May 2018

Calibration of a Stochastic Price Model for American Electricity Markets

Oliver G. Meister
University of Wisconsin-Milwaukee

Calibration of a Stochastic Price Model for American Electricity Markets

by

Oliver G. Meister

A Thesis Submitted in
Partial Fulfillment of the
Requirements for the Degree of

Master of Science
in
Mathematics

at

The University of Wisconsin-Milwaukee
May 2018

ABSTRACT

Calibration of a Stochastic Price Model for American Electricity Markets

by

Oliver G. Meister

The University of Wisconsin-Milwaukee, 2018
Under the Supervision of Professor Richard H. Stockbridge

Abstract

This thesis discusses models for electricity spot prices from the Midwestern American and Manitoba market. The models are based on experiences in European markets and rely on a superposition model with several jump components. The methodology of Bayesian Inference solved with a Markov chain Monte Carlo algorithm has been applied to find estimators for the processes of the model. The specific Markov chain Monte Carlo algorithm applied a Random Walk Metropolis combined with a Gibbs sampler. The different estimators of the models are evaluated with the posterior predictive value and simulations of the electricity spot prices. We have modified this methodology to apply to the US market.

Thank You.

Professor Richard Stockbridge for the advice and support you offered during the last semester.
Charles Beer offering your feedback from reading this thesis and guiding me the last weeks.
Professors Gabriella Pinter and Chao Zhu for serving on my committee and for your ideas and feedback.
Professor Gerhard Dikta for the first year of my Masters's level and your support applying for the Dual Degree Program.

Oliver G. Meister

Thank you!

Table of Contents

List of Figures

List of Tables

Introduction

Electricity spot prices are influenced by many diverse drivers, for example renewable power sources or changes on the world market such as the fiscal crisis. The number and variety of different factors that influence the behavior of the electricity prices complicate a calibration or prediction of the change in the market. The structure of the price process consists of diverse components such as different jumps, general trading behavior and several kinds of volatility. The prediction of electricity spot prices by calibrating models based on observations has been documented in several papers. One specific paper describes an approach of defining a model which is based on daily average prices from APXUK and EEX, two European markets. In the paper, an algorithm is described to calibrate a model using Bayesian inference. Our goal is to set a model for the daily electricity spot prices of the Midwestern American and Manitoba market by using the results of the models that have been calibrated through the European data. We are using LMP five minutes data as a base which requires some modification of the given algorithm of the related paper since that paper uses daily average price data. Therefore, in a first step, we are comparing and assessing the similarity of the European and American data. Main factors to look at will be the structure of the data. The European data show a high volatility with different kinds of jumps, a certain reversion to the mean level and different spikes that occur frequently. This structure is a main reason for the approach of solving the model, so we have to confirm a similar structure in our data. After that, we take a look at the methodology for calibrating a model. The algorithm used here is based on a Random Walk Metropolis with a Gibbs Sampler. This method is a common algorithm for solving a Bayesian Inference by applying Markov chain Monte Carlo. We will

explain the different techniques and put their relations together to show how the algorithm, used here, is derived. The analysis will be a key factor for adjusting the model and the algorithm which is stated in the paper. We will adjust the code to calibrate our model and find adequate estimators to simulate the spot prices of the MISO market, specifically the WEC node used by WE Energies. The resulting models will be evaluated by comparing the final estimators and calculating posterior predictive values for our different processes of the model. At the end we assess our results by simulating electricity spot prices. The resulting price processes paths will be compared with our observed prices to show the adequacy of our model. In addition, we are interested in calibrating a model for our five-minute data. After the analysis and application of the algorithm, we are interested in evaluating the techniques for our original five-minute data. Therefore, we will adjust the seasonality of the spot prices, which will differ from the daily average in that it also displays day and night differences.

Bayesian Calibration in Electricity Spot Price Models

I.1 Model

The algorithm that is used for the calibration of the model in electricity spot price markets from the US is based on the code derived by Jhonny Gonzalez and related paper "Bayesian calibration and number of jump components in electricity spot price model" by Jhonny Gonzalez, John Moriarty and Jan Palczewski. The general approach is applying a Markov chain Monte Carlo (MCMC) to generate samples for calculating posterior functions of the unknown parameters. In general, it has been observed that electricity spot prices have structures consisting of jumps, mean reversion and seasonality (see Jhonny Gonzalez and Palczewski (2016)). Those jumps and their depending mean reversion of the electricity price is comparable to similar structures in other contexts, such as financial time series. The application of a superposition model solving with MCMC is common to calibrate an adequate model. Such a superposition model consists of a diffusion process where the volatility is realized through a sum of reverting jump processes. The program runs an MCMC algorithm for a Bayesian inference to find estimators for an Ornstein-Uhlenbeck process compounded with a sum of multiple jump components. The detailed model is explained in the following sections.

I.1.1 Ornstein-Uhlenbeck Processes

The model and the related processes are based on the observations in the electricity spot markets United Kingdom APXUK and European EEX. Both markets have a mean level to which the spot price reverts, as desribed before. In addition, they present different types of jumps that occur regularly. On one hand there are jumps with a quick decay back to the mean level. On the other hand, there are spikes with a gradual decay.
For the model, different Ornstein-Uhlenbeck processes are defined. A process $Y(t)$, $\theta \leq t \leq T$, which is right continuous with left limits is called an Ornstein-Uhlenbeck process if it is a solution to the following stochastic differential equation:

$$Y(t) = \lambda^{-1}(\mu - Y(t))dt + \sigma dL(t), \qquad Y(0-) = y_0 t.$$

The first term of the equation describes the reversion to a constant level μ. λ is the strength of the return to a mean level. For example, if $\mu \leq Y(t)$, the slope becomes negative, which ensures a reversion to μ. The second term $(\mu - Y(t))$ describes the reversion. The last part of the equation is a noise process which describes the volatility of this component, which is either a Wiener process or a compound Poisson process.

$$y(t) = \mu + (y_0 - \mu)e^{-\lambda^{-1}t} + \int_0^t \sigma e^{-\lambda^{-1}(t-s)} dL(s)$$

For the noise process $L(t)$ we consider two options. A standard Wiener process can be used for modeling the general trading prices. The resulting process is defined with a conditional distribution of $Y(t+s)$ given $Y(t)$, $t \in [0,T]$, $s \in [0, T-t]$ is normallz distributed with the mean

$$\mathbb{E}[Y(t+s)|Y(t) = y] = \mu + (y - \mu)e^{-\lambda^{-1}s}$$

and variance

$$Var[Y(t+s)|Y(t)=y] = \lambda\sigma^2(1-e^{-2\lambda^{-1}s}/2).$$

In the other case, we are using a compound Poisson process

$$L(t) = \sum_{j=1}^{\infty} \xi_j I\{t \geq \tau_j\} \quad \text{or} \quad L(t) = \sum_{i=1}^{N(t)} \xi_i$$

in which τ_j is the time of the jth price jump, ξ_j is the size of the price jump and $N(t)$ counts the number of jumps. Both formula describe the same process. The ξ_i's are independent and identically exponentially distributed. Or for the other formula, $N(t)$ is a Poisson process and describes the number of jumps up to time t. Applying the compound processes as $L(t)$ in our OU process, we get the following explicit form:

$$Y(t+s) = \mu + (Y(t-)-\mu)e^{-\lambda^{-1}s} + \sum_{j:t\leq\tau_j\leq t+s} e^{-\lambda^{-1}(t+s-\tau_j)}\xi_j$$

in which μ is the mean level. It follows the summand that defines the reversion with a speed variable λ. The last term is the noise process with the jumps. It is a sum of iid random variables with the arrival time between t and s.

I.1.2 The Multi-factor Model for Energy Spot Prices

We define $X(t)$ as the deseasonalised price at time t. $X(t)$ is the sum of three different Ornstein-Uhlenbeck processes $(Y_i(t); i = 0, 1, 2)$, which have been described in I.1.1.
The first process has a Wiener process as the noise as mentioned before. It describes the general regular trading characteristics with small price variations:

$$dY_0(t) = \lambda_0^{-1}(\mu - Y_0(t))dt + \sigma dW(t).$$

The two other processes are the positive and negative jumps in the spot prices:

$$dY_i(t) = -\lambda_i^{-1} Y_i(t)dt + dL_i(t) \quad \text{for} \quad i = 1, \cdots, n.$$

In full, the spot price is defined as a sum of processes:

$$X(t) = \sum_{i=0}^{n} w_i Y_i(t)$$

in which w_i can either be 1 or -1 and determines if the process has a negative or positive jump, for our model $w_1 = 1$ and $w_2 = -1$. Furthermore, we define the constant η_i as the intensity rate of the jump process.

At the end $X(t)$ is the resulting superposition model. In this case, the mean level μ is a constant. The mean level is in contrast to the actual spot price, where μ(t) depends on the time (seasonal). The seasonal mean is a result of the variation in demand during the different seasons. The spot price is determined by a product of the deseasonalised prices and an exponential function representing this seasonal behavior,

$$S(t) = e^{f(t)} X(t)$$

with

$$f(t; a_1, \cdots, a_6) \;=\; a_1 + a_2 t + a_3 sin(2\pi t) + a_4 cos(2\pi t) + a_5 sin(4\pi t) + a_6 cos(4\pi t)$$

I.1.3 Jump Intensity Rate

L_i is a compound Poisson process with exponentially distributed jump sizes having mean β_i. For the compound Poisson processes L_i we define a jump intensity rate. There are two options that will be considered for the model. The intensity function can either be defined as constant, which means that we are assuming that the rate is independent of the time, or we are defining a function that considers a periodicity in the jump. This specification is relied on the period in Geman and Roncoroni (2006).[1]

$$I_i(\eta_i, \theta_i, \delta_i, t) \;=\; \eta_i \left[\frac{2}{1 + |sin(\pi(t - \theta_i)k_i)|} - 1 \right]^{\delta_i}$$

in which k_i describes the periodicity in days, η_i the maximal jump rate and δ_i the shape of the periodic function. If we are using the model with a constant intensity function η_i, we define $\vartheta_i = \eta_i$. Otherwise $\vartheta_i = (\eta_i, \theta_i, \delta_i)$.

I.2 Algorithm

I.2.1 Bayesian Inference

The method which is described in this thesis uses Bayesian inference. Bayesian inference is a methodology to get estimators for a parameter θ of a specific distribution. The method differs to common point estimators as follows.

In classical approaches of finding a point estimator for a unknown parameter θ, θ is regarded as a fixed value. Therefore, random samples $x_1, ..., x_n$ from a population indexed by θ are collected. After that, the sample is used to find an estimator for the unknown θ. Instead of assuming that θ is a fixed value, the Bayesian approach is to think that θ is a random variable with a distribution rather than a fixed value. The distribution of the unknown

[1] "Bayesian calibration and number of jump components in electricity spot price models" by Jhonny Gonzalez, John Moriarty and Jan Palczewski (Preprint submitted to Elsevier; January 12, 2016)

parameter is called the prior distribution of θ. It is formed by the experimentor's beliefs, which are based on observed values and the experimenter's experiences. After defining the prior distribution, a random sample of data is taken to update the prior. As a result, we get a posterior distribution which is a conditional distribution of the parameter based on the observed sample of data. The posterior distribution is updated by using the Bayes' Theorem:

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}.$$

We are interested in approximating the posterior distribution $\pi(\theta|x)$. Assume that we have a prior distribution $\theta \sim \pi(\theta)$ and the samples $x_1, ..., x_n$ have the distribution $f(x|\theta)$. Applying the Bayes' Theorem we get:

$$\pi(\theta|x) = \frac{f(x|\theta)\pi(\theta|x)}{m(x)},$$

where $m(x)$ defines the marginal distribution. We can now calculate the posterior distribution by maximizing the likelihood function $f(x|\theta)$. So at the end we get the following result:

$$\pi(\theta|x) \propto f(x|\theta)\pi(\theta|x).$$

in which $\propto$ denotes that the posterior distribution is proportional to the the likelihood function times the prior distribution.

I.2.2 MCMC Algorithm

The Bayesian inference describes a general method to find the approximation of a posterior distribution. Nevertheless, it is not always possible to calculate this distribution directly. An approach estimating the posterior distribution is the Markov chain Monte Carlo Algorithm. It consists of two main ideas: The Markov chain and the Monte Carlo Method, which is

described in the following section.

We are interested in approximating a parameter θ which can be calculated by finding the expected value of a function $h(X)$. Let X be a discrete random vector of possible values $x_j, j \geq 1$. We are interested in finding θ with

$$\theta = E[h(X)] = \sum_{j=1}^{\infty} h(x_j) P\{X = x_j\}.$$

The idea is to use a large number of random numbers and get an estimator through the strong law of large numbers:

$$\lim_{n\to\infty} \sum_{i=1}^{n} \frac{h(x_i)}{n} = \theta = E[h(x)].$$

We can use the Monte Carlo algorithm to simulate a sequence of independent and identically distributed random vectors X_j. Hence, it follows that we can estimate θ by simulating a large number of samples and using the average of these values.

Nevertheless, sometimes it is difficult to generate a sequence of independent values having the same mass function $P\{X = x_j\}$. In this case, we can simulate values to estimate θ by generating a sequence of pairwise states of a vector Markov chain with the stationary probability of $P\{X = x\}$. By using the Markov chain, the values of the sequence do not have to be independent (see Ross (2010)).

The algorithm that is described in this paper makes use of a Markov chain whose stationary distribution is the posterior distribution of the parameters. Each parameter or group of parameters is updated separately while the others are fixed. This is realized with the Gibbs sampler, which operates as follows.

Assume that we have our parameters θ_i. A single iteration of the cycle of the algorithm is an update of one parameter:

$$\theta_i = \theta_1^{'}, \cdots, \theta_{i-1}^{'}, \theta_i, \theta_{i+1}, \cdots, \theta_k.$$

So at iteration i of one step of the algorithm, all parameters before $\theta_j^{'}$ are already updated.

$$\theta_i = \theta_1^{'}, \cdots, \theta_{i-1}^{'}, \theta_i^{'}, \theta_{i+1}, \cdots, \theta_k,$$

after the ith iteration $\theta_i^{'}$ is updated and fixed for the next iteration (see Chris Sherlock and Roberts (2010)).

