,g1,g2),nigmu(m1,m2,g1,g2))
}

# Estimate alpha parameter of NIG
# distribution from sample moments
#
# Input:
# m2  - sample variance
# g1  - sample skewness
# g2  - sample kurtosis
#
nigalpha = function(m2,g1,g2)
{
  sqrt(3*g2 - 4*g1^2) / (sqrt(m2)*(g2-(5/3)*g1^2))
}

# Estimate beta parameter of NIG
# distribution from sample moments
```

```
#
# Input:
# m2  - sample variance
# g1  - sample skewness
# g2  - sample kurtosis
#
nigbeta = function(m2,g1,g2)
{
  g1 / (sqrt(m2)*(g2-(5/3)*g1^2))
}

# Estimate delta parameter of NIG
# distribution from sample moments
#
# Input:
# m2  - sample variance
# g1  - sample skewness
# g2  - sample kurtosis
#
nigdelta = function(m2,g1,g2)
{
  sqrt(m2*(3*g2-5*g1^2)) / (g2-(4/3)*g1^2)
}

# Estimate mu parameter of NIG
# distribution from sample moments
#
# Input:
# m1  - sample mean
# m2  - sample variance
# g1  - sample skewness
# g2  - sample kurtosis
#
nigmu = function(m1,m2,g1,g2)
{
  m1 - g1*3*sqrt(m2)/(3*g2-4*g1^2)
}

# Estimate kurtosis from data sample
#
# Input:
```

```
# X  - data sample
#
samplekurtosis = function(X)
{
  n = length(X)
  diff = X-mean(X)
  k1 = ((1/n)*sum(diff^4)) / ((1/n)*sum(diff^2))^2

  k0 = ((n-1)/((n-2)*(n-3))) * ((n+1)*k1 - 3*(n-1))
  k0
}

# Estimate skewness from data sample
#
# Input:
# X  - data sample
#
sampleskew = function(X)
{
  n = length(X)
  diff = X-mean(X)

  s1 = ((1/n)*sum(diff^3)) / ((1/n)*sum(diff^2))^(3/2)

  s0 = (sqrt(n*(n-1))*s1) / (n-2)
  s0
}
```

```
# Obtain residuals by removing auto-regressive
# effects from a time series
#
# Input:
# X     - time series
# arobj - returned object from call to ar()
#
removeAR = function(X, arobj)
{
  n=length(X)
  result = c()
  for(i in (arobj$order+1):n)
  {
```

```
    w = X[i]
    for(j in 1:arobj$order)
    {
      w = w-arobj$ar[j]*X[i-j]
    }
    result = c(result,w)
  }
  result
}
```

```
# Ordinary LS method, fitting season/trend
# function to data (e.g. log prices)
#
# Input:
# data    - the data
# init    - initial parameters for iterative minimization
# typsize - typical size of model parameters
# iterlim - maximum number of iterations
#
ordinaryLS = function(data,init=c(0,0,0,0,0,0),
                      typsize=c(5,1,0.1,0.1,0,0), iterlim=250)
{
  result = nlm(SSR,p=init,data=data,typsize=typsize,
               print.level=1, iterlim=iterlim)
  result
}

# Utility function for ordinaryLS()
# Returns the sum of squares of the residuals
# after function has been fitted
#
# Input:
# params   - input parameters to logLambda()
# data     - the data to which model is being fitted
#
SSR = function(params, data)
{
  rsq = data - logLambda(params,1:length(data))
  result <- sum(rsq^2)
  attr(result, "gradient") <- SSRgrad(params,data)
  result
```

```
}

# Utility function for SSR()
# Return gradient vector of SSR.
#
# Input:
# params   - input parameters to logLambda()
# data     - the data to which model is being fitted
#
SSRgrad = function(params,data)
{
  grad = rep(0,6)
  n = length(data)
  j=1:n

  # translate parameters for readability
  b0 = params[1]
  b1 = params[2]
  b2 = params[3]
  b3 = params[4]
  tau1 = params[5]
  tau2 = params[6]

  grad[1] = 2*sum(-data+b0+b1*cos((tau1+2*pi*j)/260) +
            b2*cos((tau2+2*pi*j)/5)+b3*j)
  grad[2] = 2*sum(cos((tau1+2*pi*j)/260) *
            (-data+b1*cos((tau1+2*pi*j)/260) +
            b0 + b3*j+b2*cos((tau2+2*pi*j)/5)))
  grad[3] = 2*sum(cos((tau2+2*pi*j)/5) *
            (-data+b1*cos((tau1+2*pi*j)/260) +
            b0 + b3*j+b2*cos((tau2+2*pi*j)/5)))
  grad[4] = 2*sum( j*(-data + b3*j + b0 +
            b1*cos((tau1+2*pi*j)/260) +
            b2*cos((tau2+2*pi*j)/5)))
  grad[5] = sum(b1* (sin((tau1+2*pi*j)/260)/130) *
            (data - b1*cos((tau1+2*pi*j)/260) -
            b0 - b3*j - b2*cos((tau2+2*pi*j)/5)))
  grad[6] = sum(b2* (sin((tau2+2*pi*j)/5)/2.5) *
            (data - b2*cos((tau2+2*pi*j)/5) - b0 -
            b3*j - b1*cos((tau1+2*pi*j)/260)))
```

