Technical report, IDE1125 , August 21, 2011

Parameter Estimation of the Pareto-Beta Jump-Diffusion Model in Times of Catastrophe Crisis

Master's Thesis in Financial Mathematics

Wojciech Reducha

School of Information Science, Computer and Electrical Engineering
Halmstad University

Parameter Estimation of the Pareto-Beta Jump-Diffusion Model in Times of Catastrophe Crisis

Wojciech Reducha

Halmstad University
Project Report IDE1125

Master's thesis in Financial Mathematics, 15 ECTS credits

Supervisor: Ph.D. Jan-Olof Johansson
Examiner: Prof. Ljudmila A. Bordag
External referees: Prof. Vladimir Roubtsov

August 21, 2011

Department of Mathematics, Physics and Electrical engineering
School of Information Science, Computer and Electrical Engineering
Halmstad University

Preface

I would like to express my thanks to all my teachers, both in Gdańsk University of Technology and in Halmstad University who passed on their knowledge to me amd helped make this master thesis possible. I am especially grateful to Jan-Olof Johansson, my supervisor, who provided invaluable support and insight during these months.
Also, I'd like to thank my family and friends for their support in keeping the work going.

Abstract

Jump diffusion models are being used more and more often in financial applications. Consisting of a Brownian motion (with drift) and a jump component, such models have a number of parameters that have to be set at some level. Maximum Likelihood Estimation (MLE) turns out to be suitable for this task, however it is computationally demanding. For a complicated likelihood function it is seldom possible to find derivatives. The global maximum of a likelihood function defined for a jump diffusion model can however, be obtained by numerical methods. I chose to use the Bound Optimization BY Quadratic Approximation (BOBYQA) method which happened to be effective in this case. However, results of Maximum Likelihood Estimation (MLE) proved to be hard to interpret.

Contents

Chapter 1

Introduction

Recent developments in the field of jump-diffusion models (JD) have shown that this type of models is more suitable than those basing on just brownian motion. Consisting of three components - a linear drift, a Brownian motion which represents (normal) price variations and a compound Poisson process which is responsible for the impact of good and bad news on the price. Different approaches have been used to describe the "news" part of the process. For example, the Double Exponential Jump-Diffusion model (DEJD) proposed by Kou in 2002 [9] which covers the "news" part by using a Poisson process to invoke jumps of magnitude described by a double exponential distribution.

Much work has been done on the Jump-Diffusion (JD) models, including work of Merton (1976), who introduced the JD model, through Kou (2002), who developed the DEJD and Ramezani and Zeng (1998), who independently invented the Pareto-Beta Jump-Diffusion (PBJD) model, an equivalent to DEJD and many other papers discussing the application of these models. Most of them cover the area of pricing assets in relatively calm times, when both good and bad news rarely have an extremely significant effect on the prices. However, recent catastrophical events in Japan (earthquake and subsequent troubles in Fukushima I nuclear power plant in March, 2011) which might be the costliest in history (as per The Economist Online [20]) has provided some research opportunities in the field of modelling. The question is: how should the model be changed in order to make it fit better to the data from catastrophe affected markets. In my research I am using intra-day data of Nikkei225, OMX30, SEB A and Ericsson B from 21-12-2011 to 01-04-2011 obtained thanks to *Six Edge*$^{\text{TM}}$, professional software designed for purpose of intra-day data gathering and analysis.

In this paper, I try to use Maximum Likelihood Estimation (MLE) to estimate the parameters of a jump-diffusion model, namely the PBJD. Unfortu-

nately, the likelihood function of the process is too complicated to maximize by analytical methods, therefore numerical methods had to be employed (see Ramezani and Zeng, 2007). To do this, I decided to use the Bound Optimization BY Quadratic Approximation (BOBYQA) method developed recently by Powell (2009), thus providing test ground for this relatively new algorithm. More detailed description of the method is given in Chapter 4. My another aim was to show how high frequency of data can affect the estimation of the parameters. To see this, I use 5-minute, 10-minute, 20-minute and 30-minute data. It is the consistent property of MLE that ensures us that more data points give us more precise results, however at a cost. At the same time, for financial data, higher frequency data may be affected by "high frequency noise", see Aït-Sahalia (2004) (time-smoothing effect). The question that I am trying to answer with these considerations is: are the improved results provided by higher frequency and the PBJD model worth the computational effort?

The paper is organized as follows. In Chapter 2, I present some models (along with the Pareto-Beta Jump Diffusion, which is the main hero of the thesis) that were and are used for pricing financial assets, such as stocks and their specifications, limitations and advantages. I also present a few calibration (parameter estimation) methods. In Chapter 3, I go into detail on the Pareto-Beta Jump Diffusion model, introduced by Ramezani and Zeng (1998), and its connection to the popular Double Exponential Jump Diffusion model. Chapter 4 covers the issues of numerical estimation of parameters and its results for real market data. The conclusions, suggestions for further research and improvement are provided in Chapter 5 along with some thoughts on usefulness of PBJD.

Chapter 2

Models

Since the beginning of trade with stocks, it was very important for investors to have a way to predict price movements to some degree. The aim was, of course, increasing profit and decreasing losses during bad times. While acting on just hunch and economic analysis works for at least some investors, having another tool at one's disposal in the form of mathematical models is a huge help. Due to this, a new branch of mathematics, known as theory of pricing, emerged. Over the last few decades many models have been invented, developed and improved thus increasing their complexity. I shall describe some of these models in this chapter.

I shall first present different models that were developed for pricing stocks from the simple Brownian Motion (BM), through Geometric Browniam Motion (GBM), Jump Diffusion (JD) to the model which is the main point of this thesis, that is the Pareto-Beta Jump-Diffusion model(PBJD).

2.1 Models based on the Brownian Motion

In this section I shall present models based mainly on the Brownian Motion. I present the very simple Brownian Motion model, then the Geometrical Brownian Motion and the Generalized Hyperbolic Brownian Motion models as extensions.

2.1.1 The Brownian Motion

One of the first works dealing with brownian motion in the stock market is the paper of Osborne (1959). The equation for development of the stock price in time t in this model is

$$S_t = S_0 + \mu t + \sigma B_t,$$

where S_0 is the initial price at time 0, μ is the price drift, σ is the price volatility and B_t is a standard Brownian motion.
This model is very simple and easy to simulate. It also provides some explanation for the price movements. Naturally, this model is too simple to be accurate and thus, should not be used for modelling prices. Also, it has a serious drawback of being able to return negative stock prices which is obviously impossible.

2.1.2 The Geometric Brownian Motion

The Geometric Brownian Motion (GBM) also known as exponential Brownian motion is the most widely used model for modelling stock prices and is also the basis for the extremely popular Black-Scholes model. It satisfies the stochastic differential equation (SDE)

$$dS_t = \mu S_t dt + \sigma S_t dB_t,$$

where S_t is the price (a stochastic process), B_t is the standard Brownian motion and μ (percentage drift) and σ (percentage volatility) are constants. The above SDE for an arbitrary starting price S_0 has the solution

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

GBM does not have the main weakness of the previous model which is the possibility of returning negative price values. It is also relatively easy to make calculations with GBM. These advantages, among others mentioned in Hull (2009), have made GBM very popular and most widely used model in finance. GBM naturally has weaknesses. One of them is the assumption of the normal distribution of returns which are actually affected with the higher kurtosis.
The GBM model is inferior to the Jump Diffusion models I shall present in the next section (which was proven in Ramezani and Zeng (2007), for example) but despite this, it still retains high popularity mainly due to ease of use.

2.2 Jump Diffusion models

In this section I describe a few Jump-Diffusion (JD) models, starting with the Merton JD model, the first of this type of asset pricing models to appear. I shall then describe Kou's model - the Double Exponential Jump-Diffusion (DEJD) and finally, the model of interest of this thesis work, the Pareto-Beta Jump-Diffusion model, developed by Ramezani and Zeng (1998).

2.2.1 The Merton model

The Jump Diffusion model, first introduced by Robert Merton (1976), describes the behaviour of stock prices by taking into account a price drift, small day-to-day movements (diffusive element) and price jumps, caused by political, social, economical or natural (e.g. catastrophes) events. The asset price under this model is described by the equation

$$S(t) = S(0) \exp\left\{(\mu - \frac{1}{2}\sigma^2)t + \sigma B(t)\right\} \prod_{i=1}^{N(t)} e^{Y_i}, \tag{2.1}$$

where $N(t)$ is a Poisson process.
$Y_i,\ i = 1, \ldots, N(t)$ are iid normally distributed random variables with the density function

$$f_{Y_i}(y) = \frac{1}{\sigma'\sqrt{2\pi}} \exp\left\{-\frac{(y-\mu')^2}{2\sigma'}\right\}, \tag{2.2}$$

where μ' and σ' are the mean and the standard deviation of Y_i.

2.2.2 The Kou model

The Kou model, also known as the Double Exponential Jump Diffusion (DEJD) model, is similar to the Merton model. The difference however, lies in the assumption regarding the jumps. While in Merton's model the jumps are log-normally distributed, Kou (2002) decided to use the double exponential distribution for this purpose. Its density function is

$$f_{Y_i}(y) = p \cdot \eta_1 e^{-\eta_1 y} \mathbb{1}_{y \geq 0} + q \cdot \eta_2 e^{\eta_2 y} \mathbb{1}_{y<0}, \tag{2.3}$$

where $\eta_1 > 1;\ \eta_2 > 0;\ p, q \geq 0;\ p + q = 1$. Here p is the probability of an upward jump while q is the probability of a downward jump occuring.
It is worth noting that Kou's model gives more control over the jumps' intensities and magnitudes as its parameters can be easily interpreted in contrast to the normal distribution used in Merton's model where both good and bad news are covered by just one random variable whose parameters are rather hard to interpret. Also, this approach allows the DEJD model to model skewness, a feature that Merton's model lacks.

2.2.3 The Pareto-Beta Jump-Diffusion model

Another model in the JD family, Pareto-Beta Jump Diffusion (PBJD) developed by Ramezani and Zeng (1998) is similar to Kou's model but has a

different approach towards jumps. While Kou proposed using two exponential distributions with different parameters to describe the jumps, Ramezani and Zeng assume that both good and bad news are generated by two independent Poisson processes with different intensities and that jump magnitudes are drawn from Pareto and Beta distributions, respectively.
The PBJD model and its connection to the DEJD model is described in detail in Chapter 3.

2.3 The calibration procedure

The calibration of a model is nothing else as parameter estimation based on data, in our case market data. During the calibration we assume that the analyzed data set follows a model but we do not know its parameters and therefore, we have to estimate them in order to obtain full model specification. There are many different methods of the parameter estimation. I shall present some of them in this section.

2.3.1 The Generalized Moments Method

The generalized moments method (GMM) is a parameter estimation method developed by Hansen (1982). It is usually applied in models where maximum likelihood estimation cannot be applied for various reasons. GMM requires some moment conditions to be specified for the estimated model. These conditions are functions of data and the model's parameters, such that the expectations are zero at the true values of the parameters. The trick of the GMM method is to minimize a norm of the sample averages of the moment conditions.
Aït-Sahalia (2004) extensively describes the method in his paper and its application to Merton's Jump Diffusion model with Poisson jumps. He also shows, on example of Cauchy jumps, that GMM can be applied to other types of Lévy pure jump processes.
An interesting conclusion that can be drawn from Aït-Sahalia's work is that GMM estimators using absolute moment of various noninteger orders are not as efficient as maximum likelihood estimators (MLE) but are in turn better than traditional moments such as a variance (second moment) and a kurtosis (fourth moment).