I.2.3 Random Walk Metropolis

The Gibbs sampler is combined with another methodology. The calculation of the parameters for the jump components Y_1 and Y_2 is based on the Random Walk Metropolis, which is one of the most common Markov chain Monte Carlo algorithms. We are defining X as our current value that we want to update. For the new value we are proposing a jump Y^* that is defined by $Y^* = X^* - X$ with a density function

$$\widetilde{r}(y^*; \lambda) \ := \ \frac{1}{\lambda^d} r\left(\frac{y^*}{\lambda}\right)$$

where λ defines the overall size of our proposed jumps. Updating our value X depends on our acceptance rate of our jump. That means, we have a acceptance probability for updating our value

$$\lambda(x, y^*) = \min\left(1, \frac{\pi(x + y^*)}{\pi(x)}\right).$$

with π as the distribution of our parameter. If we are accepting the jump our new value $X^* \ = 3 \ X \ + \ Y^*$. Otherwise, the value stays unchanged.

In other words, a change of X which brings it closer to the local mode of the posterior

is always accepted, while a downhill move is accepted with a probability related to how far below the local mode this move brings X^*. This kind of acceptance rate ensures, that we find the global mode of the function and do not stick at local mode (see Chris Sherlock and Roberts (2010)).

I.2.4 Application of the MCMC on the Superposition Process

This chapter develops the MCMC scheme to estimate the parameters developed by Gonzalez. For the Bayesian inference, three important functions are needed. The likelihood, prior distribution and conditional distribution. In the first three sections, the derivation of the single functions and distribution are explained. We will start with the likelihood function and the related space augmentation. Secondly, we will briefly state the prior distributions. After that, we will state the particular steps of the algorithm and the resulting conditional distributions.

Defining the likelihood functions through space augmentation

Now assume that we have our observations $\mathcal{X} = \{x_0, \cdots, x_N\}$ of the superposition OU process and define the times as $0 = t_0, \cdots, t_N = T$, and $\Delta_i = t_i - t_{i-1} > 0, i = 1, \cdots, N$, which are the time increments between the different observations. The likelihood $l(\mathcal{X}|\mu, \lambda_0, \sigma, \lambda_1, \vartheta, \beta)$ (compare to $f(x|\theta)$) in I.2.1 can neither be solved analytically nor by numerical integration. Therefore, space augmentation is used to gain independence between the jump components and the other parameters. Furthermore, an independence improves the Gibbs sampler since it is easier to update these parameters separately. The methodology is to augment observations $\mathcal{Y}_y = y_{1,0}, \cdots, y_{1,N}$ of the jump process Y_1 at times t_i. The likelihood function of $\mathcal{X}$ becomes independent of λ_1, ϑ and β. We are defining the variable z_i as the change of x_i through the Gaussian process. At the end, we get the following likelihood function based on the density of a Gaussian OU process:

$$l(\mathcal{X}|\mu,\lambda_0,\sigma,\mathcal{Y}_1)=\prod_{i=1}^{N}\frac{1}{\sqrt{2\pi}\sum_i}\exp\{-\frac{1}{2\sum_i^2}(z_i-\mu-(z_{i-1}-\mu)e^{-\lambda_0^{-1}\Delta_i})^2\}$$

where $\sum_i^2=\lambda_0\sigma^2(1-e^{-2\lambda_0^{-1}\Delta_i})/2$ and

$$z_i=x_i-y_{1,i},\qquad i=0,\cdots,N.$$

In addition, the compound Poisson process $Y_1(t)$ is augmented to make λ_1 and (τ_j,ξ_j) independent. Instead of looking at the whole process Y_1, the jump times and sizes, τ_j and ξ_j, of the Poisson process are summarized in a new variable Φ. The methodology helps to improve the mixing in Gibbs samplers since we reach more independence. The state space is $S=[0,T]\times(0,\infty)$, with taking values in the first component an τ_j an ξ_j in the second. Finally, the following likelihood function is defined for Φ.

$$l(\Phi|\vartheta,\beta)=L(\vartheta;\Phi)\cdot\beta^{-N_T}exp\{-(\beta^{-1}-1)\sum_{j=1}^{N_T}\xi_j\}.$$

$L(\vartheta;\Phi)$ is the density, with the unit intensity function $I(\vartheta,t)$:

$$L(\vartheta;\Phi)=\exp\left\{\sum_{j=1}^{N_T}logI(\vartheta,\tau_j)-\int_0^T I(\vartheta,t)dt+T\right\}.$$

At the end we get the following proportional equation for the Bayesian inference, which can be compared to the general Bayesian inference Model in I.2.1:

$$\pi(\mu,\lambda_0,\sigma,\lambda_1,\vartheta,\beta,\Phi|\mathcal{X})\propto l(\mathcal{X}|\mu,\lambda_0,\sigma,\lambda_1,\Phi)l(\Phi|\vartheta,\beta)\pi(\mu,\lambda_0,\sigma,\lambda_1,\vartheta,\beta),$$

where $\pi(\mu, \lambda_0, \sigma, \lambda_1, \vartheta, \beta)$ is the joint prior density of the parameters.

Prior functions

After defining the likelihood functions, the prior functions are stated for the Bayesian inference. Where IG defines a Inverse-Gamma distribution, Ga a Gamma-distribution and U an Uniform distribution.

parameter	prior distribution
mean level of Gaussian component μ	$N(a_\mu, b_\mu^2)$
volatility of Gaussian component σ^2	$IG(a_{\sigma^2})$
jump size parameter β	$IG(a_\beta, b_\beta)$
mean reversion parameter λ_i	$IG(a_{\lambda_i}, b_{\lambda_i})$
intensity function η	$Ga(a_\eta, b_\eta)$
δ	$Ga(a_\delta, b_\delta)$
θ	$U(a_\theta, b_\theta)$

Table I.1: Unknown parameter and prior distributions

All distributions are supposed to be independent and are based on the analysis of the markets. "Prior expectations are based on existing results in the literature, combined with further exploratory analysis of historical data as necessary (see Jhonny Gonzalez and Palczewski (2016)).

Structured Gibbs samplers and resulting conditioned distributions

After defining the likelihood and prior functions, we can state the actual algorithm and its conditional distributions. The steps are divided by the Gibbs samplers mixing. The mean reversion of the jump component (λ_i) is updated through a random-walk Metropolis Hasting procedure. A transformation of this parameter with an exponential function was applied

Step 1:	update $\mu \sim \pi(\mu\|\rho_0, \sigma, \rho_1, \mathcal{X}, \Phi)$
Step 2:	update $\sigma^2 \sim \pi(\sigma^2\|\rho_0, \rho_1, \mathcal{X}, \Phi)$
Step 3:	update $\rho_0, \rho_1 \sim \pi(\rho_0, \rho_1\|\mu, \sigma, \mathcal{X}, \Phi)$
Step 4:	update $\vartheta \sim \pi(\vartheta\|\Phi)$
Step 5:	update $\pi(\beta\|\Phi)$
Step 6:	update $\Phi \sim \pi(\Phi\|\mu, \rho_0, \sigma, \rho_1, \vartheta, \beta, \mathcal{X})$
Step 7:	Go to step 1.

Table I.2: Overview of particular steps of the algorithm

to ensure a mixing between small and large values. A new parameter for the algorithm $\rho_i = e^{-\lambda_i^{-1}}$ is defined.

Update μ: Conditional distributions can be derived by the likelihood functions. For μ the following distribution is specified:

$$l(\mathcal{X}|\mu, \lambda_0, \sigma, \rho_1, \Phi) \propto \frac{1}{\prod_{i=1}^{N} \sum_i} \exp\{-\frac{1}{2} \sum_{i=1}^{N} \frac{1}{\sum_{i=1}^{2}} (z_i - z_{i-1} e^{-\lambda_0^{-1}\Delta_i} + \mu \ (e^{-\lambda_0^{-1}\Delta_i} - 1))^2\}$$

with conditional distribution $\pi(\mu|\rho_0, \sigma, \rho_1, \mathcal{X}, \phi)$:

$$N\left(\frac{\sum_{i=1}^{N} (1 - e^{-\lambda_0^{-1}\Delta_i}) \sum_i^{-2} (z_i - z_{i-1} e^{-\lambda_0^{-1}\Delta_i}) + \frac{a\mu}{\sigma_0^2}}{\sum_{i=1}^{N} (1 - e^{-\lambda_0^{-1}\Delta_i})^2 \sum_i^{-2} + \frac{1}{b_\mu^2}}, \frac{1}{\sum_{i=1}^{N} (1 - e^{-\lambda_0^{-1}\Delta_i})^2 \sum_i^{-2} + \frac{1}{b_\mu^2}} \right).$$

Update σ^2:

$$\pi(\sigma^2|\rho_0, \rho_1, \mathcal{X}, \Phi) = IG\left(\frac{N}{2} + a_\sigma, \frac{1}{\lambda_0} \sum_{i=1}^{N} \frac{s_i}{1 - e^{-2\lambda_0^{-1}\Delta_i}} + b_\sigma \right)$$

with $s_i = \left(z_i - z_{i-1} e^{-\lambda_0^{-1}\Delta_i + \mu(e^{-\lambda_0^{-1}\Delta_i} - 1)}\right)^2$.

Update ρ_0 and ρ_1: For updating ρ_0 and ρ_1 a random-walk Metropolis-Hastings within Gibbs is used. The conditional distributions are proportional:

$$\pi(\rho_0|\mu,\sigma,\rho_1,\Phi) \propto l(\mathcal{X}|\mu,\rho_0,\sigma,\rho_1,\Phi)\pi(\rho_0),$$

$$\pi(\rho_1|\mu,\sigma,\rho_0,\Phi) \propto l(\mathcal{X}|\mu,\rho_0,\sigma,\rho_1,\Phi)\pi(\rho_1).$$

Update ϑ: For updating ϑ, it is specialized for the two cases of constant intensity and time-dependent intensity. First, a constant intensity function is considered:

$$\eta|\Phi \sim Ga(a_\eta + N_T, T + b_\eta).$$

If the intensity function depends on the time, a random-walk Metropolis-Hastings within Gibbs algorithm is used to update the parameters η, θ and δ. So the conditional distribution satisfies:

$$\pi(\eta,\theta,\delta|\Phi) \propto l(\Phi|\eta,\theta,\delta)\pi(\eta)\pi(\theta)\pi(\delta).$$

while $L(\vartheta|\Theta)$ is calculated numerically.

Update β: The conditional distribution under the condition of Φ is

$$\beta|\Phi \sim IG\left(a_\beta + N_T, \sum_{i=1}^{N_T} \xi_i + b_\beta\right).$$

Update Φ: The latent variable Φ is updated through a random walk Metropolis-Hastings algorithm related to the work of Geyer and Møller(1994), Roberts et al. (2004) and Fröwirt-Schnatter and Sögner (2009) on MCMC techniques for simulating point processes, extending it where appropriate to the case of inhomogeneous Poisson processes (see Jhonny Gonzalez and Palczewski (2016)).
The current state is defined as

$$\Phi \ = \ \{(\tau_1, \xi_1), \cdots, (\tau_{N_T}, \xi_{N_T})\}.$$

N_T is the Poisson process of the jump component Y_1, and states the number of points with jump times $\tau_j \leq \tau$. There are three methods that are used to update the (τ, ξ) that are chosen randomly with equal probability.

Birth/Death: The update depends on a birth-and-death-step. At one state, a pair of (τ_j, ξ_j) can either be added or removed from the current state. The probability for a birth is $p \in (0, 1)$ while the probability for a death is $1 - p$. Our current designation of the data of the Poisson point process is Φ, and the proposed new state (birth) is reached by $\Phi \cup \{(\tau, \xi)\}$. τ is uniformly distributed over $[0, T]$ and ξ follows and exponential distribution $exp(\beta)$. The

transition kernel is:

$$q(\Phi, \Phi \cup \{(\tau, \xi)\}) = \beta^{-1}\ exp(-(\beta_{-1} - 1)\xi)).$$

For the death, a randomly chosen point is removed from the current realization. The points are uniformly chosen and a transition kernel is defined as:

$$q(\Phi, \Phi\{(\tau_i, \xi_i)\}) = \frac{1}{N_T}.$$

Recall that N_T is the number of points, at the current state.
For the Metropolis-Hastings acceptance ratio of the birth we define:

$$\alpha(\Phi, \Phi \cup \{(\tau, \xi)\}) = \min\{1, r(\Phi, (\tau, \xi))\}$$

and for the death:

$$\alpha(\Phi, \Phi \setminus \{(\tau_i, \xi_i)\}) = \min\left\{1, \frac{1}{r(\Phi \setminus \{(\tau_i, \xi_i)\}, (\tau_i, \xi_i))}\right\}$$

in which

$$r(\widetilde{\Phi}, (\tau, \xi)) \ =$$

$$\frac{l(\mathcal{X}|\mu, \rho_0, \sigma, \rho_1, \widetilde{\Phi} \cup \{(\tau, \xi)\})}{l(\mathcal{X}|\mu, \rho_0, \sigma, \rho_1, \widetilde{\Phi})} \frac{\pi(\widetilde{\Phi} \cup \{(\tau, \xi)\}|\vartheta, \beta)}{\pi(\widetilde{\Phi}|\vartheta, \beta)} \frac{1-p}{p} \frac{1}{(N_T + 1)q(\Phi, \Phi \cup \{(\tau, \xi)\})}$$

$$= \frac{l(\mathcal{X}|\mu, \rho_0, \sigma, \rho_1, \widetilde{\Phi} \cup \{(\tau, \xi)\}}{l(\mathcal{X}|\mu, \rho_0, \sigma, \rho_1, \widetilde{\Phi})} \frac{1-p}{p} \frac{T}{\widetilde{N}_T} I(\vartheta, \tau)$$

.