```
  grad
}
```

```
# Return function values for log Lambda(t)
# i.e. deterministic model of seasonal
# and weekly variations + trend
#
# Input:
# param        - vector of model parameters
# t     - vector of times at which we
#         want to evaluate the function
#
logLambda = function(param,t)
{
  # translate variable names
  # just for readability of formula
  b0 = param[1]
  b1 = param[2]
  b2 = param[3]
  b3 = param[4]
  tau1 = param[5]
  tau2 = param[6]

  b0 + b1*cos((tau1 + 2*pi*t)/260) +
  b2*cos((tau2+2*pi*t)/5) + b3*t
}
```

C.2 Density forecast and path-independent derivatives

```
# Method for calculating input values to PI methods
# based on AR(1) model with deterministic seasons
# and NIG distributed residuals
#
# Input:
# m       -   time steps until t=T
# data    -   historical values of S_T, ending at t=0
#
setupPI = function(m,data)
{
  # estimate parameters from the complete data set
```

```
# in order to obtain starting values for next LS procedure
LStest = ordinaryLS(log(data))
startval = LStest$estimate

# fit a seasonality function to the
# m last historical log prices
LSfit = ordinaryLS(log(tail(data,m)),init=startval,
                                        typsize=startval)

# calculate residuals after deseasonalizing,
# and fit AR(1) model
detrended = log(tail(data,m))- logLambda(LSfit$estimate, 1:m)
arfit = ar(detrended,order=1)
print(paste("AR parameter",arfit$ar))
residuals = removeAR(detrended, arfit)

# estimate NIG parameters
NIGpar = estimateNIGparams(residuals)
p = ppoints(length(residuals))
quant = qnig(p,NIGpar[1],NIGpar[2],NIGpar[3],NIGpar[4])

# do k MC runs to find approximate grid size
k = 1000
MC = c()
for(i in 1:k)
{
  # simulate m NIG-residuals
  nigvar = rnig(m,NIGpar[1],NIGpar[2],NIGpar[3],NIGpar[4])

  # simulate Autoregressive process X
  X = c(tail(detrended,1))
  alpha = arfit$ar-1
  for(j in 1:m)
  {
    dX = alpha*tail(X,1) + nigvar[j]
    X = c(X,tail(X,1)+dX)
  }
  X = tail(X,m)
  MC = cbind(MC,X)
}
```

```
  # setup PDF grid (expand range by 100% to be sure)
  gridmin = min(MC) - 0.5*(max(MC)-min(MC))
  gridmax = max(MC) + 0.5*(max(MC)-min(MC))

  # compile setup object
  setup = NULL
  setup$X0 = tail(detrended,1)
  setup$range = range(MC)
  setup$gridrange = c(gridmin,gridmax)
  setup$seasonpar = LSfit$estimate
  setup$arpar = arfit$ar
  setup$nigpar = NIGpar
  setup$m = m
  # return setup object
  setup
}
```

```
# Use the PI approach to approximate PDF of log(S_T)
#
#
# Arguments:
# m         -   time steps until t=T
# data      -   historical values of S_T, ending at t=0
# setup     -   output from the setupPI function
# N         -   number of gridpoints for representing PDF
# K         -   number of gridpoints in the integration grid
# zero      -   any density less than this is set to 0
#
PIdensity = function(m,data,setup=NULL, N=100, K=1000,
                     zero=10^-6)
{
  ptm = proc.time()

  # we need K to be an odd number i.o. to use Simpsons method
  if(K%%2==0) K = K+1

  # create setup if not submitted as argument
  if(is.null(setup))
    setup = setupPI(m,data)

  # setup PDF grid (expand range by 100% to be sure)
```

```
gridmin = setup$gridrange[1]
gridmax = setup$gridrange[2]
grid = seq(gridmin,gridmax,length.out=N)

# calculate and store values of propagator: p(x|x')
# xMat contains x' values
# propMat contains corresponding p(x|x')
# x is given by grid[row index]
propMat = c()
xMat = c()
# test run to find location of probability mass
tempMat = c()
for(i in 1:N)
{
  Xval = grid[i]   # value of x for this col
  condpdf = dnig(grid, setup$nigpar[1], setup$nigpar[2],
            setup$nigpar[3],
            (setup$nigpar[4]+setup$arpar*Xval))
  tempMat = cbind(tempMat, condpdf)
}
# find grid and propagator densities for each x
for(i in 1:N)
{
  # find limits and grid for this propagator row
  # to avoid calculations where propagator is zero
  limits = range(grid[which(tempMat[i,]>zero)])
  xMat = rbind(xMat,seq(limits[1],limits[2],length.out=K))

  # find densities for this propagator row
  row = c()
  for(j in 1:K)
  {
    row = c(row, dnig(grid[i], setup$nigpar[1],
            setup$nigpar[2], setup$nigpar[3],
            (setup$nigpar[4]+setup$arpar*xMat[i,j])))
  }
  propMat = rbind(propMat,row)
}