2.3.2 The Indirect Inference

The Indirect Inference is a method which can be used for extremely complex models for which a direct approach is virtually impossible. This method bases on usage of an additional auxiliary model which is simpler and easier to work with. Extensive description and application proposals in microeconomics, finance and macroeconomics are available in Gourieroux, Monfort, Renault (1993).

This method can be presented in short as follows. First, the data is used to obtain parameter estimations of the auxiliary model and then these parameters are inserted (and converted if necessary) into the primary model. With these parameters, the primary model is used to simulate another data set. This set is then employed to estimate the auxiliary model's parameters again. So it is an alternating method. The estimation is terminated when the newest parameter set is close enough to the previous one. This approach has many applications but is, however very costly in computation since many simulations of the primary model and parameter estimations of the auxiliary model have to be carried out.

2.3.3 The Maximum Likelihood Estimation

The Maximum Likelihood Estimation (MLE) could be generally described as a procedure which, for a given model, maximizes a function (likelihood) of fixed data and the same parameters as in the model and thus finds the parametric values which are most probable (or likely), i.e. return the highest value of the likelihood function. MLE provides us with an unified approach to the estimation which is well defined in the case of normal distributions and many other problems, including the one in this thesis. Also, MLE has many desirable properties, these are described in more detail in Chapter 4.

Chapter 3

The Pareto-Beta Jump Diffusion model

In this chapter I shall present the Pareto-Beta Jump Diffusion (PBJD) model of Ramezani and Zeng (1998) on which my thesis is based. It's a jump diffusion model which assumes that the good and bad news are generated by two different independent Poisson processes where the magnitudes of the jump are drawn from Pareto and Beta distributions, respectively.

3.1 Properties

The reason behind discerning good news from the bad ones has an economic justification. Firstly, people react differently to good and bad news. A few times in history stocks have suffered from extreme drops of value, take for example the Dow Jones crash of 1987, when Dow Jones Industrial Average lost 22.61% in just one day. When observing the history of returns of other large world indexes, one can notice that spectacular losses occured more often than spectacular gains. The reason behind this could be human irrationality - investors (non-investors as well) are more likely to panic and thus cause a disastrous chain reaction than fall into mass euphoria, at least in short time. An example of such behaviour can be the so-called internet bubble (also known as dot-com bubble) phenomenon which occured in the end of the previous century. The prices of IT sector companies were rising relatively quickly over a few years when in the year 2000, the bubble popped. Another example of a similar event is the recent financial crisis, which could be refered to as a "loan bubble".

Making a disctinction between good and bad news implies that there should be different ranges of values in changes of price in both cases. Good

news obviously should generate only positive values of price change while bad news, due to limited liability posed by stocks, must be bounded from below by -100% (common sense also suggests that it should be bounded from above by 0). These constraints therefore limit the choice of distributions for the jump magnitudes. Under the PBJD, good news are drawn from Pareto distribution, which is supported in the interval $[1, \infty]$ and is also concentrated near 1, which relates to the "Pareto principle" also known as the "80-20 law" saying that 20% of all people possess 80% of all wealth. In our case, we can interpret it that among the good news, a large amount of them have a small effect on the price and a small amount in turn have a tremendous effect, e.g. discovering large oil deposits by an energy company - an event that doesn't happen often but has a huge impact on the stock price of such company. Similarly, for bad news we use the Beta distribution which is supported in the interval $[0, 1]$. In addition to these economic arguments, Pareto and Beta distributions provide a tractable likelihood function and therefore facilitate MLE.

3.2 The Model Specification

Let $S(t)$ denote the price of stock at time t and assume that the price process can be represented as

$$\frac{dS(t)}{S(t-)} = \mu dt + \sigma dZ(t) + \sum_{j=u,d} (V^j_{N^j(\lambda^j t)} - 1) dN^j(\lambda^j t), \tag{3.1}$$

where μ is the drift, σ is the volatility term, $Z(t)$ is a standard Wiener process, V^j is the jump magnitude and $N^j(\lambda^j)$ are independent Poisson process with intensities λ^j - here $j = u, d$ represent the up- and down- jumps, respectively. The assumption about the up-jump magnitudes (V^u) is that they are Pareto(η_u) distributed (actually, Pareto distribution has 2 parameters but one of them is the starting point of the interval where density is larger than zero which in our case is 1) with density function

$$f_{V^u}(x) = \frac{\eta_u}{x^{\eta_u+1}} \quad V^u \geq 1,$$

$$E(V^u) = \frac{\eta_u}{\eta_u + 1} \quad \text{and} \quad \sigma^2_{V^u} = \frac{\eta_u}{(\eta_u - 2)(\eta_u - 1)^2}.$$

For the down-jumps magnitudes (V^d) it is assumed that they are Beta($\eta_d, 1$) distributed with density function

$$f_{V^d}(x) = \eta_d x^{\eta_d - 1} \quad 0 < V^d < 1,$$

$$E(V^d) = \frac{\eta_d}{\eta_d + 1} \quad \text{and} \quad \sigma^2_{V^d} = \frac{\eta_d}{(\eta_u + 2)(\eta_u + 1)^2}.$$

All jumps are assumed to be independent, which implies a mixture of Pareto-Beta distributions for jump magnitudes. We can see that the model's specification in (3.1) is a Lévy process and it can be decomposed as a sum of three independent components: the linear drift, a Brownian motion and pure jump process. Also, it has stationary and independent increments and is continuous in probability as proven in Cont and Tankov (2003).

By using the Doléans-Dade formula (Protter, 1991) we obtain an explicit solution for (3.1)

$$S(t) = S(0) \exp\left\{\left(\mu - \frac{1}{2}\sigma^2\right)t + \sigma Z(t)\right\} \prod_{j=u,d} V^j(N(\lambda^j t)) \tag{3.2}$$

$$\prod_{j=u,d} V^j(N(\lambda^j t)) = \begin{cases} 1 & \text{if } N(\lambda^j t) = 0 \\ \prod_{i=1}^{N(\lambda^j t)} V_i^j & \text{if } N(\lambda^j t) = 1, 2, 3, \ldots \end{cases}$$

Now let $Y = \ln(V)$, then the s period rate of return, $r(s)$, can be written as

$$r(s) = \left(\mu - \frac{1}{2}\sigma^2\right)s + \sigma Z(s) + \sum_{i=1}^{N_s^u} Y_i^u + \sum_{i=1}^{N_s^d} Y_i^d, \tag{3.3}$$

where N_s^j, $j = u$ or d denotes the number of good (up) and bad (down) news over the time period s. Ramezani and Zeng (2007) show that $r(s)$ is probability weighted mixture of normal and exponential distributions. This density function will be used for maximum likelihood estimation (MLE).

It is possible to establish a connection between PBJD and DEJD. This can be done as follows. Let $\lambda = \lambda_d + \lambda_u$, $p = \frac{\lambda_u}{\lambda}$, and $q = 1 - p$, then the distribution of jump magnitudes ($\tilde{V}$) is actually a probability weighted mixture of Pareto and Beta given by

$$f_{\tilde{V}}(x) = p\frac{\eta_u}{x^{\eta_u+1}}\mathbb{1}_{x>1} + q\eta_d x^{\eta_d - 1}\mathbb{1}_{0<x<1},$$

where $\eta_u > 1$, $\eta_d > 0$. Now, setting $Y = \ln(\tilde{V})$ and taking into consideration that distribution of logarithms of Pareto and Beta are exponential (see Ramezani and Zeng (2007)), we obtain

$$f_Y(y) = p\eta_u e^{-\eta_u y}\mathbb{1}_{y\geq 0} + q\eta_d e^{\eta_d y}\mathbb{1}_{y<0}.$$

This is the formula used in DEJD for the distribution of logarithm of jump magnitudes with intensity λ and also Y has an IID mixture distribution of

exponential distributions with parameters η_u and η_d with probabilities p and q, respectively. It is worth noting that both models have the same number of parameters to estimate, i.e. $\theta_{DEJD} = (\mu, \sigma, \lambda, p, \eta_u, \eta_d)$ and $\theta_{PBJD} = (\mu, \sigma, \lambda_u, \lambda_d, \eta_u, \eta_d)$. Now that a connection between the two models has been established, I will from now on use "DEJD" to refer to both.

Chapter 4

Application of the Pareto-Beta Jump Diffusion model

4.1 The Maximum Likelihood Estimation

There are several viable methods for parameter the estimation of Jump Diffusion processes, including the generalized moment of methods (described in detail in Aït-Sahalia (2004)), the simulated moment estimation and MCMC methods are surveyed among others in Aït-Sahalia and Hansen (2004). My method of choice is MLE because of its desirable statistical properties. A detailed description of MLE estimation of jump diffusion processes can be found in Sorensen (1991), who proves that MLE is the best estimation method for large samples. Since I work on large (intra-day data) samples in this thesis, MLE estimation fits neatly in my case. The reason behind MLE's great effectiveness is that under mild regularity conditions, the estimated parameters are consistent, asymptotically normal and asymptotically efficient - see Hamilton (1994). MLE does have a drawback of requiring complete specification of the transition density which is difficult to obtain for nonlinear models. Luckily, this is not a problem in the case of DEJD which is a linear process with an explicit transition density and independent increments. Moreover, the distributions of the jump components make MLE tractable and lastly, as I mentioned earlier, Aït-Sahalia (2004) has shown that MLE offers many advantages in disentangling jumps from diffusion.

Let $D = \{S(0), S(1), S(2), \ldots, S(M)\}$ denote the realizations of stock price at equally spaced times $k = 0, 1, 2, \ldots, M$. The one period rate of return $r_i = \ln S(i) - \ln S(i-1)$ is IID. As Ramezani and Zeng (2007) have

shown, the unconditional density of s period returns, $f_{r(s)}(r)$ is

$$f_{r(s)}(r) = e^{-(\lambda_u+\lambda_d)} f_{r(s)|0,0}(r) + e^{-\lambda_u} \sum_{n=1}^{\infty} P(n,\lambda_d) f_{r(s)|0,n}(r) +$$
$$+e^{-\lambda_d} \sum_{m=1}^{\infty} P(m,\lambda_u) f_{r(s)|m,0}(r) + \sum_{m=1}^{\infty}\sum_{n=1}^{\infty} P(n,\lambda_d)P(m,\lambda_u) f_{r(s)|n,m}(r),$$

where $P(n,\lambda) = \frac{e^{-\lambda}\lambda^n}{n!}$ and $f_{r(s)|m,n}(r)$ ($n \geq 0$ and $m \geq 0$) is the conditional density for one period returns depending on the given numbers of up and down jumps (m,n). The formulas for these densities are

$$f_{r(s)|0,0}(r) = \frac{1}{\sqrt{2\pi s}\sigma} \exp\left\{-\frac{(r-\mu s+0.5\sigma^2 s)^2}{2\sigma^2 s}\right\},$$

$$f_{r(s)|0,n}(r) = \frac{\eta_d^n}{(n-1)!\sqrt{2\pi s}\sigma} \int_{-\infty}^{0} (-x)^{n-1} \exp\left\{-\eta_d x - \frac{(r-x-\mu s+0.5\sigma^2 s)^2}{2\sigma^2 s}\right\} dx,$$

$$f_{r(s)|m,0}(r) = \frac{\eta_u^m}{(m-1)!\sqrt{2\pi s}\sigma} \int_{0}^{\infty} x^{m-1} \exp\left\{-\eta_u x - \frac{(r-x-\mu s+0.5\sigma^2 s)^2}{2\sigma^2 s}\right\} dx,$$

$$f_{r(s)|m,n}(r) = \frac{\eta_u^m \eta_d^n}{(m-1)!(n-1)!\sqrt{2\pi s}\sigma} \int_{-\infty}^{\infty} \left(\int_{-\infty}^{0\wedge t} (-x)^{n-1}(t-x)^{m-1} e^{(\eta_u+\eta_d)x} dx \right) \cdot$$
$$\cdot \exp\left\{-\eta_u t - \frac{(r-t-\mu s+0.5\sigma^2 s)^2}{2\sigma^2 s}\right\} dt.$$