Random time change: All jumps times of the Poisson process are assumed to be ordered $\tau_1 < \cdots < \tau_{N_T}$. For the update, one jump time τ_j is randomly picked and changed to a new jump time τ with a uniform distribution on $[\tau_{j-1}, \tau_{j+1}]$. The point of τ_i is changed to (τ, ξ) with $\xi = e^{-\lambda_1^{-1}(\tau-\tau_i)}\xi_j$. It follows the transition kernel $\pi(\tau, \xi|\mathcal{X}, \mu, \rho_0, \sigma, \rho_1, \vartheta, \beta, \Phi \setminus \{(\tau_j, \xi_j)\})$. The first variable of the j-kernel is uniformly distributed and after, the second variable is a deterministic transformation of ξ_j: $\mathcal{T}(\xi, \tau, \tau')$ and $\mathcal{T} = \mathcal{T}^{-1}$.

The acceptance ratio for the Metropolis-Hastings is $|det\Theta(\xi_j, \tau_j, \tau)|$, related to Tierney (1998, Section 2). Therefore the acceptance ration is:

$$\begin{aligned} r(\Phi, \Phi_{new}) &= \frac{l(\mathcal{X}|\mu, \rho_0, \sigma, \rho_1, \Phi_{new})}{l(\mathcal{X}|\mu, \rho_0, \sigma, \rho_1, \Phi)} \frac{\pi(\tau, \xi|\vartheta, \beta)}{\pi(\tau_j, \xi_j|\vartheta, \beta)} \frac{\widetilde{q}(\tau, \tau_j)}{\widetilde{q}(\tau_j, \tau)} |det\Theta(\xi_j, \tau_j, \tau)| \\ &= \frac{l(\mathcal{X}|\mu, \rho_0, \sigma, \rho_1, \Phi_{new})}{l(\mathcal{X}|\mu, \rho_0, \sigma, \rho_1, \Phi)} \frac{I(\vartheta, \tau)}{I(\vartheta, \tau_j)} \frac{e^{-\beta^{-1}\xi}}{e^{-\beta^{-1}\xi_j}} e^{-\lambda_1^{-1}(\tau-\tau_j)} \end{aligned}$$

where $\widetilde{q}(\tau, \tau') = (\tau_{j+1} - \tau_{j-1})^{-1}$ is the transition density with respect to Lebesgue measure on (τ_{j-1}, τ_{j+1}).

Jump size change: All jump sizes are updated independently. Every jump size ξ_j is changed to $\xi_j' = \xi_j\phi_j$ with $log(\phi_j) \sim N(0, c^2)$. The variance is chosen inversely to the current number of jumps. For this update the Metropolis-Hastings acceptance ratio is

$$\alpha(\Phi, \Phi_{new}) = \min\left\{1, \frac{l(\mathcal{X}|\mu, \rho_0, \sigma, \rho_1, \vartheta, \beta, \Phi_{new})}{l(\mathcal{X}|\mu, \rho_0, \sigma, \rho_1, \vartheta, \beta, \Phi)}\right\} \exp\left\{-(\beta^{-1} - 1)\sum_{i=1}^{N_T}(\xi_i' - \xi_i)\right\} \prod_{i=1}^{N_T} \frac{\xi_i'}{\xi_i}$$

,

while $\prod_{i=1}^{N_T} \frac{\xi_i'}{\xi_i} = \prod_{i=1}^{N_T} \phi_j$.

Expanding the model to three OU processes

For the 3-OU-Model one further jump component is added. Based on the observations the jump can either simulate negative spikes or an additional positive spike in the model. We define $Y_2(t)$ as the second component where w_2 determines a negative or positive jump. The resulting superposition model is

$$X(t) \;=\; Y_0(t) + w_1\; Y_1(t) + w_2\; Y_2(t).$$

For the model the data augmentation is adjusted using observations of the first and the second jump which results in Φ_1 and Φ_2. Both jump components are subtracted to get the pure Gaussian process. The likelihood $l(\mathcal{X}|\mu, \lambda_0, \sigma, \mathcal{Y}_1, \mathcal{Y}_2)$ is given with $z_j \;=\; x_j - y_{1,j} - y_{2,j}$. The likelihood of $\Phi = (\Phi_1, \Phi_2)$ is defined as

$$\pi(\Phi|\vartheta_1, \vartheta_2, \beta_1, \beta_2) \;=\; \pi(\Phi_1|\vartheta_1, \beta_1)\pi(\Phi_2|\vartheta_2, \beta_2)$$

where for i = 1,2,

$$\pi(\Phi_i|\vartheta_i, \beta_i) \;=\; \exp\left\{ \sum_{j=1}^{N_T^i} logI_i(\vartheta_i, t)dt + T \right\} \beta_i^{N_T^i} \exp\left\{ -(\beta_i^{-1} - 1) \sum_{j=1}^{N_T^i} \xi_{i,j} \right\},$$

in which the number of points of Φ_i is N_T^i and

$$\Phi_i \;=\; \{(\tau_{i,1}, \xi_{i,1}), \cdots, (\tau_{i,N_T^i}, \xi_{i,N_T^i})\}.$$

All other parameters are a priori independent. Therefore, the other steps are similar to the 2-OU model. ρ_1 and ρ_2 are updated with a random-walk Metropolis-Hastings algorithm.

Application to the US Data

II.1 Data

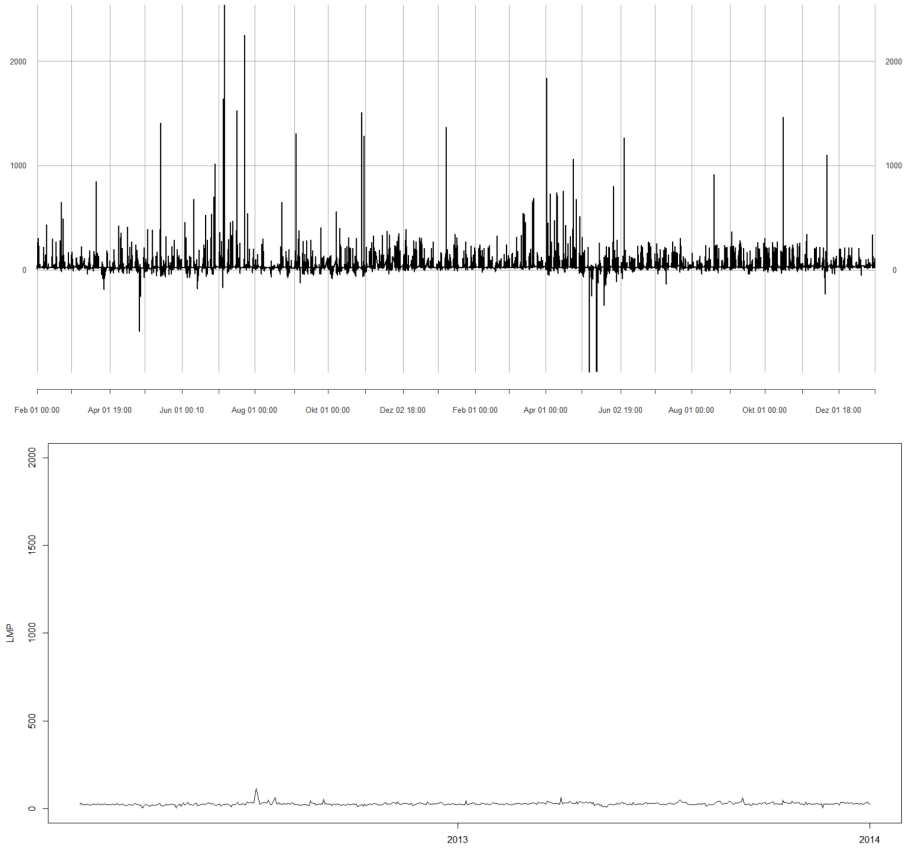

Figure II.1: LMP data of the electricity spot market(WEC) from 2012 and 2013

Figure II.1 illustrates the LMP (locational marginal pricing) in 2012 and 2013. LMP was defined for electric energy prices and reflects the value of electric energy at different locations (see lmp (2018)) The first plot represents the values that were measured every five minutes,

while the other plot shows the average LMP of each day.

Both data are presenting a structure with a mean reversion and different jumps. The plot, showing the five minute data has two different types of jumps. They differ in their frequency, intensity, and decay. While small jumps occur frequently with a slow decay, high spikes are less frequent with a quick reversion to the mean level. Both types can be either negative or positive jumps. The daily average data have the same structure of jumps. Comparing the daily data with the five minutes data, the first plot shows a higher volatility with greater spikes and greater variability in general. The average data values range between 0 and 120 while the 5 minute plot has spikes greater than 2000. This is a result of taking the average which reduces the variance and eliminates outliers.

For the Bayesian calibration, data from the MISO (Midwestern America and Manitoba) electricity spot market, node WEC, have been used. The described algorithm in the paper is applied on daily average data. In the first step, we calculated the average LMP of each day based on the five minute data. Furthermore, the original code excludes weekend prices, because of a different trading behavior comparing to the weekdays. In the US data we removed the weekends, and they are not contained in the plots. If we wanted to calibrate a model for the five minute data by getting estimators from the daily average first, we have to scale the parameters. One problem could be that the scale is not linear. On the other side, we could try to adjust the code by adapting the seasonality.

The electricity spot prices of the United Kingdom APXUK and European EEX that were described in the paper "Bayesian calibration and number of jump components in electricity spot prices model", have a similar structure with two different types of jumps. In addition, negative jumps occur in the European market. Taking a first look at the data, it seems reasonable to apply the algorithm on our daily average data. So, our first step will be to calibrate the Bayesian inference with our daily average data that are derived from the 5-minute LMP.

II.2 Code Adjustments

The first step in the calibration was to deseasonalise the LMP. All US data are listed in an Excel sheet. A program, written in R, collects all data and saves them as a time series which can be more easily handed in MATLAB. After that, they are aggregated by calculating the average for each day. The resulting average daily values are deseasonalised with an exponential function. Figure II.2 shows the deseasonalised data. These data have the same structure as before. They are scaled from 0.5 to 4.5. The mean level is approximately 1. Compared to the results in the paper the plots look similar.

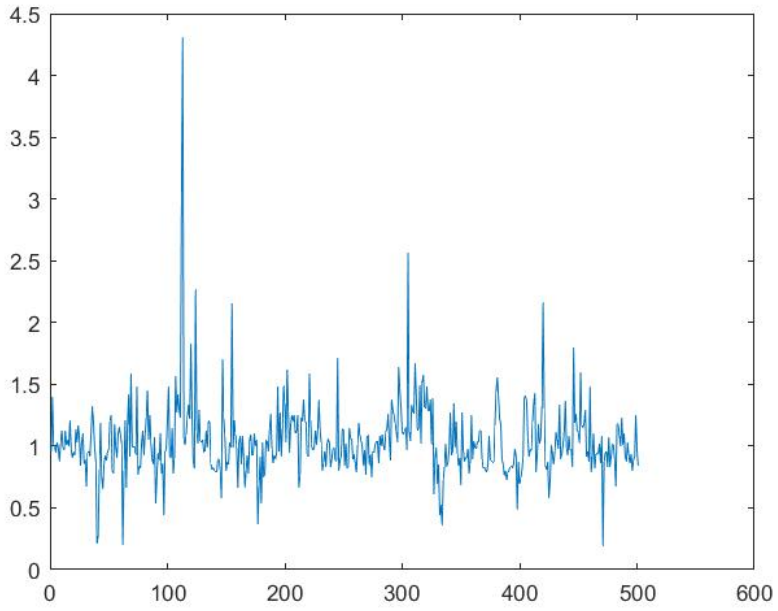

Figure II.2: Deseasonalised LMP of the electricity spot market(WEC) from 2012 and 2013 (daily average)

The function to deseasonalise the data is

$$f(t; a_1, \cdots, a_6) = a_1 + a_2 t + a_3 sin(2\pi t) + a_4 cos(2\pi t) + a_5 sin(4\pi t) + a_6 cos(4\pi t).$$

Dataset	a_1	a_2	a_3	a_4	a_5	a_6
WEC (2012-13)	3.142	0.0034143	-0.027684	-0.032935	0.074056	0.0029311
APXUK (2001-6)	2.5770	0.0008	-0.0817	0.0443	-0.0097	-0.0395
EEK (2000-6)	2.9399	0.0006	0.0055	-0.0803	0.0415	-0.0140
APXUK (2011-15)	3.9005	-0.0001	-0.0014	0.0342	0.0104	-0.0368
EEX (2011-15)	4.0399	-0.0005	-0.0585	-0.0156	0.0298	-0.0315

Table II.3: a_i values for the given data set

Parameter	Posterior mean	Posterior SD
μ	0.991433	0.013706
σ^2	0.069317	0.007807
ρ_0	0.421206	0.040524
λ_0	1.164833	0.131452
ρ_1	0.061567	0.035691
λ_1	0.349751	0.082700
η	0.189159	0.092524
β	1.016242	0.287719
θ	112.351500	43.652695
δ	0.179000	0.136730

Table II.4: 2-OU-I1 model for US WEC data (2012-2013)

The algorithm uses `nlinfit`, a Matlab function, to solve a non linear regression problem with

$$\sum_{i=0}^{N}(logS_{obs}(t_i) - f(t_i))^2 \;\rightarrow\; min,$$

in which S_{obs} defines the observed spot prices. The resulting parameters for $f(t_i)$ are listed in Table II.2. Compared to the parameters of the European and British market, the parameters for the US data appear similar but slightly higher.

The first trial was to calibrate a model with one jump component and a non constant intensity function $I_i(\eta_i, \theta_i, \delta_i, t)$ which is called 2-OU-I1 in the paper. For the initial parameter we set $(\mu, \lambda_0, \sigma, \lambda_1, \eta, \beta, \Phi) = (1, 5, 0.1, 2, 0, .1, 0.5, 0)$ which are the same parameters used for the data. Looking at the posterior means (Table II.4), the values look similar to the

resulting parameters for the 2-OU-I1 model for the 2000-2006 EEX data. These data seem to be the most similar to ours since they have a higher volatility than after the fiscal crisis and they show negative jumps like those present in our data.