# find distribution at t=1
X0 = log(tail(data,1))-logLambda(setup$seasonpar, m)
```

```
pdf = c(0,dnig(grid[2:(N-1)], setup$nigpar[1],
        setup$nigpar[2], setup$nigpar[3],
        (setup$nigpar[4] + setup$arpar*X0)),0)
pdfspline = splinefun(grid, pdf,method="natural")

print("starting iteration forward in time")
# iterate through time steps
for(t in 2:m)
{
  # iterate through grid points
  newpdf = c(0)
  for(i in 2:(N-1))
  {
    # perform integration
    dens = 0
    h = diff(xMat[i,1:2])
    integrand = propMat[i,] * pdfspline(xMat[i,])
    for(j in seq(1,K-2,2))
    {
      simpson = (h/3) * (integrand[j] +
                  4*integrand[j+1] + integrand[j+2])
      dens = dens + simpson
    }
    newpdf = c(newpdf,dens)
  }
  newpdf = c(newpdf,0)
  pdfspline = splinefun(grid,newpdf,method="natural")

  if(t%%30==0) print(paste("Progress:",t))
}

# find probability mass of spline
mass = integrate(pdfspline,gridmin,gridmax)

# construct and return result
result = NULL
result$time = proc.time() - ptm
result$setup = setup
result$density = (function(x){pdfspline(x)/mass$val})
result$range = setup$gridrange
result
```

```
}
```

```
# Approximate probability density by cell mapping
#
# Input:
# setup   - object containing model parameters
#           e.g. result from call to setupPI()
# N       - number of points in grid
#
CellMapping = function(setup, N=100)
{
  ptm = proc.time()

  # setup PDF grid (expand range by 100% to be sure)
  gridmin = setup$gridrange[1]
  gridmax = setup$gridrange[2]
  grid = seq(gridmin,gridmax,length.out=N)

  # construct matrix of probability densities
  P = c()
  for(i in 1:N)
  {
    # value of X at grid point i
    Xval = grid[i]
    # get conditional densities for column i,
    # normalize and add to matrix
    condpdf = dnig(grid, setup$nigpar[1], setup$nigpar[2],
                   setup$nigpar[3], (setup$nigpar[4]+
                   setup$arpar*Xval))
    condpdf = condpdf/(sum(condpdf))
    P = cbind(P, condpdf)
  }

  # probability density vector at t=1
  p1 = dnig(grid, setup$nigpar[1], setup$nigpar[2],
            setup$nigpar[3], (setup$nigpar[4]+
            setup$arpar*setup$X0))
  g1 = p1/sum(p1)

  # propagate probability forward to t=m
  gm = powmat(P,setup$m-1) %*% g1
```

```
  pm = powmat(P,setup$m-1) %*% p1
  # set densities to zero at each end, and create spline
  pm[0]=pm[N]=0
  pspl = splinefun(grid,pm,method="natural")

  # find probability mass of pspl
  mass = integrate(pspl,gridmin,gridmax)

  # construct and return result object
  result = NULL
  result$time = proc.time()-ptm
  result$setup = setup
  result$grid = grid
  result$gm = gm
  result$density = (function(x){pspl(x)/mass$val})
  result$P = P
  result
}


# Utility function for CellMapping()
# Finds the n'th power of a matrix.
#
# M   - the matrix
# n   - the power
powmat = function(M,n)
{
  result = diag(1, ncol(M))
  while(n>0)
  {
    if(n %% 2 != 0)
    {
      result = result %*% M
      n = n-1
      if(n==0) break
    }
    M = M %*% M
    n = n/2
  }
  result
```

```
}
```

```
# Run Monte Carlo simulations to obtain density estimate
#
# Input:
# setup   - object that contains model parameters
# n       - number of MC runs
#
MonteCarloDensity = function(setup,n=10^6)
{
  ptm <- proc.time()
  mc = rep(0,n)
  for(i in 1:n)
  {
    # simulate m NIG-residuals
    nigvar = rnig(setup$m,setup$nigpar[1],setup$nigpar[2],
                  setup$nigpar[3],setup$nigpar[4])

    # simulate Autoregressive process X
    X = setup$X0
    alpha = setup$arpar-1
    for(j in 1:setup$m)
    {
      dX = alpha*X + nigvar[j]
      X = X+dX
    }
    mc[i] = X
  }
  timer=proc.time()-ptm

  # construct and return result
  result = NULL
  result$n = n
  result$mc = mc
  result$time=timer
  result$setup = setup
  result
}
```

```
# Return value of European call option
# calculated from density at expiry
```

```
#
# Input:
# PIobj - returned
# E     - exercise price
#
valueEuropeanCall = function(PIobj, E)
{
  # deterministic mean of log price
  logLambda = logLambda(PIobj$setup$seasonpar, 2*PIobj$setup$m)

  # limits of integration for log S density
  limits = PIobj$setup$gridrange + logLambda

  # check if exercise price is small enough
  if(log(E)>limits[2])
    value = 0
  else
  {
    # get density function for log S
    logdens = logpriceDensity(PIobj)

    # create integrand function
    integrand = (function(lnS){(exp(lnS)-E)*logdens(lnS)})

    # perform integration
    value = integrate(integrand,max(log(E),limits[1]),
                      limits[2])$val
  }

  value
}
```

```
# Return value of European put option
# calculated from density at expiry
#
# Input:
# PIobj - returned
# E     - exercise price
#
valueEuropeanPut = function(PIobj, E)
{
```

```
  # deterministic mean of log price
  logLambda = logLambda(PIobj$setup$seasonpar, 2*PIobj$setup$m)

  # limits of integration for log S density
  limits = PIobj$setup$gridrange + logLambda

  # check if exercise price is small enough
  if(log(E)<limits[1])
    value = 0
  else
  {
    # get density function for log S
    logdens = logpriceDensity(PIobj)

    # create integrand function
    integrand = (function(lnS){(E-exp(lnS))*logdens(lnS)})

    # perform integration
    value = integrate(integrand,limits[1],
                      min(log(E),limits[2]))$val
  }

  value
}
```

```
# Return value of forward contract
# calculated from density at maturity
#
# Input:
# PIobj - returned PI object
# F     - forward price
#
valueForward = function(PIobj, F)
{
  # deterministic mean of log price
  logLambda = logLambda(PIobj$setup$seasonpar, 2*PIobj$setup$m)