The log-likelihood given M equally spaced returns observations is

$$L(D;\lambda_u,\lambda_d,\eta_u,\eta_d,\mu,\sigma^2) = \sum_{i=1}^{\infty} \ln(f(r_i)). \tag{4.1}$$

So, the unconditional density $f(r)$ is actually a mixture density - a Poisson weighted sum of four conditional densities. As was shown by Honoré, care must be taken with mixture densities to keep the log-likelihood bounded as a singularity problem can arise and the function becomes infinite. This problem

can be dealt with by properly bounding the admissible parameter space. In the case of DEJD, as long as σ is kept away from zero, the likelihood function should not explode and the obtained MLE remains asymptotically normal and consistent. Fortunately, the BOBYQA algorithm used for maximization of the log-likelihood supports lower and upper bounds and thus ensuring proper restrictions of the parameter space.

The density function f in (4.1) poses a formidable computational effort, further magnified by the fact that there are many data points to analyze due to intra-day data. The function involves double improper integrals and double infinite summations which only makes it more costly to calculate. For the latter I used a well known termination criterion. Let $S_n = \sum_{i=1}^{n} X_i$, then the summation will be stopped when $2|X_{n+1}| \leq FTOL \cdot (|S_n| + |S_{n+1}|)$. For the calculation I have chosen $FTOL$ (acronym for Fractional TOLerance) to be equal 10^{-8}, which according to Ramezani and Zeng (2007) guarantees at least six digit accuracy.
As for integrals, gaussian quadratures have been employed - based on QGAUS procedure by Press et. al (1992). Tests and visualisations made in *Mathematica* have shown that it is safe to truncate the integrals to $[-2, 0]$ (symmetrical where needed) for plausible parameter values - range of plausible values are taken from results of Ramezani and Zeng (2007). Actually, as visualisations from *Mathematica* show, even a $[-0.5, 0]$ truncaction could be employed for a great majority of cases. However, due to concerns about accuracy I decided to modify the QGAUS procedure. The reason was the shape of the $f_{0,n}$ and $f_{m,0}$ functions which can be either neat as in Figures 4.1 and 4.2 or, on the contrary, look like thin and high "blades" as in Figures 4.3 and 4.4. This could lead to interpolation points of the gaussian quadrature to "not hit" the function where it is actually noteworthy. To deal with this problem, I made a simple detection algorithm. The detection routine starts at 0 and proceeds in a proper direction by a small step. On every step, the value of the function is compared with a tiny numerical value which in this case I decided to be 10^{-14}. If the function value is larger than 10^{-14}, then a proper point is assigned as the beginning of the integration interval. Next, the end of the "hill" is searched for in a similar manner. If there is no "hill" detected, the value of the integral is returned as 0. Thanks to this detection algorithm, the integral is calculated only where it makes sense. For the single integrals, I use 16 interpolation points.

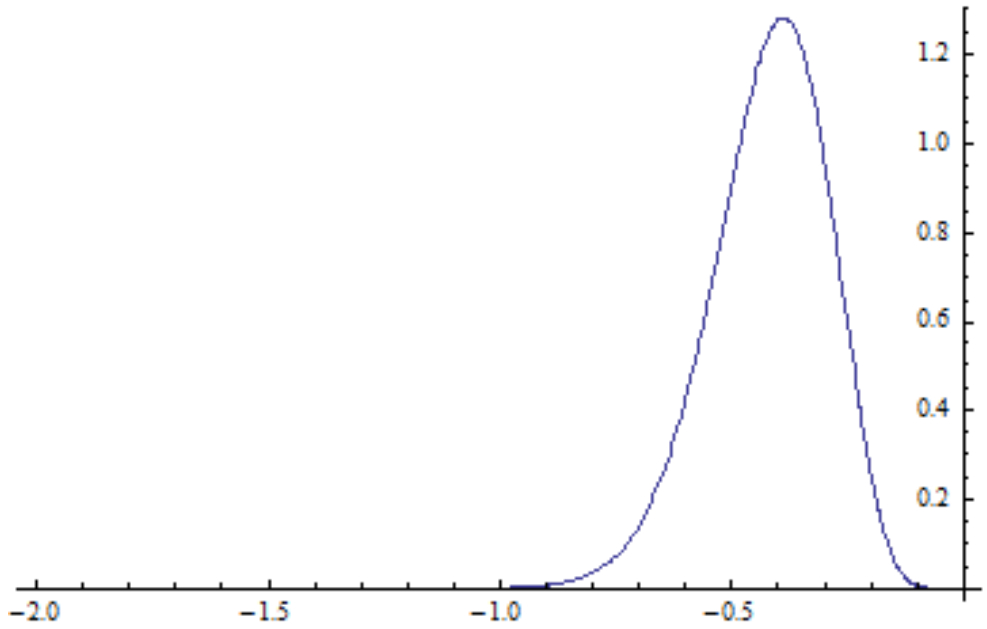

Figure 4.1: Plot of the integrand in the function $f_{r(s)|0,n}(r)$ in $(-2, 0)$ for parameters $\eta_d = 16.8, \mu = 0.0332, \sigma = 0.0311, n = 9, r = -0.036$

.

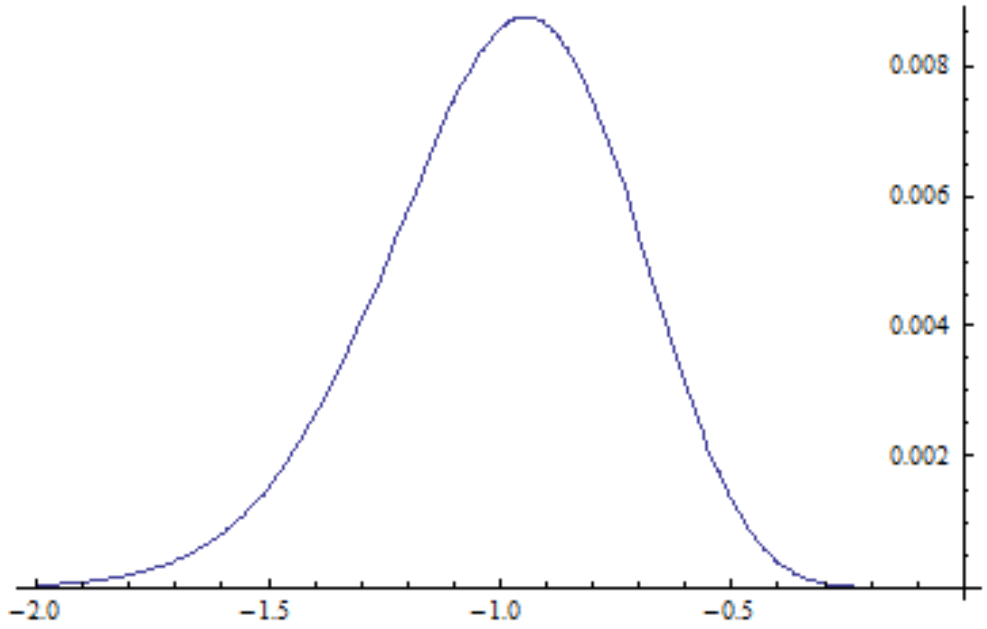

Figure 4.2: Plot of the integrand in the function $f_{r(s)|0,n}(r)$ in $(-2, 0)$ for parameters $\eta_d = 5, \mu = -0.007, \sigma = 0.5, n = 10, r = 0.057$

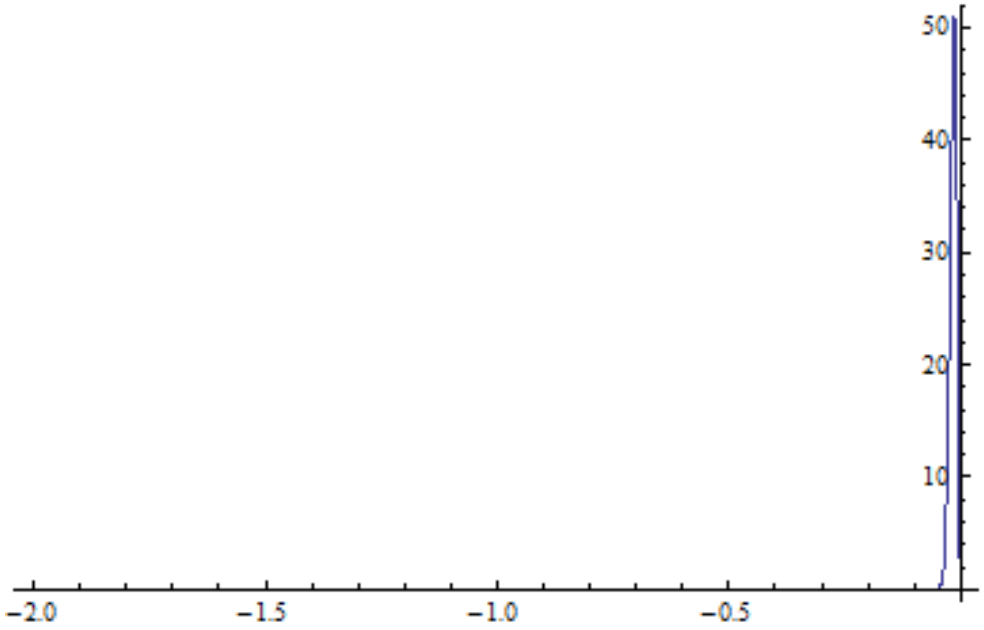

Figure 4.3: Plot of the integrand in the function $f_{r(s)|0,n}(r)$ in $(-2, 0)$ for parameters $\eta_d = 39.4, \mu = 0.0068, \sigma = 0.012, n = 3, r = 0.008$