Another option is to calibrate a superposition model with two positive jumps. Two jumps seem to be more reasonable by looking at our data plots since two different jump sizes are observed: the jumps with high spikes and fast decays and those with gradual decays. First, we are trying to use a model with two constant intensity functions for the jump. The results are presented in Table II.5. In comparison to the model with two jumps the mean of the electricity price is smaller. This might be a result of our two positive jumps, since the Gaussian process Y_0 is the only process that determines the negative spikes. However, the variance did not increase which might assume that the simulated data could have smaller negative spikes than the observed data. A significant difference to the first trial is a high jump of Y_1 ($\beta_1 = 2.834976$) with a fast decay ($\lambda_1 = 2.977244$) and a rather small spike for the second jump Y_2 ($\beta_2 = 1.431118$).

Parameter	Posterior mean (US WEC)	Posterior SD (US WEC)	Posterior mean (2000-2006 EEX)	Posterior SD (2000-2006 EEX)
μ	0.978192	0.062138	1.0146	0.0226
σ^2	0.062138	0.008822	0.0119	0.0012
ρ_0	0.477978	0.053905	0.8835	0.0150
λ_0	1.374713	0.210605	8.2193	1.1698
ρ_1	0.337353	0.287506	0.1809	0.0344
λ_1	2.977244	33.520654	0.5859	0.0657
ρ_2	0.047897	0.031345	0.2230	0.0384
λ_2	0.318735	0.077044	0.6687	0.0775
$\eta 1$	0.056765	0.060683	0.2515	0.0428
$\eta 2$	0.104151	0.069221	0.1422	0.0320
$\beta 1$	2.834976	128.235855	0.8998	0.1052
$\beta 2$	1.431118	8.609116	0.4308	0.0793
$\theta 1$	-	-	141.3725	3.5071
$\delta 1$	-	-	0.3408	0.0884

Table II.5: 3-OU model for US WEC data (2012-2013) with 2 positive jumps and the EEX(2000-2006 EEX)

Looking at our plots again, we assume that a model with two jump components, a

Parameter	Posterior mean (US WEC)	Posterior SD (US WEC)	Posterior mean (2000-2006 EEX)	Posterior SD (2000-2006 EEX)
μ	1.000359	0.014352	1.0146	0.0226
σ^2	0.047755	0.007655	0.0119	0.0012
ρ_0	0.447031	0.047868	0.8835	0.0150
λ_0	1.255521	0.170665	8.2193	1.1698
ρ_1	0.107612	0.047619	0.1809	0.0344
λ_1	0.446520	0.091994	0.5859	0.0657
ρ_2	0.334034	0.101501	0.2230	0.0384
λ_2	0.947172	0.285864	0.6687	0.0775
$\eta 1$	0.205438	0.072638	0.2515	0.0428
$\eta 2$	0.060739	0.034204	0.1422	0.0320
$\beta 1$	0.901904	0.226196	0.8998	0.1052
$\beta 2$	0.538974	0.232407	0.4308	0.0793
$\theta 1$	114.022577	43.083111	141.3725	3.5071
$\delta 1$	0.125424	0.094486	0.3408	0.0884

Table II.6: 3-OU-I1 model for US WEC data (2012-2013) and the EEX(2000-2006 EEX)

negative and a positive jump, might be more reasonable. We applied a model with two jump components and a non constant intensity function (3-OU-I1). The initial state is $(\mu, \lambda_0, \sigma, \lambda_1, \eta_1, \beta_1, \lambda_2, \eta_2, \beta_2, \Phi) = (1, 5, 0.2, 5, 0.001, 0.5, 1, 0.001, 0.5, 0)$.

The mean level is increased to 1.000359, which again seems to fit with the observation of the data. The volatility of the Gaussian process decreased. This might be a result of our change in the jump component. Y_0 does not have to simulate the negative spikes in the model since we added a negative jump component. The jumps sizes (β) became significant smaller, even if the decay for the first jump is similar to our 3-OU model.

We are comparing the results with the European data before the fiscal crisis. The volatility for WEC seems to be higher. If we are looking at the plot, we can see that the US market shows variance in the data, so the difference is reasonable.

Posterior predictive check

The approach for testing the model adequacy is to apply a posterior predictive check. According to the OU-process Y_0, we can define $\epsilon_j,\ j = 1, \cdots, N$ with

$$Y_0(t_j) \ = \ \mu + (Y_0(t_{j-1}) - \mu)e^{-\lambda_0^{-1}\delta_j} + \left(\frac{\sigma^2\lambda_0}{2}(1 - e^{-2\lambda_0^{-1}\Delta_j})\right)^{\frac{1}{2}} \epsilon_j.$$

They are independent and identically distributed as $N(0, 1)$. If we have observations of Y_0 at times t_j, we can prove model adequacy by testing if ϵ_j is normally distributed. The algorithm uses, at each iteration k, the current state $\Theta^{(k)} = \{\mu^{(k)}, \lambda_i^{(k)}, \sigma^{(k)}, \vartheta_i^{(k)}, \beta_i^k\}$ to retain the path of each jump process $y_{i,j}^{(k)}$. The actual path of Y_0^k can then be computed as

$$z_j^{(k)} \ = \ x_j \ - \ \sum_{i=1}^{n} w_i y_{i,j}^{(k)}, j = 0, \cdots, N,$$

where x_i is the deseasonalised price at time t_i and w_i is the sign of the jump component which determines a negative or positive jump. The noise data are tested with the Kolmogorov-Smirnov (KS) test for the standard Normal distribution. The resulting p-value is used to check the model adequacy of the different models.

For the jump components, at each iteration k, a KS test is performed. It is differed between the set of jump sizes and the set of arrival times at each state. For the jump size, a KS test is subjected an exponential distribution with mean $\beta_i^{(k)}$. For the arrival times we either have a homogeneous or a time-varying case. For the constant intensity function we are using a KS test for an exponential distribution with mean $(\eta_i^{(k)})^{-1}$. Otherwise, a two sample KS test is applied which are taken from the inter-arrival times. All posterior predictive values are reported and stated at the end of the resulting posterior means in the output of the code.

The interpretation of the posterior predictive value differs from the common value. We

OU Process Y_i	2-OU-I1	3-OU	3-OU-I1	3-OU-
Y_0	0.089943	0.079930	0.357189	0.303439
Y_1 (jump times)	0.366119	0.492211	0.353426	0.494793
Y_1 (jump size)	0.508801	0.496615	0.502169	0.502399
Y_2 (jump times)	-	0.497447	0.500015	0.510538
Y_2 (jump size)	-	0.506229	0.505905	0.502649

Table II.7: Posterior predictive values based on the calibration of the models for the daily average data 2012-2013

are evaluating the different p-values like Jhonny Gonzalez and Palczewski (2016). The predictive value is supposed to concentrate around 0.5 if we have a posterior with uncertainty parameters (see Gelman (2013)).

According to the paper, we are accepting model if the p-value is above the threshold of 10%. Table II.7 gives the resulting p-values of our models. It shows that a calibration without a negative jump component is not reasonable since the calculated p-values for Y_0 are clearly below the threshold. The p-value for 2-OU-I1 that only includes a positive jump is 0.089943 and for our 3-OU model that contains two positive jumps we got a p-value equal to 0.079930. So these two models are rejected.

The remaining models are a superposition model with one negative jump and one positive jump component. The first one, 3-OU-I1, includes time-varying jump intensities, while the other model, 3-OU, has a constant jump intensity. The p-values are all above 10%. The posterior predictive value for Y_0 of a model with a non-constant intensity rate is slightly closer to 0.5 than for a constant intensity function. In contrast to that, the p-value for the jump size for component Y_1 is much closer to 0.5 applying a constant intensity. All in all, we can say that both models show a p-values that are close to 0.5. Therefore, both model seem to be acceptable for simulating our data.

Deseasonalised Five Minute Data

After applying the MCMC algorithm on the daily average data that have been calculated through the LMP, we are interested in calibrating a model for the original five minute LMP

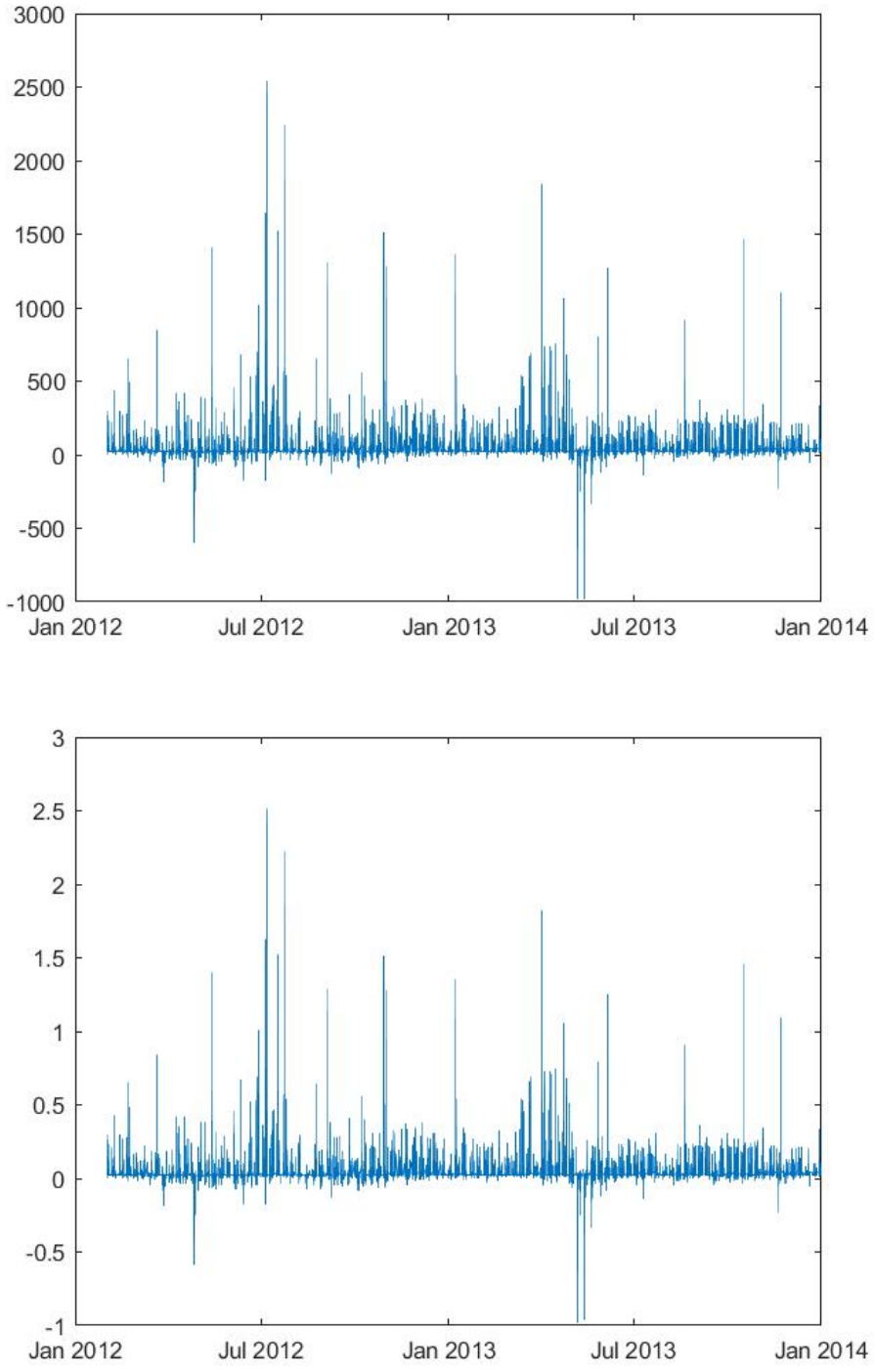

Figure II.3: The first plot shows the deseasonalised plot of the minute data, the other plot shows the related deseasonalised data

data. Recall the seasonality function for the daily data is:

$$f(t; a_1, \cdots, a_6) \;=\; a_1 + a_2 t + a_3 sin(2\pi t) + a_4 cos(2\pi t) + a_5 sin(4\pi t) + a_6 cos(4\pi t)$$

while our deseasonalised price is determined by

$$S(t) \;=\; e^{f(t)} X(t).$$

The function removed the effect of different seasons on spot prices during the year. Each t is divided by a period which is the number of days during the year subtracted by the weekends (260 days). For our five-minute data we have to adjust the period to $260\ days \times 288 \;=\; 74880$, where 288 is number of 5 minutes intervals over one day.

Furthermore, the seasonality function for the daily data does not include a deseasonalisation for the day and night time. In our calibration, before the difference between day and night have not been included since we took the average of all day. For our five minutes data, we are adding two more terms of a *sin* and *cos* function to include day and night time. The resulting function $f(t; a_1, \cdots, a_8)$ can be defined as

$$f(t; a_1, \cdots, a_8) \;=\; a_1 + a_2 t + a_3 sin(2\pi \frac{t}{74880}) + a_4 cos(2\pi \frac{t}{74880})$$

$$+ a_5 sin(4\pi \frac{t}{74880}) + a_6 cos(4\pi \frac{t}{74880}) + a_7 sin(2\pi \frac{t}{288}) + a_2 cos(4\pi \frac{t}{288}).$$

The deseasonalisation is still solved by the Matlab function `nlinfit`, a function to solve the non linear least squares problem.

Figure II.2 shows the results of the deseasonalised data. The structure of the original and new data are the same. The range for our values was restricted to $[-1, 2.5]$.

Data	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8
(2012-13)	3.142	0.0034	-0.0277	-0.0329	0.0741	0.0029	-	-
5 minute	6.9103	0.0000	-0.0001	-0.0008	-0.0005	0.0004	-0.0028	-0.0066

Table II.8: a_i values for the given data set for the daily data and 5 minutes data

Application to the five-minute data

For the deseasonalised five-minute data we calibrated a model based on our observations on the data. The plot shows both high positive spikes and high negative spikes. Comparing this to the results of our daily data simulation, the reasonable models is a model with two jumps, where one jump component determines the negative jump and the other jump component the positive spike.

The model that we applied was the 3-OU model with a negative jump. Recall that this model contains a time homogeneous intensity function.