  # limits of integration for log S density
  limits = PIobj$setup$gridrange + logLambda

  # get density function for log S
```

```
  logdens = logpriceDensity(PIobj)

  # create integrand function
  integrand = (function(lnS){(exp(lnS)-F)*logdens(lnS)})

  # perform integration
  value = integrate(integrand,limits[1], limits[2])$val

  value
}
```

```
# Return log price density function.
# Utility function for valuation of
# derivatives.
#
# Input:
# PIobj     - PI object
# (returned from CellMapping or PIdens)
#
logpriceDensity = function(PIobj)
{
  (function(lnS)
  {
    dens = rep(0,length(lnS))
    x = lnS - logLambda(PIobj$setup$seasonpar,2*PIobj$setup$m)
    notzero = which(x>PIobj$setup$gridrange[1] &
                    x<PIobj$setup$gridrange[2])
    dens[notzero] = PIobj$density(x[notzero])
    dens
  })
}
```

```
#  Value European call option from MC results
#
#  Input:
#  MCobj  - result from MonteCarloDensity-function
#  E      - exercise price
#
mcEuropeanCall = function(MCobj,E)
{
```

```
  logLambda = logLambda(MCobj$setup$seasonpar, 2*MCobj$setup$m)
  values = exp(MCobj$mc + logLambda)-E
  values[which(values<0)]=0
  mean(values)
}
```

```
#  Value European put option from MC results
#
#  Input:
#  MCobj  - result from MonteCarloDensity-function
#  E      - exercise price
#
mcEuropeanPut = function(MCobj,E)
{
  logLambda = logLambda(MCobj$setup$seasonpar, 2*MCobj$setup$m)
  values = E-exp(MCobj$mc + logLambda)
  values[which(values<0)]=0
  mean(values)
}
```

```
#  Value forward contract from MC results
#
#  Input:
#  MCobj  - result from MonteCarloDensity-function
#  F      - forward price
#
mcForward = function(MCobj,F)
{
  logLambda = logLambda(MCobj$setup$seasonpar, 2*MCobj$setup$m)
  values = exp(MCobj$mc + logLambda)-F
  mean(values)
}
```

C.3 Barrier options

```
# Calculate density of Up-and-out barrier
# process, by Path Integrals propagated
# with Simpson's Rule
#
# Model: Deterministic + AR(1)
```

```
#
# Input
# m         - number of days until expiry
# data      - the data from which we estimate model params
# B         - the upper barrier
# N         - number of points in density representation grid
# K         - number if points in integration grid
# setup     - obj returned from setupPI-function
# zero      - value at which the propagator function is
#             considered zero
# bgwidth   - width of barrier grid, as a fraction of
#             gridspacing
# bgpoints  - number of points in the grid around barrier
# barrierGrid - boolean, true if we should use fine grid around
#               barrier
#
UAObarrierDens = function(m, data, B, N=100, K=1000,
                          setup=NULL, zero=10^-6, bgwidth=5,
                          bgpoints=20, barrierGrid=TRUE)
{
  ptm <- proc.time()

  # check for input errors
  if(tail(data,1)>B)
    stop("Value out of range: B must be larger than S_0")

  # retrieve regular PI setup
  if(is.null(setup))
    setup = setupPI(m,data)

  # get seasonality function for t=1:m
  logLambda = logLambda(setup$seasonpar, (m+1):(2*m))
  # calculate log barrier and x-barrier
  logB = log(B)
  XB   = logB-logLambda

  # setup grid for h_m(x)
  # let gridmax be slightly above largest barrier
  gridmin = setup$gridrange[1]
  gridspace = (max(XB)-gridmin)/(N-2)
  gridmax = max(XB)+gridspace
```

```
# expand grid up to maximum barrier position
grid = seq(gridmin,gridmax,length.out=N)

# calculate and store values of propagator: p(x|x')
# xMat contains x' values
# propMat contains corresponding p(x|x')
# x is given by grid[row index]
propMat = c()
tempMat = c()
xMat = c()
# test run to find location of probability mass
for(i in 1:N)
{
  Xval = grid[i]   # value of x for this col
  condpdf = dnig(grid, setup$nigpar[1], setup$nigpar[2],
                 setup$nigpar[3], (setup$nigpar[4]+
                 setup$arpar*Xval))
  tempMat = cbind(tempMat, condpdf)
}
# find integration grid and propagator
# densities for each x in grid
for(i in 1:N)
{
  # find limits and grid for this propagator row
  # to avoid calculations where propagator is zero
  limits = range(grid[which(tempMat[i,]>zero)])
  xMat = rbind(xMat,seq(limits[1],limits[2],length.out=K))

  # find densities for this propagator row
  row = c()
  for(j in 1:K)
  {
    row = c(row, dnig(grid[i], setup$nigpar[1],
            setup$nigpar[2],
            setup$nigpar[3], (setup$nigpar[4]+
            setup$arpar*xMat[i,j])))
  }
  propMat = rbind(propMat,row)
}