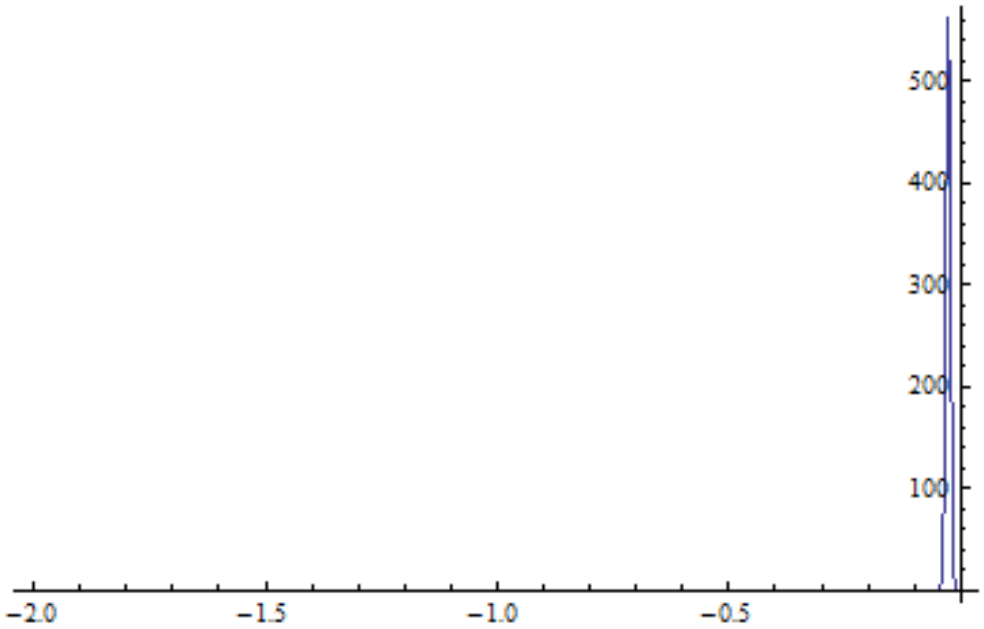

Figure 4.4: Plot of the integrand in the function $f_{r(s)|0,n}(r)$ in $(-2, 0)$ for parameters $\eta_d = 79.8, \mu = 0.003, \sigma = 0.0055, n = 5, r = -0.022$

The real challenge came with the double improper integrals. Here, however, using *Mathematica* again, I managed to find good truncations for a plausible range of parameter values. To make notation more convenient, from now on I shall use the $\wedge$ instead of $min()$, i.e. $a \wedge b$ $min(a, b)$. As it turns out, it is safe to truncate the integration to $[-2; 2]$ for t and $[-2.5; 0 \wedge t]$ for x. This time however, I decided not to implement the detection algorithm mentioned earlier and just turned to the piecewise integration. To be more specific, the program makes steps of 0.2 for t but does not do that for x, which is calculated on the whole $[2.5; 0 \wedge t]$ interval. To compensate for this lack of piecewise integration in the case of x, a large number of interpolation points is used for this coordinate, namely 32 while for t it is 20 to save computation time. The algorithm this time was based on QGAUS, but the multidimensional implementation of the gaussian quadrature was based on an algorithm by Mitra [11] so as to be able to use a different number of interpolation points for each variable. An issue that arised is that sometimes x is stepping over the $0 \wedge t$ bound and therefore returned large negative values (which should never appear due to the conditions imposed by the model and the density function). This problem has been dealt with by simply defining the function as equal to zero in those rare occurences of negative values.

To save computation time, I used a small trick to prevent repeated calculations of the density function for the same sets of parameters. The opportunity appears in the calculation of (4.1). Since the single parameter for f is r_i - the i-th return, it is worth noting that the value of L does not depend on the order of returns because it is just a summation of logarithms. Order is important for stock data to obtain returns but not for returns themselves. I decided to use this property to my advantage and optimize the program in the following way.

First, I use the native C function *qsort* to sort the array of returns. Quicksort is a method of the average computational complexity of $O(n \ln n)$ - cost worth bearing considering the benefits.

After this, when calculating the density function for r_i, it is checked if $r_i = r_{i-1}$. If this is true, instead of calculating $f(r_i)$ from the beginning, $f(r_{i-1})$, the previous result which is always saved in the memory is used. This optimization caused a decrease of calculations ranging from 0 to as much as 78 percent which obviously drastically reduced computation time.
Further improvement of the calculation time was connected with using the full potential of the available computers. Since they have dual-core processors, it seemed reasonable to try applying multi-threading into my program. However, the solution proved to be much simpler - since each instance of the program uses only one core and I had two cores at my disposal it was enough to just launch two instances of the program on one computer, therefore us-

ing the CPU to its full potential. However, even with these optimizations in place, the estimation times ranged from 20 minutes to as much as 6 hours for a single initial parameter set, thus limiting my abilities to launch a full battery of tests.

4.2 The programming method selection

Here I shall present different methods I attempted to apply in order to obtain the MLE estimation of parameters of the DEJD model. I write them here so that others may benefit from my experience and avoid some dead ends.

At first, since the estimation program requires many improper integration calculations and double improper integrations, I decided to employ *Wolfram Mathematica*™for my purposes, particularly the NIntegrate routine which proved to be relatively quick and easy to program. Also, there was no need to worry about proper truncation of the integral since *Mathematica* can handle this issue well. Next, I attempted to use *Mathematica*'s numerical maximization routine NMaximize to obtain the parameter estimates. This time, however, it turned out that NMaximize cannot offer satisfying methods for my purpose. That, along with the fact that NMaximize apparently does not work with functions which contain NIntegrate (errors occur), made me abandon this path and pick another way to maximize the log-likelihood function. Following in the steps of Ramezani and Zeng (2007) I tried to use Powell's method. The reason for not choosing another popular method for likelihood optimization, the Newton-Raphson method, was the requirement of first and second order derivatives of the log-likelihood function which are very difficult to obtain for DEJD. Since I still wanted to take advantage of *Mathematica*'s NIntegrate, I translated the Powell's method's C code which I obtained from J.P. Moreau's web page [12]. Unfortunately, Powell's method does not work with constraints on the parameters which, what was discovered during tests, can lead to situations where the likelihood function returned complex values due to the density function (which is being logarithmized) being negative, which in turn is caused by some of the parameters being negative while testing (in DEJD, only μ can be negative) and thus, corrupting the maximization procedure as a whole.

This led me to searching for a method of constrained optimization without the use of derivatives and I have found a method recently developed by Powell (2009) called BOBYQA (Bound Optimization BY Quadratic Approximation). It is an iterative algorithm for optimizing a function $F(\bar{x}), \bar{x} \in \mathbb{R}^n$, subject to bounds $\bar{a} \leq \bar{x} \leq \bar{b}$ on the variables based on Powell's other algorithm called NEW Unconstrained Optimization Algorithm (NEWUOA).

Each iteration applies a quadratic approximation Q to F with m interpolation points being adjusted automatically. The number m is a constant given by the user and is typically set to $m = 2n+1$ and so it is set in my program's case. Detailed description of BOBYQA is given in Powell (2009).

Due to concerns about the program's running time, I also decided to abandon *Mathematica* in favor of C language. I did however, use *Mathematica* it as a benchmark for the precision of calculations of integrals. I based my integration algorithm on "QGAUS" procedure from Press et. al (1992). As to the optimization procedure, I decided to employ the *dlib* library [23], which has BOBYQA routine translated from Fortran to C and implemented among many other algorithms.

4.3 The Market data

4.3.1 Description of the data

The real market data I have decided to analyze comes from the time period of 2011-02-21 to 2011-04-01. Also, I have split this data into two even parts - from 2011-02-21 to 2011-03-11 and from 2011-03-14 to 2011-04-01. The reason behind this operation is the tremendous earthquake in Japan which took place during the weekend between 2011-03-11 (Friday) and 2011-03-14 (Monday) and to make the "before" and "after" parts provide equivalent results for both time periods.
I have picked two stock indexes and two stocks for analysis. The first, most obvious choice was Nikkei225 index in Tokio since it could provide interesting results connected with the catastrophe. The other stock index was OMX30 in Stockholm since the availability of intra-day data from *SixEdge*^{TM} is limited mainly to Nordic markets. Also, it presents an opportunity to study the impact of major catastophes on this region's markets. One of the stocks I have picked is Ericsson B - it's a well known telecommunication company cooperating with Sony (a Japanese company) in the field of cell phones. I expected to see an influence of the damage caused by the earthquake to Sony facilities on Ericsson's stock value. Another Swedish stock considered was SEB A - a company from the banking sector.
As mentioned earlier, the high frequency data was employed to provide good MLE estimation since it benefits greatly from a high number of data points. Thus, I used 5-, 10-, 20- and 30-minute intervals which provided 1530, 765, 390 and 255 data points, respectively for OMX30, Ericsson B and SEB A. In the case of Nikkei225, due to different working hours, the respective amounts are 1380, 690, 345 and 240. Hourly and daily data was not used due to

the small number of data points it would have provided. Also, for every time interval the value of s was changed to bring the results in line with the daily data. This value was obtained by simply counting the number of corresponding data points each day.

4.3.2 Results

The results of the computer experiments are presented in the appendix Table 5.1 to Table 5.8. Each represents a stock or index in period before and after the earthquake. The first column denotes the time interval of analyzed data, second the parameters of the model: λ_d, λ_u, η_d, η_u, μ, σ described earlier and also the value of the maximized likelihood function, denoted by V. The next 10 columns show the maximization results for 10 different sets of initial parameters. The bold sets are the ones with the highest value of the likelihood function, i.e. the most likely.

The first look at the numerical results clearly says that the likelihood function has many local maxima. It can be deducted from the fact that there is no pair of identical or even close results for any of the initial parameter sets. This points to a conclusion that one has to run many experiments with different initial parameters and basically get a good hit to obtain a satisfying result. For now it is hard to tell if the 10 initial sets I have used are enough. Another fact worth noting is that in some cases, the V values of "non-bold sets" are quite close to those of the "bold sets" while having very different estimations of the model parameters (e.g. table 5.1, 20-minute interval, column no. 1 or 5.4, 30-minute interval, column no. 6). This emphasises how important it is to run many experiments to obtain a good result and that some phenomenons can be deceptively explained almost equally well by different parameters, e.g. combinations of jump intensities and means of the jump magnitudes.

Quite interestingly the second set returned the most likely result for many cases, but only for 10, 20 and 30 minute intervals, while for every single 5-minute interval experiment, the result is disastrously low which either points for a very bad initial guess or a trap BOBYQA fell into. I refrained from collecting all the results and deriving the mean and standard deviation since the range of these is quite large and some of the results are clearly inferior in terms of likelihood and so discarding them seems to be a reasonable course of action. In my interpretations of the results I decided to compare the "before" and "after" parts using the same interval between returns in order to work on equivalent data. Parameters should be interpreted as follows. μ and σ are well known parameters presenting drift and volatility, respectively. λ_d and λ_u are jump intensities, the higher the number, the higher the intensity, i.e.

λ_d^{-1} is the average time between down jumps. η_d and η_u are respective jump magnitudes and in this case, the lower the number, the higher the magnitude, i.e. eta_d^{-1} is the percentage magnitude of the jump.

I shall now proceed to analysis of results for the particular indexes and stocks and begin with Tokio's Nikkei225 - the most relevant index to the Japanese earthquake event. One observation is that σ in all cases has approximately doubled. This is an expected outcome, seeing that the times shortly after the earthquake were full of uncertainty and emotions. Results concerning μ, the drift, are not so consistent but one can see that the downfalls are actually larger than risings and so the conclusion can be driven, that μ has fallen because of the earthquake. This can be explained as an effect of pessimism and distress the investors went through. As to down jumps, it is hard to derive the conclusion about λ_d, their intensity but it is, similar with η_d, although one can see that if η_d falls, it is a big fall, contrary with the up movement of η_d. This could mean that in the times of weakness, the market is very susceptible to any bad news and reacts more harshly than in normal times. With the up jumps, the situation is clearer - jumps are generally more frequent (λ_u rises) while η_u stays at the same level, on average. This could mean that investors were more likely to accept more news as a glimmer of hope and react accordingly. Overall, Nikkei225 reacted to the earthquake by becoming more jumpy and more volatile.