Parameter	Posterior mean (5-minute)	Posterior SD (5-minute)	Posterior mean (daily)	Posterior SD (daily)
μ	0.104957	0.001678	0.997542	0.018762
σ^2	0.000054	0.000004	0.035613	0.006802
ρ_0	0.964783	0.002718	0.551822	0.055498
λ_0	28.061740	2.208318	1.716843	0.297355
ρ_1	0.316932	0.001354	0.078652	0.034537
λ_1	0.870276	0.003229	0.390014	0.069508
ρ_2	0.394513	0.004757	0.341645	0.113984
λ_2	1.075247	0.013980	0.982283	0.360222
$\eta 1$	0.094283	0.005347	0.208096	0.059393
$\eta 2$	0.082339	0.003702	0.071908	0.039316
$\beta 1$	0.204016	0.010773	0.953824	0.219386
$\beta 2$	0.116257	0.004979	0.561401	0.249962

Table II.9: 3-OU model with one negative jump for US WEC data (2012-2013) for five-minute and daily data

Table II.9 show the estimators for the 3-Ou model with one negative jump for the five-minute and daily data. The mean level is reliable since it is close to the mean level that we would assume by looking at the original data. Even if we got estimators for our model, the posterior predictive value for our five-minute data did not work. Therefore, there is more

research necessary. Looking at the simulated data, the data show a much higher volatility than the original data. Which might assume that the model can not be applied to our five minute data. A reason for that, can be that we need more jump components for our model, since the jumps in the original data seem to differ more than in the daily average.

II.3 Simulation

After calibrating the models, we ran a simulation with the given posterior parameters to compare the resulting data with the original data. We are using the resulting mean to simulate a Gaussian process and one or two jump processes depending, on the model used.

2-OU-I1 Model Though if the model with only one positive jump component does not seem reasonable, we ran a simulation with these parameters. The data, derived by the 2-OU-I1 Model, are plotted in Figure II.4. The positive jumps occur as frequently as in the original plot. Also the mean reversion of both plots do not differ significantly. Nevertheless, the original data show higher negative spikes than the simulated values. This is explained by the fact that the model used does not contain a negative jump component. Therefore, prices less than the mean level are just caused by Y_0 the Gaussian process. As assumed before, a model with two jump components, a positive and negative one, seem to be more reasonable for our model.

3-OU-I1 Model The simulation applying parameters for a 3-OU-Model-I1 is presented in Figure II.5. We can see in the plotted simulated data that Figure higher spikes, which are probably caused by the positive jump component, occur more often than in our observations. If we are looking back at the p-value of the jump time for Y_1 (in Table II.7), the value is 0.353426, which might indicate that a p-value greater than 0.3 is not a good enough threshold. Nevertheless, the variance and mean level are comparable to the original data. In

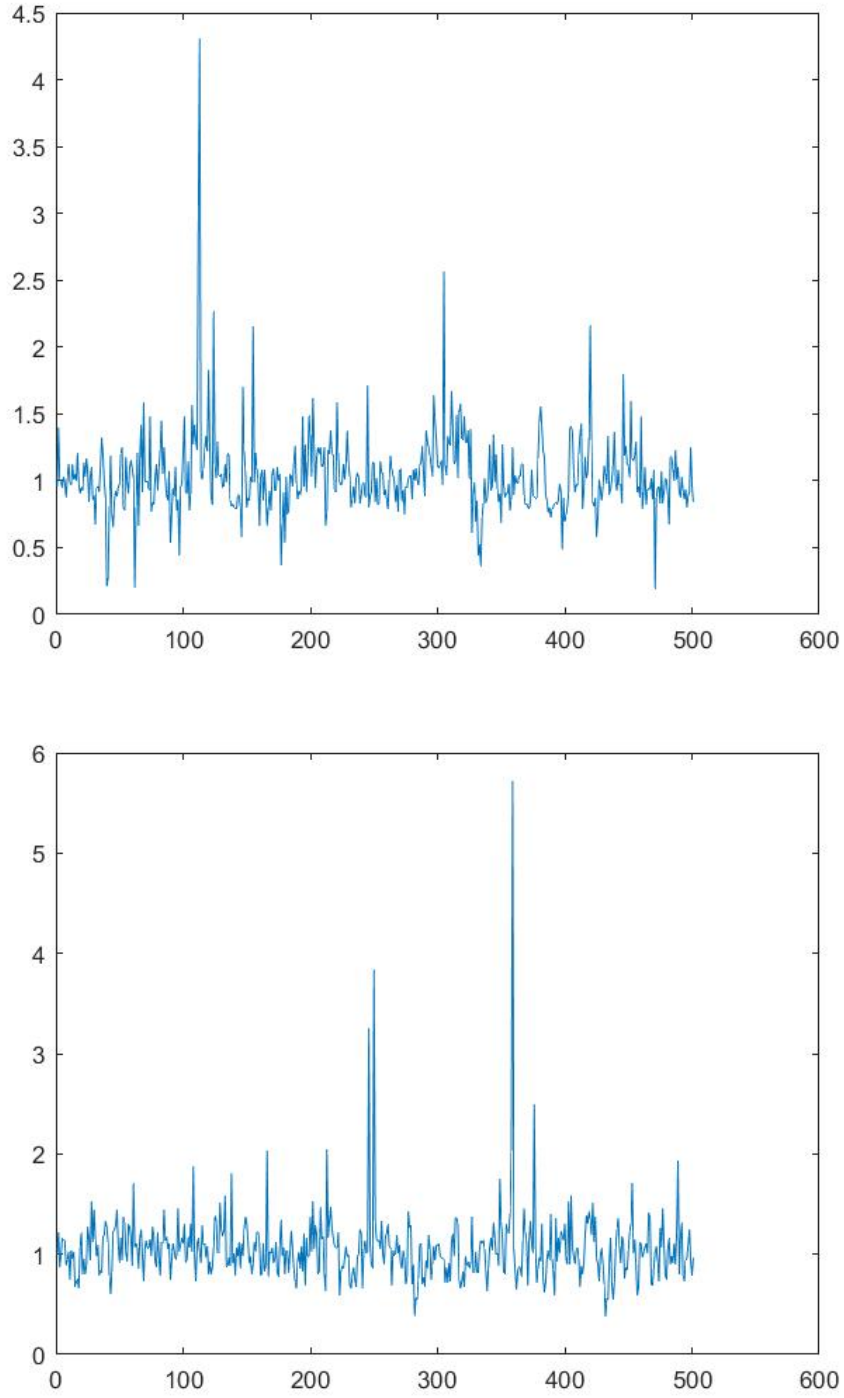

Figure II.4: The first plot shows the deseasonalised plot of the data, the other plot shows the simulated data after calibrating the 2-OU-I1 model

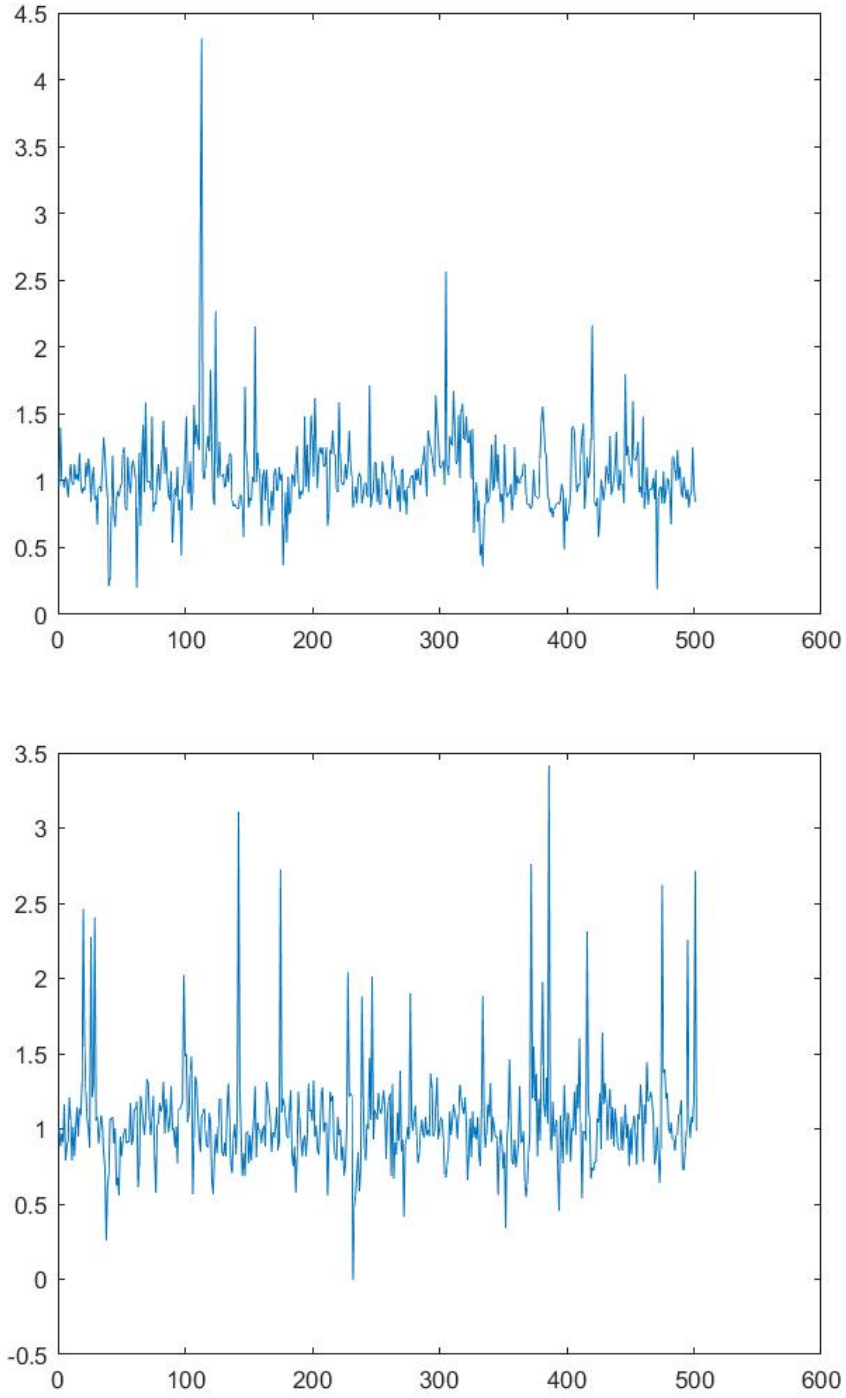

Figure II.5: The first plot shows the deseasonalised plot of the data, the other plot shows the simulated data after calibrating the 3-OU-I1 model

addition, the extension of positive and negative jumps seem to be correct, which confirms our expectations since the p-value of both of our jump sizes are nearby 0.5. Also, the negative jumps occur as often as in the simulated data. All in all, the simulation of the data is similar to the originals. The different structures and assumptions which have been derived by the parameters and p-values have been confirmed by looking at the simulation.

3-OU-Model negative jump The other model that seemed to be reasonable according to our observations was a model with two jump components, one positive and one negative jump with a homogeneous intensity function. Again, we simulated a Gaussian process and two jump processes separately and added them for the superposition model. The resulting data are plotted in Figure II.6. We are comparing the simulated values with the original data again. The positive spikes of both plots are in the same range. However, there occur two negative jumps that seem to differ from the original data. Regarding the jump times, the positive and negative jumps occur as often as in the original data. As a conclusion, the simulated data are reasonable comparing to the original values. The plot confirms our expectations derived from the observations of the data and resulting parameter and p-values. The two negative spikes that are not in the range of the original data can be assumed to be exceptions or outliers.

II.4 Conclusion

Our goal was to find a adequate model to predict the change of electricity prices from the Midwestern America and Manitoba market. The model was derived from former analysis of the European market. We wanted to assess if the results of the European market can be applied on the American market. Therefore, we took a look at the models and algorithms and adjusted these for our data. We were able to run an algorithm and get different estimators for the diverse models that we assessed. Our expectations that the pricing model would fit the data were confirmed. For the Midwestern American and Manitoba market a model