# find distribution at t=1
```

```
# (do calculations in deseasonalized space)
X0 = log(tail(data,1))-logLambda(setup$seasonpar, m)
pdf = c(0,dnig(grid[2:(N-1)], setup$nigpar[1],
               setup$nigpar[2], setup$nigpar[3],
               (setup$nigpar[4] + setup$arpar*X0)),0)
spline = splinefun(grid, pdf,method="natural")
# find H-density function
hj = knockOutHdens(spline,setup$gridrange,XB[1])

print("starting iteration forward in time")
# iterate through time steps
for(t in 2:m)
{
  # iterate through grid points
  newpdf = c(0)
  for(i in 2:(N-1))
  {
    # perform integration
    dens = 0
    h = diff(xMat[i,1:2])
    integrand = propMat[i,] * hj(xMat[i,])
    for(j in seq(1,K-2,2))
    {
      simpson = (h/3) * (integrand[j] + 4*integrand[j+1] +
                         integrand[j+2])
      dens = dens + simpson
    }
    newpdf = c(newpdf,dens)
  }
  newpdf = c(newpdf,0)


  #-----------------------------------------------------------
  # Part where we find a fine grid around barrier and
  # calculate pdf in those points
  #-----------------------------------------------------------

  if(barrierGrid)
  {
    # Find pdf in the close vincinity of the barrier
    # barrier grid
```

```
bpdf = c()
bgrid = seq(XB[t-1]*setup$arpar-0.5*gridspace*bgwidth,
            XB[t-1]*setup$arpar+0.5*gridspace*bgwidth,
            length.out=bgpoints)

# calculate and store values of propagator: p(x|x')
# bxMat contains x' values
# bpropMat contains corresponding p(x|x')
# x is given by grid[row index]
bpropMat = c()
btempMat = c()
bxMat = c()

# test run to find location of probability mass
for(i in 1:N)
{
  Xval = grid[i]   # value of x' for this col
  condpdf = dnig(bgrid, setup$nigpar[1], setup$nigpar[2],
                 setup$nigpar[3],
                 (setup$nigpar[4]+setup$arpar*Xval))
  btempMat = cbind(btempMat, condpdf)
}
# find grid and propagator densities for each x
for(i in 1:length(bgrid))
{
  # find limits and grid for this propagator row
  # to avoid calculations where propagator is zero
  limits = range(grid[which(btempMat[i,]>zero)])
  bxMat = rbind(bxMat,seq(limits[1],limits[2],
                length.out=K))

  # find densities for this propagator row
  row = c()
  for(j in 1:K)
  {
    row = c(row, dnig(bgrid[i], setup$nigpar[1],
                      setup$nigpar[2], setup$nigpar[3],
                      (setup$nigpar[4]+
                       setup$arpar*bxMat[i,j])))
  }
  bpropMat = rbind(bpropMat,row)
```

```
    }

    # iterate through grid points
    bpdf = c()
    for(i in 1:length(bgrid))
    {
      # perform integration
      dens = 0
      h = diff(bxMat[i,1:2])
      integrand = bpropMat[i,] * hj(bxMat[i,])
      for(j in seq(1,K-2,2))
      {
        simpson = (h/3) * (integrand[j] + 4*integrand[j+1] +
                           integrand[j+2])
        dens = dens + simpson
      }
      bpdf = c(bpdf,dens)
    }
    spline = splinefun(c(grid,bgrid),c(newpdf,bpdf),
                       method="natural")

  }
  else
    spline = splinefun(grid,newpdf,method="natural")

  #-----------------------------------------------------------

  #find H-density function; h_j
  hj = knockOutHdens(spline,setup$gridrange,XB[t])

}

# construct and return result obj
result=NULL
result$time = proc.time()-ptm
if(barrierGrid)
{
  result$bgrid = bgrid
  result$bpdf = bpdf
}
result$newpdf = newpdf
```

```
  result$B = B
  result$grid = grid
  result$spline = spline
  result$density = hj
  result$setup = setup
  result
}
```

```
# Return knock-out H-density function.
# Utility function for barrier PI functions.
#
# Input:
# dens      - density function
# limits    - interval where all mass is located
# B         - the barrier (in appropriate scale)
#
knockOutHdens = function(dens,limits,B)
{
  (function(x){
    val = rep(0,length(x))
    notzero = which(x>limits[1] & x<=B)
    val[notzero] = dens(x[notzero])
    val
  })
}
```

```
# Return density function that is
# zero below the barrier.
# Utility function for UAIbarrierDens()
#
# Input:
# dens      - density function
# limits    - interval where all mass is located
# B         - the barrier (in appropriate scale)
#
barOverflow = function(dens,limits,B)
{
  (function(x){
    val = rep(0,length(x))
    notzero = which(x<limits[2] & x>=B)
```

```
    val[notzero] = dens(x[notzero])
    val
  })
}
```

```
#  Estimate values of up-and-out barrier
#  call options by repeated MC runs
#
#  Input:
#  setup    -  setup object from setupPI()
#  E        -  exercise price of option
#  B        -  barrier price value
#  n        -  number of MC runs
#
mcUAObarrier = function(setup, E, B, n=10^6)
{
  ptm <- proc.time()

  # get seasonality function for t=1:m
  logLambda = logLambda(setup$seasonpar,
                        (setup$m+1):(2*setup$m))
  # calculate deseasonalized x-barrier
  Xbar   = log(B)-logLambda

  # create vector for X-values
  Xvals = rep(NA,n)

  for(i in 1:n)
  {
    # simulate m NIG-residuals
    nigvar = rnig(setup$m,setup$nigpar[1],setup$nigpar[2],
                  setup$nigpar[3],setup$nigpar[4])