Next, I shall move to the Nordic markets and analyze Stockholm's OMX30 results. The volatility has increased slightly after the earthquake while the drift dropped but still remains positive. This could be viewed as a damaging factor caused by the earthquake. A conclusion that can be drawn from observing jump magnitudes is that magnitudes of up jumps lost it's significance while down jump magnitudes increased a little, also both jump intensities dropped. To sum up, it seems that the jumps generally lost significance in favor of volatility and overally, OMX30 came out damaged from the earthquake.

Moving on to Ericsson B stock, it seems quite interesting that the σ parameter remained quite indifferent to the earthquake event. The μ parameter seems to have increased a little after the earthquake which could be another indicator that Ericsson B remained "unharmed". Jumps, however became affected. While the magnitudes of both types of jumps have increased a little, the intensities have changed differently. Namely, down jumps intensity seems to have risen more than the up jumps intensity. This might mean that Ericsson actually got affected a little by the bad news incoming from Japan and the damage done to this country's industry. Overall, Ericsson B seems to have withstood the earthquake without meaningful damage although it did suffer a little from the bad news.

Finally, the results for SEB A shall be considered. In this case, σ seems to be untouched or even experienced a slight fall. The drift parameter rose a little. These two factors could indicate the Nordic banking sector's immunity to the earthquake in Japan. Both η_d and η_u parameters have fallen slightly (indicating a slight raise in jump magnitudes) while λ_u has risen and λ_d remained on roughly the same level on average. This indicates the good news seemed to have higher impact on SEB A after the crisis. Overall, SEB A was not affected much by the earthquake and only jumps gained some influence "stealing" it from the volatility. This in turn could indicate that the movements of SEB A price became larger after the catastrophical event.

To sum up, Tokio's Nikkei225 index was obviously most severely damaged from the earthquake of 11-03-2011. The catastrophe did have its effect on the Nordic market as well. Though not so severe, OMX30 also suffered some damage, more than SEB A and Ericsson B which could indicate that other companies from other sectors took a much bigger hit.

Chapter 5

Conclusions

In this thesis I presented various models and methods of parameter estimation. The Pareto-Beta Jump-Diffusion (PBJD) model of Ramezani and Zeng (1998) has been chosen to be estimated with the Maximum Likelihood Estimation method using high frequency intra-day data. A fact worth noting is that the PBJD model is equivalent to the Double Exponential Jump-Diffusion (DEJD) model by Kou (2002). The link between these two models has been presented and therefore provided with a method to compare results of PBJD with those of DEJD. The method of application of MLE to the PBJD model has been presented and executed.

In order to apply the MLE to the PBJD model I have used intra-day data from the Tokio's Nikkei225 index, Stockholm's OMX30 index and also SEB A and Ericsson B - two companies from Sweden. A procedure of the application in form of a program written in C has been presented along with results and their interpretation. The difficulties and challenges encountered during programming have been described along with ways to solve these problems. Also, I have provided a few useful hints for future researchers who might want to use a similar method so that they can avoid some time-consuming dead ends.

Numerous program runs (320, to be exact) have been carried out to obtain the estimations of the parameters. The results, however have proven to be difficult to interpret which could point to either not enough program runs made or to some imperfections in the program. Indeed, the results show that the maximization procedure is sensitive to the choice of initial parameter set and suggest that many more than 10 initial sets per data set should be tested. Unfortunately, due to lack of time and computer power more tests could not have been carried out.

From this comes a suggestion for improvement for further research - more estimations have to be made and also the program I have written might

require some tweaks and corrections, especially in the integration parts. After these adjustments, the program should be perfectly capable of providing estimations for other research using PBJD model or DEJD, using the link between the models presented in Chapter 3. Another suggestion would be to find other methods of integration, i.e. more accurate and efficient, to implement to the program or try another optimization algorithm.

As a final thought, I would like to discuss the usefulness of PBJD. As it was shown, this method requires large computational effort which takes a lot of time. In the financial world, results have to be provided quickly because, as someone said, "Time is money", which unfortunately hinders PBJD's utility. Does that make PBJD a bad model? Not necessarily, since there are situations in which there is time to make a well thought decision and this is where PBJD can indeed shine.

Notation

$a \wedge b$	Minimum of a and b
λ_d	Intensity of down-jumps
λ_u	Intensity of up-jumps
η_d	Magnitude of down-jumps
η_u	Magnitude of up-jumps
μ	Drift component of returns
σ	Volatility of returns

Bibliography

[1] Y. Aït-Sahalia (2004)
Disentangling diffusion from jumps. Journal of Financial Economics, **74**, 487 – 528.

[2] Y. Aït-Sahalia, L. P. Hansen (2004)
Handbook of Financial Econometrics. Amsterdam: North-Holland.

[3] R. Cont, P. Tankov (2003)
Financial modelling with jump processes. London: Chapman & Hall / CRC, Financial Mathematic Series.

[4] C. Gourieroux, A. Monfort, E. Renault (1993)
Indirect Inference. Journal of Applied Econometrics, **8**, Supplement: Special Issue on Econometric Inference Using Simulation Techniques, S85 – S118.

[5] J. D. Hamilton (1994)
Time Series Analysis. Princeton: Princeton University Press.

[6] L. P. Hansen (1982)
Large sample properties of generalized method of moments estimators. Econometrica, **50**, 1029 – 1054.

[7] P. Honoré (1998)
Pitfalls in estimating jump diffusion models. Working Paper, University of Aarhus.
`http://www.stat.purdue.edu/~figueroa/Teaching/Stat598F/HonoreJmpDffsEst.pdf`

[8] J. C. Hull (2009)
Options, Futures and Other Derivatives. Prentice Hall India; 7th edition.

[9] S. G. Kou (2002)
A jump diffusion model for option pricing. Management Science, **48**, 1086 – 1101.

[10] R. C. Merton (1976)
Option pricing when underlying stock returns are discontinuous. Journal of Financial Economics, **3**, 125 – 144.

[11] A. K. Mitra
Evaluation of a Multiple Integral by Gauss Quadrature
. Department of Aerospace Engineering, Iowa State University.
`http://www.public.iastate.edu/~akmitra/aero361/design_web/gauss_quad.pdf`

[12] J. P. Moreau
A library of programs in C and C++.
`http://jean-pierre.moreau.pagesperso-orange.fr/c_function.html`

[13] M. F. M. Osborne (1959)
Brownian motion in the stock market. Operation Research, **7**, 145 – 153.

[14] M. J. D. Powell (2009)
The BOBYQA algorithm for bound constrained optimization without derivatives. Technical report DAMTP 2009/NA06, Centre for Mathematical Sciences, University of Cambridge, Cambridge, UK.
`http://www.damtp.cam.ac.uk/user/na/NA_papers/NA2009_06.pdf`

[15] W. H. Press, S. A. Teukolsky, W. T. Vetterling, B. P. Flannery (1992)
Numerical Recipes in C. Cambridge University Press.

[16] P. Protter (1991)
Stochastic Integration and Differential Equations. Berlin Heidelberg New York: Springer.

[17] C. A. Ramezani, Y. Zeng (1998)
Maximum likelihood estimation of asymmetric jump-diffusion processes: Application to security prices. Working Paper, Department of Mathematics and Statistics, Univeristy of Missouri, Kansas City.
`http://alturl.com/tysz9`

[18] C. A. Ramezani, Y. Zeng (2007)
Maximum likelihood estimation of the double exponential jump-diffusion process. Annals of Finance (2007), **3**, 487 – 507.

[19] M. Sorensen (1991)
Likelihood Methods for Diffusion with Jumps. In N.U. Prabhu and I.V. Basawa (eds.): *Statistical inference in Stochastic Processes*, Marcel Dekker, New York, 1991, 67 – 105.

[20] The Economist Online (2011)
Counting the cost
http://www.economist.com/blogs/dailychart/2011/03/natural_disasters?fsrc=scn/fb/wl/dc/countingthecost

[21] Wikipedia article on Pareto distribution
http://en.wikipedia.org/wiki/Pareto_distribution

[22] Wikipedia article on Generalized Method of Moments
http://en.wikipedia.org/wiki/Generalized_method_of_moments

[23] The dlib library
http://dlib.net/

Appendix

APPENDIX A: *Results of the computations of MLE*

Table 5.1 to Table 5.8 present the results of MLE calibrations for all 8 analyzed data sets. The first column indicates the minute intervals between data points, second lists the parameters (see notation) and V - value of the likelihood function for these parameters. The next 10 columns contain the results, each cell is a resulting parameter set and likelihood function value for a given initial parameter set in each case of time intervals. Numbers in bold indicate the set with highest likelihood function value.
A blank page has been inserted so that "after" and "before" results for each company or index are on adjacent pages for convenience.

Int	Par	1	2	3	4	5	6	7	8	9	10
	λ_d	.52517	.39095	.02729	1.08173	**.27342**	1.03804	1.02107	.70850	2.03774	1.17563
	λ_u	.55869	.35187	1.62755	1.32725	**.94596**	.02419	.89862	1.27222	1.50255	2.10671
	η_d	83.822	49.783	102.968	207.680	**126.068**	121.013	67.031	61.264	73.466	201.022
5	η_u	84.237	50.064	129.027	204.315	**136.159**	247.371	60.952	64.131	51.622	196.261
	μ	-.0000009	.0619457	-.0000470	-.0000259	**-.0000040**	.0000236	-.0000007	-.0000174	.0000037	-.0000517
	σ	.00018	.03500	.00060	.00065	**.00014**	.00059	.00012	.00058	.00062	.00061
	V	6603.81	-238499	5395.21	5187.17	**6718.88**	5434.02	5812.87	4711.37	3467.89	5095.88
	λ_d	.76689	**.01030**	.27837	.97731	.01228	.59766	1.11398	.89899	1.83818	1.36033
	λ_u	.64749	**.03372**	2.55268	1.51240	1.59025	.46435	1.04558	1.22196	1.56069	1.87370
	η_d	79.170	**22.394**	59.060	198.835	102.887	107.509	68.601	43.339	54.609	203.515
10	η_u	76.920	**136.026**	92.621	199.523	79.122	147.007	65.468	70.596	72.254	197.970
	μ	.0000010	**-.0000009**	-.0001584	-.0000614	-.0000156	-.0000009	.0000007	-.0000152	.0000017	-.0000781
	σ	.00020	**.00026**	.00081	.00090	.00018	.00018	.00086	.00086	.00016	.00090
	V	3046.12	**3708.19**	2235.11	2606.45	2830.77	3289.75	2279.35	2291.45	2119.17	2493.87
	λ_d	.81171	.79534	.12013	1.35964	.45968	**1.06074**	1.38968	1.13033	2.26643	1.77459
	λ_u	.65624	1.01478	2.98295	1.41287	1.19155	**.12473**	.96881	1.30309	1.39951	1.81423
	η_d	80.444	6.699	99.773	201.864	160.665	**67.191**	71.663	63.483	70.400	195.882
20	η_u	77.070	24.638	154.431	225.557	98.041	**174.426**	68.283	65.112	62.686	195.094
	μ	.0000005	.0000092	-.0004244	-.0000651	-.0001041	**.0000251**	.0000016	-.0000043	.0000509	-.0001398
	σ	.00029	.00034	.00112	.00133	.00119	**.00036**	.00028	.00027	.00123	.00125
	V	1451.60	1249.59	1267.22	1268.28	1314.84	**1495.00**	1240.26	1228.22	875.06	1194.77
	λ_d	.82116	.71301	**.25777**	.93818	.33344	.97507	1.39050	1.26140	2.30710	1.61010
	λ_u	.65333	.83583	**.02430**	1.76267	.71669	.45937	1.07426	1.13030	1.51380	1.89638
	η_d	82.112	1.630	**74.718**	199.729	241.018	90.108	74.912	59.282	62.052	198.979
30	η_u	75.706	67.299	**185.037**	198.717	26.004	150.821	68.946	74.098	59.492	195.834
	μ	.0000017	.0398450	**.0000206**	-.0002683	-.0000028	.0000154	.0000078	-.0000009	.0000774	-.0002459
	σ	.00047	.00077	**.00067**	.00156	.00060	.00047	.00033	.00033	.00153	.00155
	V	905.60	-494.78	**1063.92**	855.24	913.29	929.11	769.45	782.10	548.59	793.79

Table 5.1: The results of the parameter estimation for Ericsson B stock before the earthquake. Note very close values of V in column 1 and 6 of 20-minute interval data.