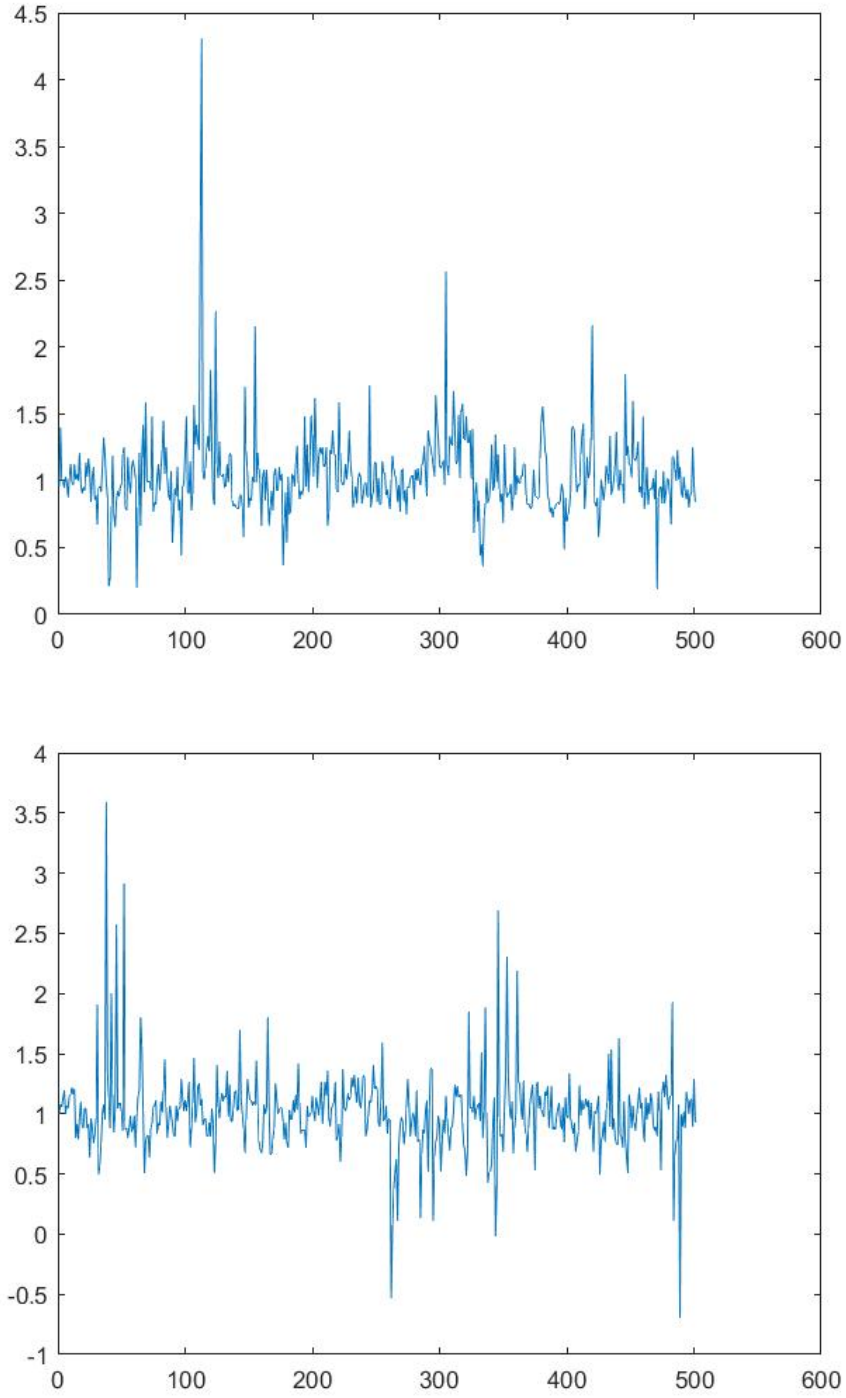

Figure II.6: The first plot shows the deseasonalised plot of the data, the other plot shows the simulated data after calibrating the 3-OU model with one negative jump component

with at least two jump components has to be applied to get reliable predictions for the price change. Furthermore, it is necessary to apply a model with a negative jump component to simulate the negative spikes in the data. We confirmed these results by applying a posterior predictive value check. The p-values gave us an overview which models seem to be adequate and which are not applicable for our data. At the end, we simulated the electricity spot prices and compared them to our data. The simulated data of the 3-OU models with one negative jump component showed a similar plot as the original data.

II.5 Outlook

Our analysis of the Midwestern America and Manitoba market was already successful. We were able to calibrate a model which simulates the prices changes adequately and whose resulting prediction values are similar to the original data. Despite this, the model can still be improved. First, the model is defined through a superposition model which is a sum of different Ornstein-Uhlenbeck processes. In our example we assessed model with one or two jump components. Looking at the plots again, we can see that there are also different sort of positive jumps. Therefore, it is interesting to see what happens to the model if we add more jump components. In this case, we would be interested in finding the optimal number of jump components that predict the price change adequate without overfitting the model.

Another fact are the data that we used to apply our model. If we want to predict a model for American electricity prices, it would be interesting to see how these models change for different areas in the USA. The models will probably differ depending on the areas in the USA. For example, our seasonality will probably be different in the northern parts of the USA compared to states with less seasonality like Florida or California. In addition, we can apply more data than the data from 2012 and 2013 which will probably result in better estimators for our model.

Bibliography

(2018). Iso new england inc. https://www.iso-ne.com/participate/support/faq/lmp [04/07/2018].

Chris Sherlock, P. F. and Roberts, G. O. (2010). The random walk metropolis. *American Economic Review*, 25:172–190.

Gelman, A. (2013). Two simple examples for understanding posterior p-values whose distributions are far from unform. 7.

Jhonny Gonzalez, J. M. and Palczewski, J. (2016). Bayesian calibration and number of jump components in electricity spot price model. *American Economic Review*.

Ross, S. M. (2010). Introduction to probability models. 10.

Appendix

A Main For Daily Data

```
% mainMCMC0.m
%
% The following commands can be used to reproduce the results from
%
% 'Bayesian calibration and number of jump components in electricity
% spot price models' by Jhonny Gonzalez, John Moriarty & Jan Palczewski
%
%
%% ========================================================================
%% Plot deseasonalised data as shown in Figure 1

plotDeasesonalisedData_US;

%% ========================================================================
%% Fitted seasonal trend coefficients shown in Table A.3

[~,beta1] = fitSeasonalTrend_1(false);
[~,beta2] = fitSeasonalTrend_1(false);
[~,beta3] = fitSeasonalTrend_1(false);
[~,beta4] = fitSeasonalTrend_1(false);

fprintf('%10s %12s %12s %12s %10s %13s\n', 'a1','a2','a3','a4','a5','a6')
```

```
fprintf('
    ------------------------------------------------------------------------\
    n')
disp(beta1)
disp(beta2)
disp(beta3)
disp(beta4)

%% ========================================================================
%% Fit 2-OU model to APXUK 2001 - 2006
% Results shown in Table 1, Table 3 and Section 4.2.4
model = '2-OU';
hPar = getHyperparameters(model);  % hyperparameters
x0   = getInitStates(model);       % initial states
n = 2000000;                       % number of mcmc steps
propVar = struct('lambda0',0.05, 'lambda1',0.05); % variance of proposals
    distributions
NPhi = 5; % number of updates of Phi per mcmc iteration
str = strcat('results/2OU-APXUK1','.mat');  % path to save results
E = fitSeasonalTrend_1(false);       % deseasonalise the data
% run algorithm
rng(0) % for reproducibility
tic
[par, Lchain, accRate] = gibbs(E, model, n, x0, hPar, propVar, NPhi, str);
toc

% Diagnostics for 2-OU model fit to APXUK 2001 - 2006
% Print posterior mean and SD, and p-values
burnin = 5000; % chain was saved one every 100th iteration
results = diagnostics(str, model, burnin, false);

%% ========================================================================
%% Fit 3-OU model to APXUK 2001 - 2006
```

```
% Results shown in Table 2, Table 3 and Section 4.2.4
model = '3-OU';
hPar = getHyperparameters(model);  % hyperparameters
x0   = getInitStates(model);       % initial states
n = 2000000;                       % number of mcmc steps
propVar = struct('lambda0',0.1, 'lambda1',0.05, 'lambda2', 0.05); % variance
    of proposals distributions
NPhi = 5; % number of updates of Phi per mcmc iteration
str = strcat('results/3OU-APXUK1','.mat');  % path to save results
E = fitSeasonalTrend_1(false);    % deseasonalise the data
% run algorithm
rng(0) % for reproducibility
tic
[par, accRate] = gibbs3OU(E, model, n, x0, hPar, propVar, NPhi, str);
toc

% Diagnostics for 3-OU model fit to APXUK 2001 - 2006
% Print posterior mean and SD, and p-values
burnin = 5000; % chain was saved one every 100th iteration
results = diagnostics(str, model, burnin, false);

%% ========================================================================
%% Plot Figure 3
% Requires MCMC results for the fitted 2-OU and 3-OU models to APXUK 2001 -
    2006
strfile1 = strcat('results/2OU-APXUK1','.mat'); % assumed path to saved
    results of 2-OU model
strfile2 = strcat('results/3OU-APXUK1','.mat'); % assumed path to saved
    results of 3-OU model
%!!!!!!!!!!!!Error time range
%plotPaths(strfile1, strfile2); % it may take a while to load files if n was
    large

```

```
%% ========================================================================
%% Fit 2-OU model to EEX 2001 - 2006
% Results shown in Table 4, Table 3 and Section 4.3.1
model = '2-OU';
hPar = getHyperparameters(model);  % hyperparameters
x0   = getInitStates(model);       % initial states
n = 2000000;                       % number of mcmc steps
propVar = struct('lambda0',0.05, 'lambda1',0.05); % variance of proposals
    distributions
NPhi = 5; % number of updates of Phi per mcmc iteration
str = strcat('results/2OU-EEX1','.mat');  % path to save results
E = fitSeasonalTrend_1(false);       % deseasonalise the data
% run algorithm
rng(0) % for reproducibility
tic
[par, Lchain, accRate] = gibbs(E, model, n, x0, hPar, propVar, NPhi, str);
toc

% Diagnostics for 2-OU model fit to EEX 2001 - 2006

burnin = 5000; % chain was saved one every 100th iteration
results = diagnostics(str, model, burnin, false);

%% ========================================================================
%% Fit 2-OU-I1 model to EEX 2001 - 2006
% Results shown in Table 4
model = '2-OU-I1';
hPar = getHyperparameters(model);  % hyperparameters
x0   = getInitStates(model);       % initial states
n    = 2000000;                    % number of mcmc steps
propVar = struct('lambda0',0.05, 'lambda1',0.05); % variance of proposals
    distributions
NPhi = 5; % number of updates of Phi per mcmc iteration
```

```
str = strcat('results/2OU-I1-EEX1','.mat');  % path to save results
E = fitSeasonalTrend_1(false);       % deseasonalise the data
% run algorithm
rng(0) % for reproducibility
tic
[par, Lchain, accRate] = gibbs(E, model, n, x0, hPar, propVar, NPhi, str);
toc

% Diagnostics for 2-OU-I1 model fit to EEX 2001 - 2006
burnin = 5000; % chain was saved one every 100th iteration
results = diagnostics(str, model, burnin, false);

%% ========================================================================
%% Fit 3-OU-minus model to EEX 2001 - 2006
% Results shown in Table 4,  Table 3 and Section 4.3.1 for 3-OU model
model = '3-OU-';
hPar = getHyperparameters(model);  % hyperparameters
x0   = getInitStates(model);       % initial states
n    = 2000000;                    % number of mcmc steps
propVar = struct('lambda0',0.1, 'lambda1',0.05, 'lambda2', 0.05); % variance
    of proposals distributions
NPhi = 5; % number of updates of Phi per mcmc iteration
str = strcat('results/3OU-minus-EEX1','.mat');  % path to save results
E = fitSeasonalTrend_1(false);       % deseasonalise the data
% run algorithm
rng(0) % for reproducibility
tic
[par, accRate] = gibbs3OU(E, model, n, x0, hPar, propVar, NPhi, str);
toc

% Diagnostics for 3-OU-minus model fit to EEX 2001 - 2006
burnin = 5000; % chain was saved one every 100th iteration
results = diagnostics(str, model, burnin, false);
```

```
144
145 %% ========================================================================
146 %% Fit 3-OU-I1 model to EEX 2001 - 2006
147 % Results shown in Table 4, Table 3, and Section 4.3.1
148 model = '3-OU-I1';
149 hPar = getHyperparameters(model);  % hyperparameters
150 x0   = getInitStates(model);       % initial states
151 n    = 2000000;                    % number of mcmc steps
152 propVar = struct('lambda0',0.05, 'lambda1',0.1, 'lambda2', 0.1); % variance of
      proposals distributions
153 NPhi = 5; % number of updates of Phi per mcmc iteration
154 str = strcat('results/3OU-I1-EEX1','.mat');  % path to save results
155 E = fitSeasonalTrend_1(false);       % deseasonalise the data
156 % run algorithm
157 rng(0) % for reproducibility
158 tic
159 [par, accRate] = gibbs3OU(E, model, n, x0, hPar, propVar, NPhi, str);
160 toc
161
162 % Diagnostics for 3-OU-I1 model fit to EEX 2001 - 2006
163 burnin = 5000; % chain was saved one every 100th iteration
164 results = diagnostics(str, model, burnin, false);
165
166 %% ========================================================================
167 %% Plot Figure 2
168 % Requires MCMC results for the fitted 3-OU model to EEX 2000 - 2006
169 strfile = strcat('results/3OU-I1-EEX1','.mat');  % assumed file path to
      results, change as needed
170 burnin = 5000;
171 plotJumpSeasonality(strfile, burnin)
172
173 %% ========================================================================
174 %% Prior sensitivity analysis for the 3-OU-I1 model with EEX 2000 - 2006 data
```

```
% Results are shown in Table A.4, Prior 2 and Prior 3 columns

%---------------- Estimate model with Prior 2, new prior for sigma
model = '3-OU-I1';
hPar = getHyperparameters(model);  % hyperparameters
x0   = getInitStates(model);       % initial states
n    = 2000000;                    % number of mcmc steps
propVar = struct('lambda0',0.05, 'lambda1',0.1, 'lambda2', 0.1); % variance of
    proposals distributions
NPhi = 5; % number of updates of Phi per mcmc iteration
str = strcat('results/3OU-I1-EEX1-sensitivity-sigma','.mat');  % path to save
   results
E = fitSeasonalTrend_1(false);       % deseasonalise the data
% run algorithm
rng(0)
[par, accRate] = gibbs3OU(E, '3-OU-I1-sensitivity-sigma', n, x0, hPar, propVar
   , NPhi, str);
burnin = 5000;
% display results
results = diagnostics(str, model, burnin, true);

%---------------- % Estimate model with Prior 3, new prior for etai
str = strcat('results/3OU-I1-EEX1-sensitivity-eta','.mat');  % path to save
   results
rng(0)
[par, accRate] = gibbs3OU(E, '3-OU-I1-sensitivity-eta', n, x0, hPar, propVar,
   NPhi, str);
% display results
results = diagnostics(str, model, burnin, false);

%% ========================================================================
%% Fit models to 2011 - 2015 data