    # simulate Autoregressive process X
    X = setup$X0
    alpha = setup$arpar-1
    brokebarrier = FALSE
    for(j in 1:setup$m)
    {
      dX = alpha*X + nigvar[j]
      X = X+dX
```

```
      if(X>Xbar[j])
      {
        brokebarrier = TRUE
        break
      }
    }
    if(brokebarrier==FALSE)
      Xvals[i] = X
  }

  # option values
  values = exp(Xvals+tail(logLambda,1))-E
  call = values[which(values>0)]
  put = -values[which(values<0)]

  # construct and return result obj
  result = NULL
  result$time = proc.time()-ptm
  result$n = n
  result$setup = setup
  result$lambdatail = tail(logLambda,1)
  result$forward = values
  result$call = call
  result$put = put
  result$callval = sum(call)/n
  result$putval = sum(put)/n
  result$Xvals = Xvals
  result$B = B
  result$E = E
  result
}
```

```
#  Estimate values of up-and-in barrier
#  options by repeated MC runs.
#
#  Input:
#  setup    -  setup object from setupPI()
#  E        -  exercise price of option
#  B        -  barrier value
#  n        -  number of MC runs
#
```

```
mcUAIbarrier = function(setup, E, B, n=10^6)
{
  ptm <- proc.time()

  # get seasonality function for t=1:m
  logLambda = logLambda(setup$seasonpar,
                        (setup$m+1):(2*setup$m))
  # calculate deseasonalized x-barrier
  Xbar   = log(B)-logLambda

  # create  for X-values
  Xvals = rep(NA,n)

  for(i in 1:n)
  {
    # simulate m NIG-residuals
    nigvar = rnig(setup$m,setup$nigpar[1],setup$nigpar[2],
                  setup$nigpar[3],setup$nigpar[4])

    # simulate Autoregressive process X
    X = setup$X0
    alpha = setup$arpar-1
    brokebarrier = FALSE
    for(j in 1:setup$m)
    {
      dX = alpha*X + nigvar[j]
      X = X+dX
      if(X>Xbar[j])
        brokebarrier = TRUE
    }
    if(brokebarrier)
    {
      Xvals[i] = X
    }
  }

  # option values
  values = exp(Xvals+tail(logLambda,1))-E
  call = values[which(values>0)]
  put = -values[which(values<0)]
```

```
  # construct and return result obj
  result = NULL
  result$time = proc.time()-ptm
  result$n = n
  result$setup = setup
  result$call = call
  result$put = put
  result$callval = sum(call)/n
  result$putval = sum(put)/n
  result$Xvals = Xvals
  result$B = B
  result$E = E
  result
}
```

```
# Calculate density of Up-and-in barrier
# process, by Path Integrals propagated
# with Simpson's Rule
#
# Model: Deterministic + AR(1)
#
# Input
# m         - number of days until expiry
# data      - the data from which we estimate model params
# B         - the upper barrier
# N         - number of points in density representation grid
# K         - number if points in integration grid
# setup     - obj returned from setupPI-function
# zero      - value at which the propagator function is
#             considered zero
# bgwidth   - width of barrier grid, as a fraction of
#             gridspacing
# bgpoints  - number of points in the grid around barrier
# barrierGrid - boolean, true if we should use fine grid
#               around barrier
#
UAIbarrierDens = function(m, data, B, N=100, K=1000,
                          setup=NULL, zero=10^-6, bgwidth=5,
                          bgpoints=20, barrierGrid=TRUE)
{
  ptm <- proc.time()
```

```
# check for input errors
if(tail(data,1)>B)
  stop("Value out of range: B must be larger than S_0")

# retrieve regular PI setup
if(is.null(setup))
  setup = setupPI(m,data)

# get seasonality function for t=1:m
logLambda = logLambda(setup$seasonpar, (m+1):(2*m))
# calculate log barrier and x-barrier
logB = log(B)
XB   = logB-logLambda
#print(XB)

# setup grid for h_m(x) and g_m(x)
gridmin = setup$gridrange[1]
gridmax = setup$gridrange[2]
gridspace = (gridmax-gridmin)/(N-1)
# expand grid
grid = seq(gridmin,gridmax,length.out=N)

# calculate and store values of propagator: p(x|x')
# xMat contains x' values
# propMat contains corresponding p(x|x')
# x is given by grid[row index]
propMat = c()
tempMat = c()
xMat = c()
# test run to find location of probability mass
for(i in 1:N)
{
  Xval = grid[i]   # value of x for this col
  condpdf = dnig(grid, setup$nigpar[1], setup$nigpar[2],
                 setup$nigpar[3], (setup$nigpar[4]+
                 setup$arpar*Xval))
  tempMat = cbind(tempMat, condpdf)
}
# find integration grid and propagator
# densities for each x in grid
```

```
for(i in 1:N)
{
  # find limits and grid for this propagator row
  # to avoid calculations where propagator is zero
  limits = range(grid[which(tempMat[i,]>zero)])
  #print(limits[2]-limits[1])
  xMat = rbind(xMat,seq(limits[1],limits[2],length.out=K))

  # find densities for this propagator row
  row = c()
  for(j in 1:K)
  {
    row = c(row, dnig(grid[i], setup$nigpar[1],
            setup$nigpar[2], setup$nigpar[3],
            (setup$nigpar[4]+setup$arpar*xMat[i,j])))
  }
  propMat = rbind(propMat,row)
}