Int	Par	1	2	3	4	5	6	7	8	9	10
5	λ_d	.56608	.39095	.06908	1.17815	.29999	**.81388**	1.03126	.68011	1.74312	1.20075
	λ_u	.62267	.35187	1.83746	1.29501	1.17549	**.06228**	.64298	1.24995	1.53228	2.12553
	η_d	85.184	49.782	130.061	202.611	143.217	**151.083**	73.911	66.217	76.754	193.262
	η_u	79.750	50.064	37.442	202.603	76.380	**177.676**	66.193	75.196	61.252	198.610
	μ	.0000008	.0619457	-.0000261	-.0000080	-.0000255	**.0000008**	.0000009	-.0000008	-.0000004	-.0000201
	σ	.00014	.03500	.00059	.00020	.00059	**.00018**	.00030	.00008	.00062	.00042
	V	6612.98	-238494	4254.43	5725.85	5194.32	**6773.60**	5502.48	5961.66	3802.48	5015.83
10	λ_d	.82402	**.05657**	.07411	.83279	.08008	.31753	1.19869	1.12052	1.93696	1.48155
	λ_u	.58813	**.07039**	1.99050	1.62872	1.47314	.47625	1.12953	1.31851	1.48161	2.06992
	η_d	76.615	**4.943**	121.858	199.686	92.125	92.444	61.351	70.261	61.528	200.660
	η_u	84.245	**76.722**	269.102	199.731	79.657	158.040	71.743	59.661	61.519	207.316
	μ	.0000019	**-.0000002**	-.0000995	-.0000849	-.0000722	-.0000029	.0000005	-.0000030	.0000120	-.0000850
	σ	.00015	**.00022**	.00094	.00087	.00084	.00017	.00015	.00086	.00087	.00089
	V	3069.87	**3645.38**	2828.18	2638.61	2538.78	3431.83	2611.12	2164.33	1823.60	2471.81
20	λ_d	**.84423**	.62330	.06531	1.30296	.39830	1.36934	1.39707	1.50484	2.22465	1.65333
	λ_u	**.68744**	1.16281	2.78094	1.38047	1.13213	.05591	.95847	1.23822	1.43412	1.90478
	η_d	**79.363**	6.216	55.457	201.995	171.106	97.409	75.261	58.056	62.125	196.744
	η_u	**77.860**	61.282	149.518	216.066	71.409	169.394	69.331	82.922	77.431	196.370
	μ	**.0000032**	-.0000019	-.0003931	-.0000923	-.0000172	.0000392	.0000061	-.0000006	.0000454	-.0001432
	σ	**.00026**	.00030	.00115	.00128	.00050	.00035	.00036	.00024	.00126	.00123
	V	**1450.37**	1310.66	1270.95	1272.91	1407.79	1418.82	1240.86	1187.83	907.81	1215.71
30	λ_d	.81300	.14432	**.13202**	1.09341	.01046	.76426	1.50585	1.35249	2.28575	1.69782
	λ_u	.65388	1.13282	**.07772**	1.71452	1.01089	.46049	1.11396	1.20885	1.54004	1.89431
	η_d	80.563	24.296	**77.964**	202.623	203.185	102.370	89.547	62.093	61.093	196.959
	η_u	75.682	41.117	**133.639**	215.429	51.177	143.212	77.352	70.075	58.725	197.907
	μ	.0000126	-.0000243	**.0000110**	-.0002167	-.0000435	.0000144	.0000077	.0000294	.0000958	-.0000826
	σ	.00043	.00051	**.00063**	.00157	.00059	.00042	.00037	.00150	.00154	.00036
	V	917.12	881.61	**1084.96**	846.96	916.17	970.14	764.65	705.43	547.78	819.70

Table 5.2: The results of the parameter estimation for Ericsson B stock after the earthquake. Note a few similar values non-bold V's to that of the bold one in 5-, 10- and 20- minute cases.

Int	Par	1	2	3	4	5	6	7	8	9	10
	λ_d	.28363	.35102	.48697	1.71607	**.01467**	1.00379	.77130	.44438	1.94028	1.13329
	λ_u	.20108	.31189	2.34258	1.79983	**.92898**	.24161	.92750	1.26936	1.58333	1.92420
	η_d	96.505	39.779	92.257	186.927	**122.288**	86.237	66.107	71.993	52.292	209.978
5	η_u	89.356	40.068	93.811	203.289	**115.479**	172.763	66.744	75.779	60.156	208.948
	μ	.0000024	.0619493	-.0000032	-.0000424	**-.0000020**	.0000012	-.0000034	-.0000267	.0000046	-.0000471
	σ	.00017	.03501	.00009	.00064	**.00007**	.00007	.00014	.00061	.00064	.00064
	V	4911.54	-185525	4889.30	4336.90	**7017.28**	6705.66	5711.20	4533.05	3233.92	4680.43
	λ_d	.81977	**.50962**	.02802	.79257	.01449	.96619	1.35774	.82367	1.77908	1.50910
	λ_u	.66647	**.55902**	2.56786	1.52394	1.78667	.01202	.89658	1.13911	1.36139	1.91855
	η_d	64.102	**48.376**	1.561	197.825	138.686	91.589	71.999	72.161	75.403	207.770
10	η_u	72.344	**61.628**	170.362	197.948	30.832	158.620	77.900	83.710	65.620	204.522
	μ	-.000101	**-.0000019**	-.0001867	-.0000041	-.0000044	.0000007	.0000013	-.0000171	.00000605	-.0000939
	σ	.00072	**.00009**	.00085	.00010	.00013	.00010	.00010	.00091	.00092	.00092
	V	3046.18	**3251.36**	2362.82	2878.99	2643.71	2916.60	2668.93	2191.98	1760.61	2233.39
	λ_d	.91002	**.33187**	.03074	1.40937	.01059	2.00137	1.37042	1.19987	2.30213	1.98171
	λ_u	.67850	**.30367**	1.37495	1.62806	.86632	.07367	1.07406	1.19668	1.51421	1.95009
	η_d	89.350	**5.138**	156.547	200.640	41.419	106.828	59.859	51.835	52.540	196.525
20	η_u	77.750	**52.804**	39.329	177.557	45.334	22.869	58.824	64.010	50.661	195.011
	μ	.0000003	**.0000007**	-.0000167	-.0000068	-.0000114	.0000154	.0000016	-.0000027	.0000028	-.0000072
	σ	.00015	**.00021**	.00031	.00015	.00026	.00035	.00015	.00128	.00015	.00016
	V	1439.49	**1628.52**	1337.50	1254.41	1513.75	1178.27	1217.62	981.49	881.77	1144.70
	λ_d	.85667	.60720	.02982	1.09201	.05722	**1.02318**	1.53684	1.12464	2.40484	1.72217
	λ_u	.57621	.98082	1.83514	1.52713	1.42016	**.08482**	1.01898	1.22736	1.47549	1.98924
	η_d	73.233	13.118	168.917	185.707	152.932	**56.380**	78.670	54.021	59.860	197.318
30	η_u	71.682	23.574	63.379	201.423	63.595	**161.040**	61.700	64.977	59.414	200.890
	μ	.0000001	-.0000004	-.0002809	-.0002396	-.0002055	**.0000106**	.0000029	.0000009	.0000849	-.0002699
	σ	.00020	.00025	.00143	.00156	.00144	**.00022**	.00019	.00018	.00157	.00154
	V	992.24	915.08	722.82	801.45	777.61	**1023.61**	811.30	847.39	510.36	747.91

Table 5.3: The results of the parameter estimation for Nikkei225 index before the earthquake. No noteworthy similiarities of V.

Int	Par	1	2	3	4	5	6	7	8	9	10
5	λ_d	**.28363**	.35102	.54763	1.67580	.01545	1.32632	.81556	1.18390	1.97965	.94075
	λ_u	**.20108**	.31189	2.56125	1.64184	.85831	.71598	1.11672	1.10045	1.60438	1.92872
	η_d	**96.505**	39.779	85.200	191.619	108.834	75.445	81.530	73.156	64.974	209.234
	η_u	**89.356**	40.068	98.325	194.722	115.817	172.394	62.673	70.736	63.229	196.928
	μ	**.0000024**	.0619493	-.0000843	-.0000346	-.0000325	.0000055	.0000002	.0000014	.0000021	-.0000483
	σ	**.00017**	.03501	.00060	.00065	.00061	.00068	.00015	.00014	.00066	.00065
	V	**6465.93**	-185525	4033.23	4287.37	5147.56	4362.27	5144.18	4862.37	3240.78	4682.95
10	λ_d	.93385	**.01520**	.08255	.71297	.01079	.98539	1.05840	1.08881	2.04909	1.47428
	λ_u	.71728	**.84799**	1.27460	1.56248	1.70434	.87134	.95930	1.09947	1.35943	1.84755
	η_d	74.116	**23.787**	4.543	199.735	128.070	124.000	68.770	61.249	60.007	213.147
	η_u	68.534	**60.449**	38.761	199.069	32.390	163.693	69.141	85.392	62.976	205.448
	μ	.0000117	**-.0000074**	-.0000454	-.0000839	-.0000091	-.0000188	-.0000006	-.0000009	.0000289	-.0000848
	σ	.00086	**.00020**	.00086	.00092	.00021	.00094	.00020	.00012	.00092	.00092
	V	2663.92	**2828.91**	2138.48	2380.55	2247.92	2294.62	2468.65	2470.48	1623.43	2204.13
20	λ_d	.85836	**.39539**	.04728	1.23220	.25015	1.48441	1.49160	1.38522	2.19890	1.96079
	λ_u	.68553	**.37630**	1.37708	1.82560	.56837	.01175	1.09794	1.11631	1.49402	1.94943
	η_d	83.060	**9.176**	143.713	191.175	53.145	125.209	75.089	57.437	65.555	195.549
	η_u	84.129	**54.717**	36.153	196.123	55.882	43.511	68.375	79.617	55.454	198.024
	μ	.0000101	**.0000037**	-.0000178	-.0001604	-.0000047	.0001888	.0000038	.0000070	.0000556	-.0001874
	σ	.00032	**.00036**	.00041	.00132	.00033	.00126	.00034	.00028	.00131	.00130
	V	1266.17	**1389.47**	1151.23	1103.64	1387.14	1092.70	1050.15	1082.00	748.85	1015.79
30	λ_d	**.97667**	.38815	.01445	1.13134	.22106	1.10942	1.54066	1.34140	2.44523	1.61586
	λ_u	**.53170**	1.08450	2.48252	1.49895	1.53129	.77797	1.13119	1.21156	1.42993	1.97908
	η_d	**61.862**	9.390	174.572	188.122	151.480	90.848	72.518	56.844	58.287	202.245
	η_u	**74.418**	28.898	50.998	178.158	54.417	121.196	72.344	72.553	63.204	196.816
	μ	**-.0000043**	-.0000187	-.0004820	-.0001951	-.0002069	-.0000031	.0000003	-.0000060	.0001884	-.0002478
	σ	**.00040**	.00058	.00153	.00157	.00144	.00040	.00040	.00153	.00150	.00157
	V	**839.02**	789.15	566.13	758.51	701.36	820.92	699.22	653.21	500.78	727.36

Table 5.4: The results of the parameter estimation for Nikkei225 index after the earthquake. One very close non-bold V to the bold one (20-minute interval, column 5).