```

```
%% ========================================================================
%% Fit 2-OU model to APXUK 2011 - 2015
% Results shown in Table 5 and 6
model = '2-OU';
hPar = getHyperparameters(model);  % hyperparameters
x0   = getInitStates(model);       % initial states
n    = 2000000;                    % number of mcmc steps
propVar = struct('lambda0',0.05, 'lambda1',0.05); % variance of proposals
    distributions
NPhi = 5; % number of updates of Phi per mcmc iteration
str = strcat('results/2OU-APXUK2','.mat');  % path to save results
E = fitSeasonalTrend_1(false);       % deseasonalise the data
% run algorithm
rng(0) % for reproducibility
tic
[par, Lchain, accRate] = gibbs(E, model, n, x0, hPar, propVar, NPhi, str);
toc

% Diagnostics for 2-OU model fit to APXUK 2011 - 2015

burnin = 5000; % chain was saved one every 100th iteration
results = diagnostics(str, model, burnin, false);

%% ========================================================================
%% Fit 2-OU^{-} model to EEX 2011 - 2015
% Results shown in Table 5 and 6
model = '2-OU-';
hPar = getHyperparameters(model);  % hyperparameters
x0   = getInitStates(model);       % initial states
n    = 2000000;                    % number of mcmc steps
propVar = struct('lambda0',0.05, 'lambda1',0.05); % variance of proposals
    distributions
NPhi = 5; % number of updates of Phi per mcmc iteration
```

```
234 str = strcat('results/2OU-minus-EEX2','.mat');  % path to save results
235 E = fitSeasonalTrend_1(false);       % deseasonalise the data
236 % run algorithm
237 rng(0) % for reproducibility
238 tic
239 [par, Lchain, accRate] = gibbs(E, model, n, x0, hPar, propVar, NPhi, str);
240 toc
241
242 % Diagnostics for 2-OU^{-} model fit to EEX 2011 - 2015
243 burnin = 5000; % chain was saved one every 100th iteration
244 results = diagnostics(str, model, burnin, false);
245
246 %% ========================================================================
247 %% Fit 2-OU-I1 model to EEX 2011 - 2015
248 % Results shown in Table 5
249 model = '2-OU-I1';
250 hPar = getHyperparameters(model);  % hyperparameters
251 x0   = getInitStates(model);       % initial states
252 n    = 2000000;                    % number of mcmc steps
253 propVar = struct('lambda0',0.05, 'lambda1',0.05); % variance of proposals
    distributions
254 NPhi = 5; % number of updates of Phi per mcmc iteration
255 str = strcat('results/2OU-I1-EEX2','.mat');  % path to save results
256 E = fitSeasonalTrend_1(false);       % deseasonalise the data
257 % run algorithm
258 rng(0) % for reproducibility
259 tic
260 [par, Lchain, accRate] = gibbs(E, model, n, x0, hPar, propVar, NPhi, str);
261 toc
262
263 % Diagnostics for 2-OU-I1 model fit to EEX 2011 - 2015
264 burnin = 5000; % chain was saved one every 100th iteration
265 results = diagnostics(str, model, burnin, false);
```

```
266
267 %% ========================================================================
268 %% Repeated analysis of the 2-OU model on simulated data
269 % Results shown in Table A.2
270 strfile = 'results/2c-estimatesSimT1000.mat'; % file path to save results
271 burnin = 3000;
272
273 %{
274 % the true simulation values are
275 mu      = 1;    % level of mean reversion
276 lambda0 = 8;    % time to mean reversion
277 sigma   = 0.1;  % volatility
278 lambda1 = 2;    % time to mean reversion
279 beta    = 0.7;  % mean jump size
280
281 eta varies over [0.05 0.1 0.2 0.3 ]
282 %}
283 % this function may take long to complete, > 15 hours
284 % use the parallel toolbox if available to improve computational time
285 repeatAnalysis(strfile);
286 results = processRepeatAnalysis(strfile, burnin);
287
288 %% ========================================================================
289 %% Compare ACF with different number of updates of Phi per MCMC iteration
290 % as shown in Figure A.1
291 % ---------------- simulate 3-OU process
292 w = [1  1];  % sign of each jump component
293 Nt = 1000;
294 mu      = 1;    sigma   = 0.15;
295 lambda0 = 8;    lambda1 = 3; lambda2 = 0.5;
296 eta1    = 0.1;  eta2    = 0.05;
297 beta1   = 0.5;  beta2   = 1;
298 rng(0)
```

```
E = simulate3OUModel(Nt,mu, lambda0, sigma, lambda1,lambda2, eta1, eta2, beta1
    , beta2, w);

model = '3-OU';
hPar = getHyperparameters(model);  % hyperparameters
x0   = getInitStates(model);       % initial states
n = 1000000;                       % number of mcmc steps
propVar = struct('lambda0',0.1, 'lambda1',0.05, 'lambda2', 0.05); % variance
    of proposals distributions

% ---------------- estimate 3-OU model with 1 update of Phi per MCMC iter.
rng(0)
strfile1 = strcat('results/3OU-simulationNPhi_1','.mat');  % path to save
    results
NPhi = 1; % number of updates of Phi per mcmc iteration
[par1, accRate] = gibbs3OU(E, model, n, x0, hPar, propVar, NPhi, strfile1);

% ---------------- estimate 3-OU model with 5 updates of Phi per MCMC iter.
rng(0)
strfile2 = strcat('results/3OU-simulationNPhi_5','.mat');  % path to save
    results
NPhi = 5; % number of updates of Phi per mcmc iteration
[par2, accRate] = gibbs3OU(E, model, n, x0, hPar, propVar, NPhi, strfile2);

% ---------------- plot ACFs of eta
burnin = 5000;
figure;
hold on
ACF1 = autocorr(par1(burnin:end,5),800);
ACF2 = autocorr(par2(burnin:end,5),800);
plot(ACF1)
plot(ACF2)
grid on, box off, ylabel('ACF of \eta_1')
```

```
328 legend('One update of \Phi per MCMC iteration','Five updates of \Phi per MCMC
    iteration')
329 h = gca; h.FontSize = 11;
330
331 %% ================================================================================
332 %% Summary of process Phi = (Phi1, Phi2) of 3-OU model
333 % as shown in Figure A.2
334
335 % ---------------- simulate 3-OU process and save true processes Phi1, Phi2
336 w = [1  1];       % sign of each jump component
337 Nt = 1000;
338 mu      = 1;     sigma   = 0.15;
339 lambda0 = 8;     lambda1 = 3;     lambda2 = 0.5;
340 eta1    = 0.1;   eta2    = .05;
341 beta1   = 0.5;   beta2   = 1;
342 rng(0)
343 [E,LT1,LT2] = simulate3OUModel(Nt,mu, lambda0, sigma, lambda1,lambda2, eta1,
    eta2, beta1, beta2, w);
344 strfile1 = 'results/3OU-simulationLT1LT2.mat';
345 save(strfile1,'E','LT1','LT2')
346
347 % ---------------- estimate 3-OU model using the corresponding observed process
348 model = '3-OU';
349 hPar = getHyperparameters(model);  % hyperparameters
350 x0   = getInitStates(model);       % initial states
351 n    = 1000000;                    % number of mcmc steps
352 NPhi = 5; % number of updates of Phi per mcmc iteration
353 propVar = struct('lambda0',0.1, 'lambda1',0.05, 'lambda2', 0.05); % variance
    of proposals distributions
354 strfile2 = strcat('results/3OU-simulationrng0','.mat');  % path to save
    results
355 % run algorithm
356 rng(0)
```

```
357 tic
358 [par, accRate] = gibbs3OU(E, model, n, x0, hPar, propVar, NPhi, strfile2);
359 toc
360
361 % ---------------- plot true Phi vs estimated Phi
362 plotPhi(strfile1,strfile2)
```

B Fit Seasonal Trend For Daily Data

```
function [E, beta] = fitSeasonalTrend_1(plotFit)
%FITSEASONALTREND Fit seasonal trend function to dataset specified by datastr
%
% [E, beta] = fitSeasonalTrend(datastr, plotFit)
%
% Input arguments:
%   plotFit - boolean variable, if true results are plotted
%
%   Output arguments:
%
%   E - deseasonalised series
%
%   beta - fitted parameters
%%

        idx1 = 1; idx2 = 501;          % 01/01/2012 - 31/12/2013

load('../data/US2012_2013.mat');

    namedata = 'US';
    units = '$';
    %Name of the data
    %E0 is the price
    E0 = US20122013.average;


%index to just get the one before or after 2006 _______DO THAT LATER WHEN I
%HAVE ALL THE DATA
E0 = E0(idx1:idx2)';
E1 = E0;
```

```
31
32 %!!!! probably skip that for the us data first
33 % there are 3 large negative prices on EEX2, average them out with neighbours
34 %idx = find(E0<0);
35 %if ~isempty(idx)
36 %    E0(idx(1:2)) = mean(E0([idx(1)-1 idx(2)+1]));
37 %    E0(idx(3))   = mean(E0([idx(3)-1 idx(3)+1]));
38 %end
39
40 % specify seasonal trend function
41 %!!!adjust the periods
42 period = 260; % period 365 days - weekends
43 strendFun = @(a,t)(a(1) + a(2)*t +a(3)*sin(2*pi*t/period)+ a(4)*cos(2*pi*t/
       period)...
44     + a(5)*sin(4*pi*t/period)+a(6)*cos(4*pi*t/period));
45
46 % initial guess
47 beta0 = [1 1 1 1 1 1];
48 %time points
49 t = 1:length(E0);
50 % set robust options for nlinfit
51 % options = statset('nlinfit');
52 % options.Robust = 'on';
53 % fit
54 [beta, R]= nlinfit(t,log(E0),strendFun,beta0);
55 % deseasonalise raw data
56 E = E1./exp(strendFun(beta,t));
57
58 % plot results?
59 if plotFit
60
61     figure;
62     plot(log(E0))
```

```
63      hold on
64      grid on
65      plot(strendFun(beta,t),'black')
66      plot(E), grid on, box off
67
68      xlabel('Time (days)'), ylabel([units '/MWh'])
69      legend(['Daily mean ' namedata ' log-prices'],'Seasonal trend', ['
            Deseasonalised ' namedata ' series' ])
70      h = gca; h.FontSize = 11;
71  end
```

C Fit Seasonal Trend For Five-minute Data

```
function [E, beta] = fitSeasonalTrend_2(plotFit)
%FITSEASONALTREND Fit seasonal trend function to dataset specified by datastr
%
% [E, beta] = fitSeasonalTrend_2(datastr, plotFit)
%
% Input arguments:
%   plotFit - boolean variable, if true results are plotted
%
%   Output arguments:
%
%   E - deseasonalised series
%
%   beta - fitted parameters
%%

        idx1 = 1; idx2 = 15061;          % 01/01/2012 - 31/12/2013 5 mins

load('../data/US_5min.mat');

    namedata = 'US';
    units = '$';
    %Name of the data
    %E0 is the price
    E0 = WITHOUTAVERAGEFiltered.LMP;


%index to just get the one before or after 2006 _______DO THAT LATER WHEN I
%HAVE ALL THE DATA
E0 = E0(idx1:idx2)';
E1 = E0;
```

```
31
32
33 % specify seasonal trend function
34 %!!!adjust the periods
35 period = 74880; % period 365 days - weekends
36 strendFun = @(a,t)(a(1) + a(2)*t +a(3)*sin(2*pi*t/period)+ a(4)*cos(2*pi*t/
       period)...
37     + a(5)*sin(4*pi*t/period)+a(6)*cos(4*pi*t/period))...
38     + a(7)*sin(2*pi*t/288)+a(8)*cos(2*pi*t/288);
39
40 % initial guess
41 beta0 = [1 1 1 1 1 1 1 1];
42 %time points
43 t = 1:length(E0);
44 minE0 = min(E0);
45 if (min(E0) <0)
46     E0 = E0 - minE0 +1;
47 end
48 % set robust options for nlinfit
49 % options = statset('nlinfit');
50 % options.Robust = 'on';
51 % fit
52 [beta, R]= nlinfit(t,log(E0),strendFun,beta0);
53 % deseasonalise raw data
54 E = E1./exp(strendFun(beta,t));
55
56 % plot results?
57 if plotFit
58
59     figure;
60     plot(log(E0))
61     hold on
62     grid on
```

```
63      plot(strendFun(beta,t),'black')
64      plot(E), grid on, box off
65
66      xlabel('Time (days)'), ylabel([units '/MWh'])
67      legend(['Daily mean ' namedata ' log-prices'],'Seasonal trend', ['
            Deseasonalised ' namedata ' series' ])
68      h = gca; h.FontSize = 11;
69  end
```

D Simulation of the Electricity Prices

```
%simulate US Data:
%We use the simulate3OUModel.m to simulate the data:

%
    ------------------------------------------------------------------------------------------


%3OU-I1 daily 2012-13
I1.t0     = 114.022577;    % theta
I1.delta  = 0.125424;    % delta
I1.eta    = 0.205438;    % eta
I1.period = 260;
I2.t0     = 114.022577;    % theta
I2.delta  = 0.125424;    % delta
I2.eta    = 0.060739;    % eta
I2.period = 260;
[E,LT1,LT2] = simulate3OUModel(501,1.000359,1.255521,sqrt(0.047755)
    ,0.446520,0.947172,I1,I2,0.901904,0.538974,[1,-1])
plot(E);

%
    ------------------------------------------------------------------------------------------


%3OU- daily 2012-13
[E,LT1,LT2] = simulate3OUModel(501,0.997542,1.716843,sqrt(0.035613)
    ,0.390014,0.982283,0.208096,0.071908,0.953824,0.561401,[1,-1])
plot(E);
%
    ------------------------------------------------------------------------------------------


%2OU-I1 daily 2012-13
```

```
I1.t0     = 112.351500;    % theta
I1.delta  = 0.179000;    % delta
I1.eta    = 0.189159;    % eta
I1.period = 260;
[E,LT1] = simulateModel(501, 0.991433, 1.164833, sqrt(0.069317), 0.349751, I1,
    1.016242, 1)
plot(E);
%
    ------------------------------------------------------------------------------


%3OU model (2 positive jumps) 04172013
w =[1,1]
[E,LT1,LT2] = simulate3OUModel(501,0.978192,1.374713, sqrt(0.062138),
    2.977244,0.318735,0.056765, 0.104151, 2.834976, 1.431118, w);
plot(E)
%
    ------------------------------------------------------------------------------


%3OU- model five minute 15061
[E,LT1,LT2] = simulate3OUModel(15061,0.104957,28.061740,sqrt(0.000054)
    ,0.870276,1.075247,0.094283,0.082339,0.204016,0.116257,[1,-1])
plot(E);

%
    ------------------------------------------------------------------------------
```