# find distribution at t=1
# (do calculations in deseasonalized space)
X0 = log(tail(data,1))-logLambda(setup$seasonpar, m)
pdf = c(0,dnig(grid[2:(N-1)], setup$nigpar[1],
               setup$nigpar[2], setup$nigpar[3],
               (setup$nigpar[4] + setup$arpar*X0)),0)
pdfspline = splinefun(grid, pdf,method="natural")
# find G-density function (=knock-out H-density)
gj = knockOutHdens(pdfspline,setup$gridrange,XB[1])
# find H-density function
hj = barOverflow(pdfspline,setup$gridrange,XB[1])

print("starting iteration forward in time")
# iterate through time steps
for(t in 2:m)
{
  # iterate through grid points
  h_pdf = c(0)
  g_pdf = c(0)

  for(i in 2:(N-1))
  {
```

```
  # perform integration
  dens = c(0,0)  # = c(h,g)
  # distance between integration grid points
  h = diff(xMat[i,1:2])

  h_integrand = propMat[i,] * hj(xMat[i,])
  g_integrand = propMat[i,] * gj(xMat[i,])
  for(j in seq(1,K-2,2))
  {
    simpson = (h/3) * (h_integrand[j] +
              4*h_integrand[j+1] + h_integrand[j+2])
    dens[1] = dens[1] + simpson
    simpson = (h/3) * (g_integrand[j] +
              4*g_integrand[j+1] + g_integrand[j+2])
    dens[2] = dens[2] + simpson
  }
  h_pdf = c(h_pdf,dens[1])
  g_pdf = c(g_pdf,dens[2])
}
h_pdf = c(h_pdf,0)
g_pdf = c(g_pdf,0)



#-------------------------------------------------------
# Part where we find a fine grid around barrier and
# calculate pdf in those points
#-------------------------------------------------------

if(barrierGrid)
{
  # Find pdf in the close vincinity of the barrier
  # barrier grid
  bpdf = c()
  bgrid = seq(XB[t-1]*setup$arpar-0.5*gridspace*bgwidth,
              XB[t-1]*setup$arpar+0.5*gridspace*bgwidth,
              length.out=bgpoints)

  # calculate and store values of propagator: p(x|x')
  # bxMat contains x' values
  # bpropMat contains corresponding p(x|x')
```

```
# x is given by grid[row index]
bpropMat = c()
btempMat = c()
bxMat = c()

# test run to find location of probability mass
for(i in 1:N)
{
  Xval = grid[i]   # value of x' for this col
  condpdf = dnig(bgrid, setup$nigpar[1], setup$nigpar[2],
                 setup$nigpar[3],
                 (setup$nigpar[4]+setup$arpar*Xval))
  btempMat = cbind(btempMat, condpdf)
}
# find grid and propagator densities for each x
for(i in 1:length(bgrid))
{
  # find limits and grid for this propagator row
  # to avoid calculations where propagator is zero
  limits = range(grid[which(btempMat[i,]>zero)])
  bxMat = rbind(bxMat,seq(limits[1],limits[2],
                length.out=K))

  # find densities for this propagator row
  row = c()
  for(j in 1:K)
  {
    row = c(row, dnig(bgrid[i], setup$nigpar[1],
                      setup$nigpar[2], setup$nigpar[3],
                      (setup$nigpar[4]+
                      setup$arpar*bxMat[i,j])))
  }
  bpropMat = rbind(bpropMat,row)
}

# iterate through grid points
h_bpdf = c()
g_bpdf = c()
for(i in 1:length(bgrid))
{
  # perform integration
```

```
    dens = c(0,0) # c(h,g)
    # distance between integration grid points
    h = diff(bxMat[i,1:2])

    h_integrand = bpropMat[i,] * hj(bxMat[i,])
    g_integrand = bpropMat[i,] * gj(bxMat[i,])

    for(j in seq(1,K-2,2))
    {
      simpson = (h/3) * (h_integrand[j] +
                4*h_integrand[j+1] + h_integrand[j+2])
      dens[1] = dens[1] + simpson
      simpson = (h/3) * (g_integrand[j] +
                4*g_integrand[j+1] + g_integrand[j+2])
      dens[2] = dens[2] + simpson
    }
    h_bpdf = c(h_bpdf,dens[1])
    g_bpdf = c(g_bpdf,dens[2])
  }
  h_spline = splinefun(c(grid,bgrid),c(h_pdf,h_bpdf),
                       method="natural")
  g_spline = splinefun(c(grid,bgrid),c(g_pdf,g_bpdf),
                       method="natural")
}
else
{
  h_spline = splinefun(grid,h_pdf,method="natural")
  g_spline = splinefun(grid,g_pdf,method="natural")
}

#-------------------------------------------------------------


#find H-density function
hj =  (function(x){
        val = rep(0,length(x))
        aboveBar = which(x<gridmax & x>XB[t])
        val[aboveBar] = h_spline(x[aboveBar]) +
                        g_spline(x[aboveBar])
        belowBar = which(x>gridmin & x<XB[t])
        val[belowBar] = h_spline(x[belowBar])
```

```
            val
          })
    #find G-density function
    gj = knockOutHdens(g_spline,setup$gridrange,XB[t])
  }

  totalmass = integrate(gj,gridmin,XB[t])$val +
              integrate(hj,gridmin,XB[t])$val +
              integrate(hj,XB[t],gridmax)$val