Int	Par	1	2	3	4	5	6	7	8	9	10
	λ_d	.23629	.45102	.05274	1.03838	**.02622**	.51899	1.17503	.69730	1.60083	1.10054
	λ_u	.64331	.41189	.57193	1.41425	**.90092**	.06232	.89334	1.23197	1.54599	2.20729
	η_d	83.984	59.779	158.126	208.024	**163.578**	110.115	68.264	70.667	63.004	219.245
5	η_u	76.710	60.068	65.134	203.440	**134.981**	206.471	60.360	72.215	62.013	217.329
	μ	.0000012	.0999877	-.0000037	-.0000015	**-.0000031**	.0000023	.0000010	-.0000188	.0000001	-.0000151
	σ	.00009	.05403	.00018	.00005	**.00007**	.00019	.00006	.00059	.00005	.00034
	V	7537.18	-260720	7255.21	6392.41	**7596.58**	7288.36	6160.32	4837.58	4995.54	5248.98
	λ_d	.77383	**.54852**	.27790	.89646	.01583	.95159	1.18605	.88018	2.01893	1.36877
	λ_u	.46564	**.25633**	2.64698	1.48452	1.76127	.31992	1.00548	1.21744	1.40015	1.91254
	η_d	81.118	**81.094**	60.109	198.995	89.453	89.352	71.993	45.261	59.457	201.874
10	η_u	97.974	**13.298**	182.698	194.117	107.643	168.013	62.173	68.476	65.460	194.006
	μ	.0000016	**.0000034**	-.0001662	-.0000613	-.0000982	.0000015	.0000013	-.0000257	.0000126	-.0000815
	σ	.00012	**.00017**	.00081	.00089	.00082	.00015	.00011	.00085	.00088	.00087
	V	3425.56	**3565.24**	2580.74	2639.31	2594.45	3394.91	2859.44	2303.47	1833.14	2498.10
	λ_d	.85332	**.07599**	.01028	1.37239	.39146	1.35322	1.28990	1.52258	2.30592	1.95270
	λ_u	.60317	**.08196**	2.51876	1.39037	1.27324	.01969	.96065	1.12717	1.49424	2.01025
	η_d	86.432	**34.767**	225.037	206.859	173.874	116.954	79.393	58.502	58.382	201.303
20	η_u	79.972	**68.947**	55.347	189.315	91.167	221.645	69.124	72.550	62.220	198.807
	μ	.0000019	**.0000019**	-.0002335	-.0001192	-.0000136	.0000175	.0000052	.0000177	.0000041	-.0001665
	σ	.00020	**.00027**	.00111	.00124	.00025	.00027	.00025	.00123	.00020	.00121
	V	1591.76	**1921.41**	1002.50	1260.69	1543.63	1563.74	1369.42	1075.15	978.80	1187.89
	λ_d	.90657	**.02949**	.40326	1.09311	.05485	.82387	1.49880	1.27863	2.31615	1.87264
	λ_u	.71501	**.01000**	.10589	1.83266	.87144	.65614	1.02224	1.23642	1.51712	1.97714
	η_d	85.275	**42.789**	119.123	196.697	251.043	93.981	86.007	61.406	58.693	203.982
30	η_u	82.968	**23.349**	139.619	194.871	30.822	170.717	69.128	66.730	69.031	209.805
	μ	.0000119	**.0000077**	.0000152	-.0002568	-.0000080	.0000049	.0000409	.0000064	.0000174	-.0000562
	σ	.00029	**.00045**	.00035	.00153	.00038	.00029	.00152	.00029	.00026	.00032
	V	975.71	**1198.59**	1146.64	848.76	1018.44	1029.42	715.11	824.21	627.98	838.28

Table 5.5: The results of the parameter estimation for OMX30 index before the earthquake. 5-minute, non-bold V in column 1 very close to the bold one.

Int	Par	1	2	3	4	5	6	7	8	9	10
5	λ_d	**.24064**	.45102	.01166	1.10508	.01033	1.98622	1.44532	.60925	1.56991	1.18903
	λ_u	**.65336**	.41189	2.50175	1.19894	1.02266	.11279	.79805	1.29902	1.37017	2.08696
	η_d	**85.143**	59.779	144.826	205.756	127.138	165.542	73.076	86.186	69.321	210.737
	η_u	**80.074**	60.068	42.829	219.082	96.559	198.741	56.229	78.164	62.682	210.511
	μ	**.0000001**	.0999877	-.0000010	-.0000005	-.0000271	.0000020	.0000139	.0000020	-.0000004	-.0000429
	σ	**.00014**	.05403	.00015	.00007	.00059	.00005	.00060	.00013	.00005	.00062
	V	**7233.42**	-260718	4728.78	6501.17	5610.68	6026.49	4440.36	6060.72	5236.67	5155.91
10	λ_d	.47564	.57312	.01031	.84264	.15996	**.40451**	1.21755	.83371	2.04911	1.54060
	λ_u	.56040	.71308	1.43138	1.37123	1.60989	**.57677**	1.01388	1.30250	1.40424	2.04434
	η_d	101.704	37.770	22.275	193.151	88.268	**100.625**	64.245	26.994	54.747	194.727
	η_u	93.001	50.091	100.518	191.242	77.893	**154.636**	71.953	90.752	57.002	193.791
	μ	-.0000005	.0000019	-.0000744	-.0000042	-.0000733	**-.0000025**	.0000040	.0000016	.0000179	-.0000055
	σ	.00013	.00012	.00084	.00013	.00082	**.00011**	.00087	.00015	.00087	.00011
	V	3531.45	3324.01	2664.40	3125.49	2478.80	**3608.12**	2274.59	2853.94	1773.02	2682.61
20	λ_d	.84090	**.01176**	.01798	.01798	.31850	1.36736	1.36893	1.20288	2.30448	1.71587
	λ_u	.60914	**.01004**	1.34667	1.34667	1.28095	.05342	.98217	1.25940	1.39228	1.84583
	η_d	85.950	**3.786**	2.147	2.147	170.872	94.493	71.188	56.099	47.394	199.847
	η_u	78.573	**59.995**	231.378	231.378	99.076	158.347	68.423	75.438	60.034	190.592
	μ	.0000040	**.0000027**	-.0001575	-.0001575	-.0000090	.0000177	.0000302	-.0000010	.0000076	-.0001478
	σ	.00022	**.00032**	.00126	.00126	.00020	.00025	.00121	.00019	.00018	.00123
	V	1584.84	**1911.23**	1468.41	1468.41	1565.96	1535.15	1125.73	1328.16	978.86	1211.08
30	λ_d	.96169	.16402	**.02558**	1.07536	.12200	.52856	1.50791	1.37114	2.27150	1.83288
	λ_u	.72808	.03188	**.07563**	1.44711	.89342	.49429	.96820	1.20536	1.55435	2.07949
	η_d	84.765	32.675	**123.710**	201.354	218.024	97.697	74.923	62.705	59.688	186.059
	η_u	91.358	45.440	**153.949**	212.155	43.901	162.281	66.330	64.187	61.244	192.345
	μ	.0000118	.0000171	**.0000079**	-.0000235	-.0000208	.0000012	.0000222	.0000144	.0000192	-.0002534
	σ	.00028	.00052	**.00041**	.00032	.00049	.00034	.00028	.00028	.00028	.00152
	V	964.29	1168.41	**1190.54**	950.43	1010.93	1086.63	821.61	808.34	618.20	777.86

Table 5.6: The results of the parameter estimation for OMX30 index after the earthquake. Noteworthy similarities of V's in 10- and 30- minute interval rows.

Int	Par	1	2	3	4	5	6	7	8	9	10
	λ_d	.28274	.45102	.01000	1.13912	.08153	**.57571**	.86649	.61909	1.86265	1.26172
	λ_u	.61734	.41189	1.29498	1.22621	.77033	**.03081**	.91792	1.27626	1.39262	2.24829
	η_d	67.848	59.779	130.197	208.167	135.947	**99.248**	63.717	68.020	53.646	210.395
5	η_u	68.985	60.068	94.778	204.730	110.489	**219.480**	74.138	69.900	54.210	196.389
	μ	-.0000023	.0999877	-.0000366	-.0000056	-.0000233	**.0000018**	-.0000005	-.0000224	.0000125	-.0000537
	σ	.00031	.05403	.00058	.00015	.00060	**.00013**	.00016	.00059	.00060	.00061
	V	6245.97	-260721	5389.58	5906.30	5759.75	**7178.75**	5849.62	4827.30	3661.12	5057.49
	λ_d	.63770	**.06662**	.01009	.91060	.01003	.71953	.96479	.90189	1.88551	1.35048
	λ_u	.53886	**.42022**	2.85619	1.59102	1.30814	.46482	1.00967	1.19356	1.49044	1.84502
	η_d	76.435	**21.778**	103.985	198.871	93.829	117.341	69.436	40.042	57.092	207.830
10	η_u	64.337	**57.564**	261.437	203.581	115.290	143.173	81.076	61.445	54.927	196.310
	μ	.0000009	**-.0000026**	-.0001547	-.0000653	-.0000754	-.0000014	-.0000062	-.0000164	.0000169	-.0000773
	σ	.00017	**.00023**	.00089	.00091	.00085	.00021	.00086	.00086	.00088	.00089
	V	3149.01	**3459.14**	2769.30	2625.15	2732.63	3214.16	2392.38	2260.89	1800.68	2495.36
	λ_d	.83342	**.13740**	.01000	1.29858	.35376	1.12142	1.31265	1.16357	2.21104	1.69411
	λ_u	.68126	**.01533**	2.93873	1.41216	1.24156	.14312	.96283	1.30645	1.45102	1.90836
	η_d	77.251	**30.022**	78.954	200.392	168.185	48.566	74.458	44.694	59.353	198.478
20	η_u	79.795	**71.473**	168.403	226.169	96.287	167.813	72.466	95.756	73.556	199.978
	μ	.0000043	**.0000043**	-.0004005	-.0000819	-.0000294	.0000240	.0000194	-.0000141	.0000312	-.0001646
	σ	.00028	**.00041**	.00117	.00135	.00030	.00034	.00121	.00124	.00124	.00126
	V	1455.64	**1750.81**	1301.99	1278.22	1451.81	1485.04	1139.95	1131.92	900.44	1213.55
	λ_d	.81239	**.07151**	.96097	1.02862	.20377	.73353	1.41970	1.28076	2.38507	1.84213
	λ_u	.77120	**.26487**	.06094	1.47258	.99590	.46128	1.05082	1.20654	1.50840	1.86737
	η_d	76.518	**58.695**	106.925	200.729	261.652	88.495	64.420	60.997	59.921	201.329
30	η_u	88.499	**54.059**	86.036	202.280	28.806	132.435	77.483	65.340	64.561	199.198
	μ	.0000020	**-.0000132**	.0000463	-.0001932	-.0000192	.0000081	.0000214	-.0000180	.0000798	-.0002823
	σ	.00039	**.00058**	.00050	.00157	.00051	.00039	.00149	.00152	.00152	.00149
	V	920.47	**1065.81**	968.24	851.46	896.97	982.68	720.94	709.66	547.95	772.26

Table 5.7: The results of the parameter estimation for SEB A stock before the earthquake. No noteworthy similarities of V.