E Simulation of a Model With One Jump Component

```
function [E,T,L] = simulateModel(Nt, mu, lambda0, sigma, lambda1, jRate, beta,
    negJ)
%SIMULATEMODEL Simulate paths of the 2-OU models, X = Y0 + Y1
%
% [E,T,L] = simulateModel(Nt, mu, lambda0, sigma, lambda1, jRate, beta, negJ)
%
% dY0 = 1/lambda0*(mu-Y0)dt + sigma*dWt, Wt is a Wiener process
%
% dY1 = -1/lambda1*Y1*dt + dL1, L1 is a homogeneous or inhomgeenous
% compound Poisson process
%
% Input arguments:
%
%   Nt - Number of timesteps, observation period is [0,Nt] with grid size 1
%
%   mu - Level of mean reversion of Y0
%
%   lambda0 - Time to mean reversion (inverse of speed of mean reversion)
%
%   sigma - Volatility
%
%   lambda1 - Time to mean reversion of Y1
%
%   jRate - if L1 is time-homogeneous, jRate is a scalar representing the
%       jump intensity rate of L1. If L1 is time-inhomogeneous jRate is
%       structure array containing the parameters for the intensity
%       function given in intensityFun.m, eg. jRate.t0 = 130,
%       jRate.period = 260, jRate.delta = 0.15, jRate.eta = 0.3
%
%   beta - Mean jump size of L1
```

```
%
%   negJ  - scalar indicating whether Y1 is positive (negJ = 1)
%               or negative (negJ = -1)
%
% Output arguments:
%
%   E - Discretised sample path at grid times 0,1,...,Nt, E1 below is
%       the exact solution
%   T - True jump times
%   L - True jump sizes

% choose jump size distribution
%   1 for Pareto, 2 Exponential, 3 Gamma
jumpDistribution = 2;

% simulate Poisson process

if ~isstruct(jRate)   % homogeneous

    [L,jumpTimes] = homogeneousCPoisson(Nt, jRate, beta, jumpDistribution);

else                  % inhomogeneous

    [L,jumpTimes] = compoundPoisson(Nt, jRate.t0, jRate.period, ...
        jRate.delta, jRate.eta, beta, jumpDistribution);

end
% simulation times, combine fix time grid points and jump times
[T,IX] = sort([1:Nt jumpTimes]);    % IX is the permutation vector
L = [zeros(1,Nt) L]; L = L(IX);     % sort L over the 'combined' grid

lenT = length(T); % time

```

```
% initialise variables
E1 = zeros(1,lenT);
Y0 = zeros(1,lenT);
Y1 = zeros(1,lenT);
E1(1) = mu; Y0(1) = E1(1); % initial values at time 0
%dt = 1;

% simulate path
for i = 1:lenT-1
    dt = T(i+1) - T(i); % dt can change from step to step
    temp = exp(-1/lambda0*dt);
    Y0(i+1) = mu*(1-temp) +  temp*Y0(i) + sigma*sqrt((1-temp^2)/2*lambda0)*
        randn;
    Y1(i+1) = exp(-1/lambda1*dt)*Y1(i) + L(i+1);
    E1(i+1) = Y0(i+1) + negJ*Y1(i+1);
end

% E1 contains exact simulation of the process
% the simulation times are contained in T
% E below contains the observations at times 0, 1, 2, ..., Nt

E = 0; j = 0;
for i=1:length(T)
    if T(i)==j
        E(j+1) = E1(i);
        j = j+1;
    end
end
```

F Simulation of a Model With Two Jump Components

```
function [E,LT1,LT2] = simulate3OUModel(Nt, mu, lambda0, sigma, lambda1,
    lambda2, jRate1, jRate2, beta1, beta2, negJ)
%SIMULATE3OUMODEL Simulate the 3-OU model X = Y0 + w1*Y1 + w2*Y2
%
% [E,LT1,LT2] = simulate3OUModel(Nt, mu, lambda0, sigma, lambda1,lambda2,
    jRate1, jRate2, beta1, beta2, negJ)
%
% dY0 = 1/lambda0*(mu-Y0)dt + sigma*dWt, Wt is a Wiener process
%
% dYi = -1/lambdai*Yi*dt + dLi, Li is a compound Poisson process with
% jump intensity parameters jRatei and Exp(betai) distributed jump sizes
%
% Input arguments:
%
%   Nt      - Number of timesteps, observation period is [0,Nt] with grid size
     1
%
%   mu      - Level of mean reversion of Y0
%
%   lambda0 - Time to mean reversion (inverse of speed of mean reversion)
%   of Y0
%
%   sigma   - Volatility
%
%   lambda1 - Time to mean reversion of Y1
%
%   lambda2 - Time to mean reversion of Y2
%
%   jRatei  - if Li is time-homogeneous, jRatei is a scalar representing
%       the jump intensity rate of Li. If Li is time-inhomogeneous jRatei is
```

```
28 %       a structure array containing the parameters for the intensity
29 %       function given in intensityFun.m, eg. jRatei.t0 = 130,
30 %       jRatei.period = 260, jRatei.delta = 0.15, jRatei.eta = 0.3
31 %
32 %   beta1   - Mean jump size of L1
33 %
34 %   beta2   - Mean jump size of L2
35 %
36 %   negJ  - 2-by-1 vector indicating whether Yi is positive (negJ(i) = 1)
37 %               or negative (negJ(i) = -1)
38 %
39 % Output arguments:
40 %
41 %   E       - Discretised sample path at grid times 0,1,...,Nt, E1 below is
42 %             the exact solution
43 %   LT1     - True jump process L1
44 %   LT2     - True jump process L2
45 %
46
47 % jump size distribution: 1 Pareto, 2 exponential, 3 Gamma
48 jumpDistribution = 2;
49 % generate jump times and jump sizes
50 if ~isstruct(jRate1)   % homogeneous
51     [L1,jumpTimes1] = homogeneousCPoisson(Nt,jRate1,beta1,jumpDistribution);
52 else                   % inhomogeneous
53     [L1,jumpTimes1]   = compoundPoisson(Nt, jRate1.t0, jRate1.period, ...
54         jRate1.delta, jRate1.eta, beta1, jumpDistribution);
55 end
56
57 if ~isstruct(jRate2)   % homogeneous
58     [L2,jumpTimes2] = homogeneousCPoisson(Nt,jRate2,beta2,jumpDistribution);
59 else
60     [L2,jumpTimes2]   = compoundPoisson(Nt, jRate2.t0, jRate2.period, ...
```

```
61          jRate2.delta, jRate2.eta, beta2, jumpDistribution);
62  end
63  % remove initial time 0
64  L1 = L1(2:end); jumpTimes1 = jumpTimes1(2:end); L2 = L2(2:end); jumpTimes2 =
        jumpTimes2(2:end);
65  % simulation times, combine fix intervals and jump times
66  [T,IX] = sort([0:Nt jumpTimes1 jumpTimes2]); %IX is the permutation vector
67  %L = [zeros(1,Nt) L L3(2:end)]; L= L(IX); %sort L over the 'combined' grid
68  % length of time grid including jump times and discrete times 0,1,...,Nt
69  lenT = length(T);
70  % process LT1 over this time grid
71  LT1 = [zeros(1,Nt+1) L1 zeros(1,length(jumpTimes2)); 0:Nt jumpTimes1
        jumpTimes2 ];
72  LT1 = sortrows(LT1',2)';
73  % process LT2 over this time grid
74  LT2 = [zeros(1,Nt+1) L2 zeros(1,length(jumpTimes1)); 0:Nt jumpTimes2
        jumpTimes1 ];
75  LT2 = sortrows(LT2',2)';
76  % allocate memory for process
77  E1 = zeros(1,lenT); Y0 = zeros(1,lenT); Y1 = zeros(1,lenT); Y2 = zeros(1,lenT)
        ;
78  % initial values at time 0
79  E1(1) = mu; Y0(1) = E1(1);
80  %dt = 1;
81  for i = 1:lenT-1
82      dt = T(i+1) - T(i);
83      temp = exp(-1/lambda0*dt);
84      Y0(i+1) = mu*(1-temp) +  temp*Y0(i) + sigma*sqrt((1-temp^2)/2*lambda0)*
            randn;
85      Y1(i+1) = exp(-1/lambda1*dt)*Y1(i) + LT1(1,i+1);
86      Y2(i+1) = exp(-1/lambda2*dt)*Y2(i) + LT2(1,i+1);
87      %print(i+1)
88      %print(lenT)
```

```
89      E1(i+1) = Y0(i+1) + negJ(1)*Y1(i+1) + negJ(2)*Y2(i+1);
90  end
91
92  % E1 contains exact simulation of the process, including jump times
93  % the simulation times are contained in T
94  % E below contains the observations at times 0, 1, 2, ... the days
95  %discretise(T,E1);
96  E = 0; j = 0;
97  for i = 1:length(T)
98      if T(i)==j
99          E(j+1) = E1(i);
100         j = j+1;
101     end
102 end
```

G Collection of the Data

```
#-----------------------------------------------------------
#
library(xts)
library(chron)
getDataForMonth <- function(newData){

  #Filter data by node name
  #newData = subset(data, pnodename == 'WEC.PLEASA142', select = c(mkthour,
      LMP))

  #transform data frame to time series

  newData[,1] = as.POSIXct(as.character(newData[,1]), format = "%m/%d/%Y %H:%M
      ")
  newData = na.omit(newData)

  newData = pad(newData)

  prices <- xts(newData[,-1], order.by = newData[,1], names= c("date","LMP"),
      include.weekends=FALSE)
  #subset(prices, format.Date(date,newData))
  #prices = as.zoo(prices, names= c("date","LMP"))
  prices = na.interpolation(prices, option = "linear")
  colnames(prices)<-c("LMP")
  #prices_daily = prices_daily[-nrow(prices_daily),]
  return(prices)
}
#-----------------------------------------------------------

#First get data
```

```
file = sprintf("C:/Uni/Master/Spring_18/Master/MISO Pricing Data/2013%02d.csv"
    , 1)
year = read.csv(file)
year = subset(year, pnodename == 'WEC.PLEASA142', select = c(mkthour, LMP))
#year = getDailyAverageDataForMonth(jan_data)

for(i in 2:12){
  file = sprintf("C:/Uni/Master/Spring_18/Master/MISO Pricing Data/2012%02d.
      csv", i)
  d = read.csv(file)
  d = subset(d, pnodename == 'WEC.PLEASA142', select = c(mkthour, LMP))
  year <- rbind(year,d)
  #month = getDailyAverageDataForMonth(data)
  #year = rbind.zoo(year,month)
  print(i)
}

for(i in 1:12){
  file = sprintf("C:/Uni/Master/Spring_18/Master/MISO Pricing Data/2013%02d.
      csv", i)
  d = read.csv(file)
  d = subset(d, pnodename == 'WEC.PLEASA142', select = c(mkthour, LMP))
  year <- rbind(year,d)
  #month = getDailyAverageDataForMonth(data)
  #year = rbind.zoo(year,month)
  print(i)
}

dailyPrices = getDataForMonth(year)

dailyPrices = dailyPrices[!weekdays(as.Date(index(dailyPrices))) %in% c("
    Samstag", "Sonntag")]

```

```
57 newTimeFormat = format(as.POSIXct(time(dailyPrices), format = "%y-%m-%d %H:%M"
       ), "%m/%d/%Y %H:%M")
58
59 p = data.frame(Date=newTimeFormat,Price=as.matrix(dailyPrices))
60
61 write.csv(p, file = 'C:/Uni/Master/Spring_18/Master/MISO Pricing Data/2012_13_
       WITHOUT_AVERAGE_Filtered.csv')
```

H Calculating the Average Data

```
#-----------------------------------------------------------
#
library(xts)
library(chron)
getDailyAverageDataForMonth <- function(newData){

#Filter data by node name
#newData = subset(data, pnodename == 'WEC.PLEASA142', select = c(mkthour, LMP))

#transform data frame to time series

newData[,1] = as.POSIXct(as.character(newData[,1]), format = "%m/%d/%Y %H:%M")
newData = na.omit(newData)

prices <- xts(newData[,-1], order.by = newData[,1], names= c("date","LMP"))
#subset(prices, format.Date(date,newData))
#prices = as.zoo(prices, names= c("date","LMP"))
colnames(prices)<-c("LMP")
#calcualte the average for each day
prices_daily = aggregate(prices$LMP,as.Date(index(prices)),mean)
prices_daily[.indexwday(prices_daily) %in% 1:5]
#prices_daily = prices_daily[-nrow(prices_daily),]
return(prices_daily)
}
#-----------------------------------------------------------

#First get data
file = sprintf("C:/Uni/Master/Spring_18/Master/MISO Pricing Data/2012%02d.csv", 1)
```

```
year = read.csv(file)
year = subset(year, pnodename == 'WEC.PLEASA142', select = c(mkthour, LMP))
#year = getDailyAverageDataForMonth(jan_data)

for(i in 2:12){
  file = sprintf("C:/Uni/Master/Spring_18/Master/MISO Pricing Data/2012%02d.
      csv", i)
  d = read.csv(file)
  d = subset(d, pnodename == 'WEC.PLEASA142', select = c(mkthour, LMP))
  year <- rbind(year,d)
  #month = getDailyAverageDataForMonth(data)
  #year = rbind.zoo(year,month)
  print(i)
}

for(i in 1:12){
  file = sprintf("C:/Uni/Master/Spring_18/Master/MISO Pricing Data/2013%02d.
      csv", i)
  d = read.csv(file)
  d = subset(d, pnodename == 'WEC.PLEASA142', select = c(mkthour, LMP))
  year <- rbind(year,d)
  #month = getDailyAverageDataForMonth(data)
  #year = rbind.zoo(year,month)
  print(i)
}

average = getDailyAverageDataForMonth(year)
average = average[!weekdays(as.Date(index(average))) %in% c("Samstag", "
    Sonntag")]

newTimeFormat = format(as.POSIXct(time(average), format = "%y-%m-%d"), "%m/%d/
    %Y")

```

```
58
59 p = data.frame(Date=newTimeFormat, Price=as.matrix(average))
60
61 write.csv(p, file = 'C:/Uni/Master/Spring_18/Master/MISO Pricing Data/2012_13_
       Filtered.csv')
```