  # construct and return result obj
  result=NULL
  result$time = proc.time()-ptm
  if(barrierGrid)
  {
    result$bgrid = bgrid
    result$g_bpdf = g_bpdf
    result$h_bpdf = h_bpdf
  }
  result$g_pdf = g_pdf
  result$h_pdf = h_pdf
  result$grid = grid
  result$g_spline = g_spline
  result$h_spline = h_spline
  result$totalmass = totalmass
  # since we have calculated the full probability density
  # i.e. hj + gj
  # we can correct the densities such that p(Omega)=1
  result$density = (function(x){hj(x)/totalmass})
  result$gdensity = (function(x){gj(x)/totalmass})
  result$setup = setup
  result
}
```

References

Abramowitz, M. and Stegun, I. (1968). *Handbook of Mathematical Functions*. New York: Dover.

Applebaum, D. (2004). Lévy processes—from probability to finance and quantum groups. *Notices of the American Mathematical Society*, 51(11):1336–1347.

Barndorff-Nielsen, O. E. (1995). Normal inverse gaussian distributions and the modeling of stock returns. *Research Report no. 300*.

Barndorff-Nielsen, O. E. (1997). Processes of normal inverse gaussian type. *Finance and Stochastics*, 2:41–68. 10.1007/s007800050032.

Benth, F. E., Benth, J. S., and Koekebakker, S. (2008). *Stochastic modelling of electricity and related markets*. World Scientific Publishing Co. Pte. Ltd., 1st edition.

Black, F. and Scholes, M. (1973). The pricing of options and corporate liabilities. *The Journal of Political Economy*, 81(3):pp. 637–654.

Burden, R. L. and Faires, J. D. (2005). *Numerical Analysis*. Thomson Brooks/Cole, 8th edition.

Byström, H. N. E. (2005). Extreme value theory and extremely large electricity price changes. *International Review of Economics & Finance*, 14(1):41 – 55.

Cartea, A. and Figueroa, M. G. (2005). Pricing in electricity markets: a mean reverting jump diffusion model with seasonality. *Applied Mathematical Finance*, 12(4):313–335.

Eberlein, E. and Keller, U. (1995). Hyperbolic distributions in finance. *Bernoulli*, 1(3):281–299.

Eberlein, E. and Prause, K. (1998). The generalized hyperbolic model: Financial derivatives and risk measures. *FDM Preprint 56, University of Freiburg.*

Fama, E. F. (1965). The behavior of stock-market prices. *The Journal of Business*, 38(1):pp. 34–105.

Fama, E. F. (1970). Efficient capital markets: A review of theory and empirical work. *The Journal of Finance*, 25(2):pp. 383–417.

Johannes, M. S., Polson, N. G., and Stroud, J. R. (2009). Optimal filtering of jump diffusions: Extracting latent states from asset prices. *The Review of Financial Studies*, 22(7).

Karlis, D. (2002). An em type algorithm for maximum likelihood estimation of the normal-inverse gaussian distribution. *Statistics & Probability Letters*, 57(1):43 – 52.

Klüppelberg, C., Meyer-Brandis, T., and Schmidt, A. (2010). Electricity spot price modelling with a view towards extreme spike risk. *Quantitative Finance*, 10(9):963–974.

Koopman, S. J., Ooms, M., and Carnero, M. A. (2007). Periodic seasonal reg-arfima–garch models for daily electricity spot prices. *Journal of the American Statistical Association*, 102(477):16–27.

Linetsky, V. (1997). The path integral approach to financial modeling and options pricing. *Computational Economics*, 11:129–163. 10.1023/A:1008658226761.

Lucia, J. J. and Schwartz, E. S. (2002). Electricity prices and power derivatives: Evidence from the nordic power exchange. *Review of Derivatives Research*, 5:5–50. 10.1023/A:1013846631785.

Madan, D. B. and Seneta, E. (1990). The variance gamma (v.g.) model for share market returns. *The Journal of Business*, 63(4):511–524.

Næss, A. (2001). Lecture notes on the numerical solution of stochastic differential equations by path integration methods. *Technical report, Department of Mathematical Sciences, Norwegian university of Science and Technology, NO-7491, Trondheim, Norway.*

Næss, A., Aukrust, E., and Westgaard, S. (2010). Pricing of barrier options under a nig market model using numerical path integration. *Working paper no. N1-2010.*

Rydberg, T. H. (1997). The normal inverse gaussian lévy process: simulation and approximation. *Communications in Statistics – Stochastic Models*, 13(4):887–910.

Samuelson, P. A. (1965). Proof that properly anticipated prices fluctuate randomly. *Industrial Management review*, 6(2).

Schoutens, W. (2003). *Lévy Processes in Finance.* Wiley, 1st edition.

Shumway, R. H. and Stoffer, D. S. (2000). *Time Series Analysis and Its Applications*. Springer, 1st edition.

Skaug, C. and Naess, A. (2005). Pricing of asian options by numerical path intergration. *Internal Report*.

Skaug, C. and Naess, A. (2007). Fast and accurate pricing of discretely monitored barrier options by numerical path integration. *Computational Economics*, 30:143–151. 10.1007/s10614-007-9091-5.

Weron, R., Bierbrauer, M., and Trück, S. (2004). Modeling electricity prices: jump diffusion and regime switching. *Physica A: Statistical and Theoretical Physics*, 336(1-2):39 – 48. Proceedings of the XVIII Max Born Symposium @'Statistical Physics outside Physics@'.

Wilmott, P., Howison, S., and Dewynne, J. (1995). *The Mathematics of Financial Derivatives*. Cambridge University Press, 2009 edition.