Int	Par	1	2	3	4	5	6	7	8	9	10
5	λ_d	**.31192**	.45102	.15051	1.13342	.11799	1.72368	.89869	.64314	2.00892	1.14185
	λ_u	**.70542**	.41189	2.25575	1.35541	1.21584	.04036	.93588	1.24086	1.49326	2.04772
	η_d	**65.920**	59.779	101.920	214.120	156.895	150.100	61.579	66.223	63.909	194.644
	η_u	**80.931**	60.068	113.603	209.596	154.572	198.662	74.016	71.682	56.694	200.912
	μ	**-.0000004**	.0999877	-.0000650	-.0000042	-.0000339	.0000409	-.0000047	.0000006	.0000084	-.0000604
	σ	**.00015**	.05403	.00058	.00017	.00061	.00060	.00061	.00012	.00063	.00058
	V	**6744.55**	-260719	4927.74	5819.47	5641.64	4844.05	4853.35	5848.40	3550.87	5086.39
10	λ_d	.64316	**.31623**	.78714	.88981	.09230	.64540	.91675	.89139	2.01554	1.54322
	λ_u	.67666	**.60755**	2.99859	1.52687	1.83137	.78479	.99265	1.17398	1.47082	2.02742
	η_d	82.520	**53.910**	102.641	197.987	112.976	120.692	69.363	40.530	56.051	201.291
	η_u	73.361	**56.924**	221.776	204.495	113.841	142.730	88.897	77.296	58.582	197.274
	μ	-.0000005	**-.0000020**	-.0001425	-.0000732	-.0001033	-.0000050	-.0000027	-.0000212	.0000136	-.0000848
	σ	.00018	**.00018**	.00086	.00090	.00085	.00088	.00016	.00082	.00088	.00088
	V	3105.15	**3269.68**	2532.90	2634.89	2567.63	2691.78	2851.74	2317.47	1774.05	2443.72
20	λ_d	.90064	**.02372**	.32084	1.31023	.31914	1.28062	1.45021	1.23248	2.15528	1.69168
	λ_u	.64134	**.01175**	2.10397	1.37702	1.20801	.05741	.96067	1.16926	1.50142	1.92128
	η_d	89.010	**39.658**	88.181	198.200	175.010	117.064	68.976	62.408	59.004	194.842
	η_u	78.018	**74.109**	75.829	209.024	82.225	189.860	71.870	75.775	70.894	191.099
	μ	.0000208	**.0000067**	-.0002182	-.0000836	-.0000164	.0000341	.0000340	.0000020	.0000317	-.0001608
	σ	.00121	**.00042**	.00115	.00131	.00032	.00041	.00122	.00029	.00124	.00124
	V	1282.58	**1788.83**	1142.37	1269.73	1460.73	1461.83	1111.12	1266.62	902.15	1208.70
30	λ_d	.88435	**.34007**	.52991	1.22296	.01011	.76505	1.46105	1.29054	2.28975	1.86028
	λ_u	.72521	**.10895**	.01133	1.53113	1.22464	.61305	1.04765	1.17865	1.52362	1.90928
	η_d	72.527	**45.149**	183.946	203.080	169.469	103.000	65.478	61.111	69.096	200.568
	η_u	83.189	**40.290**	198.818	202.207	69.397	156.082	78.765	68.094	60.266	193.931
	μ	.0000077	**.0000270**	.0000522	-.0001857	-.0001818	-.0000016	.0000206	.0000159	.0000159	-.0002617
	σ	.00042	**.00056**	.00061	.00154	.00144	.00044	.00034	.00033	.00034	.00149
	V	910.59	**1054.20**	1037.77	831.29	851.33	971.51	782.65	782.40	591.02	768.64

Table 5.8: The results of the parameter estimation for SEB A stock after the earthquake. Close value of non-bold V to bold V in 30-minute row, column 3.

APPENDIX B: *Code for PBJD optimization by MLE*

```
//Program for calibration of the PBJD model using MLE. All calculations
 of proper functions are carried out in this program, while the
 optimization by BOBYQA method is conducted by a routine provided
 by the dlib library.
#include<stdio.h>
#include<math.h>
#include<conio.h>
#include"dlib/optimization.h"
#include<iostream>
#include<time.h>
#include<stdlib.h>

using namespace std;
using namespace dlib;

// Here we just make a typedef for a variable length column
 vector of doubles.
typedef matrix<double,0,1> column_vector;

//Constants
const double pi = 3.14159265358979323846;
const double FTOL = 10E-10;

//test parameters
double lambdad = 0.21*100.0;
double lambdau = 0.42*100.0;
double etad = 60.2;
double etau = 72.9;
double mu = -0.0028*1000.0;
double sigma = 0.068*2000.0;
int m = 2;
int n = 4;
double r = -0.003;

//Global variables
double periods; //s in the formula, read from the data file
double data[4000];
double returns[4000];
```

```
double init_point[6];
int data_amount;
char input_name[40];
char output_name[40];
FILE* output;
int counter=0;


int read_data_file(char name[40]){
//routine to read the returns into an array
//it returns the amount of data points
int i,n;
FILE* input;
input = fopen(name,"r");
if(input == NULL) return -1;
//Read number of data points
fscanf(input,"%d\n",&n);
//Read number of periods (value of s - most likely going to stay 1)
fscanf(input,"%lf\n",&periods);
//Read the initial point (rescaled)
for(i=0;i<6;i++){
fscanf(input,"%lf\n",&init_point[i]);
}
//Read the data points
for(i=0;i<n;i++){
fscanf(input,"%lf\n",&data[i]);
}
fclose(input);

//return number of data points
return n;
}

void calculate_returns(int n){
int i;
for(i=0;i<n-1;i++){
returns[i] = log(data[i+1]) - log(data[i]);
}
}

//Compare function for sorting
```

```
int compare (const void * a, const void * b)
{
if(*(double*)a > *(double*)b) return 1;
else if(*(double*)a == *(double*)b) return 0;
else if(*(double*)a < *(double*)b) return -1;
}

//Minimum of 2 numbers - name changed to avoid conflict with dlib
double smaller(double a, double b){
if(a < b) return a;
else return b;
}

//Factorial function
double fact(int n){
double k=1.0;
if (n == 0) return 1.0;
for (int j = 1; j < n; j++){
k = k * (j+1);
}
return (double)k;
}

//Poisson coefficient
double poi(int n, double par){
double factorial;
factorial = fact(n);
return (exp(-par)*pow(par,n))/(factorial);
}

double f00(double s, double r, double mu, double sigma){
double numerator,a;
a = r-mu*s+0.5*sigma*sigma*s;
numerator = a*a;
return (1/(sqrt(2*pi*s)*sigma))*exp(-(numerator)/(2*sigma*sigma*s));
}

//Functions to be integrated
double f0nint(double s, double r, int n, double etad, double mu,
 double sigma, double x){
double exponent,numerator,a,coeff;
```

```
a = r-x-mu*s+0.5*sigma*sigma*s;
numerator = a*a;
exponent = etad * x - numerator/(2*sigma*sigma*s);
coeff = pow(etad,n) / (fact(n-1) * sqrt(2.0*pi*s) * sigma);

return pow(-x,n-1) * coeff * exp(exponent);
}

double fm0int(double s, double r, int m, double etau, double mu,
 double sigma, double x){
double exponent,numerator,a,coeff;
a = r-x-mu*s+0.5*sigma*sigma*s;
numerator = a*a;
exponent = -etau * x - numerator/(2*sigma*sigma*s);
coeff = pow(etau,m) / (fact(m-1) * sqrt(2.0*pi*s) * sigma);

return pow(x,m-1) * coeff * exp(exponent);
}

double fmnint(double s, double r, int m, int n, double etad, double etau,
 double mu, double sigma, double t, double x){
double numerator, denominator, exponent, numerexp, a;

numerator = pow(etau,m) * pow(etad,n) * pow(-x,n-1) * pow(t-x,m-1);
denominator = fact(m-1) * fact(n-1) * sqrt(2*pi*s) * sigma;
a = r - t - mu*s + 0.5 * sigma * sigma * s;
numerexp = a*a;
exponent = (etau+etad)*x - etau*t - numerexp / (2*sigma*sigma*s);

return numerator * exp(exponent) / denominator;
}

//Integrals
double f0nGauss(double s, double r, int n, double etad,
 double mu, double sigma){
int j;
double xr,xm,dx,sum;
double x[] = {0.0,0.095012509837637440185319 3,0.281603550779258913230460 50
,0.458016777657227386342419 4,0.617876244402643748446671 8,
0.755404408355003033895101 2,0.865631202387831743880467 9,
0.944575023073232576077988 4,0.989400934991649932596154 2};
```

```
double w[] = {0.0,0.18945061045506849628539 67,0.18260341504492358886676 37,
0.16915651939500253818931 21,0.14959598881657673208150 17,
0.12462897125553387205247 63,0.09515851168249278480992 51,
0.06225352393864789286284 38,0.02715245941175409485178 06};


//detection where it makes sense to integrate - it's always just one 'hill'
int points = 8;
double step=0.05;
double a,b,y=0.0;
bool GoFlaga = true, GoFlagb = true;
double tiny = 1E-12;
double temp1,temp2;

while(GoFlaga || GoFlagb){
if(GoFlagb){
if(f0nint(s,r,n,etad,mu,sigma,y)>tiny){
if(y<-step/2.0) b=y+step;
else b=0.0;
y=y-step;
GoFlagb = false;
}
}
//If the b was found
if(!GoFlagb){
if(f0nint(s,r,n,etad,mu,sigma,y)<tiny){
a=y;
GoFlaga = false;
}
}
y = y - step;
//If the b was found but y reached -2.5, a=y
if(y < -2.5 && !GoFlagb){
GoFlaga = false;
GoFlagb = false;
a=y;
}
//If b hasn't been found yet and y reached -2.5, return 0
if(y < -2.5 && GoFlagb){
GoFlaga = false;
GoFlagb = false;
```

```
return 0.0;
}
}
//end of detection

xm = 0.5 * (b+a);
xr = 0.5 * (b-a);
sum = 0.0;
for(j=1;j<=points;j++){
dx = xr * x[j];
temp1 = f0nint(s,r,n,etau,mu,sigma,xm+dx);
if(temp1 < 0.0) temp1 = 0.0;
temp2 = f0nint(s,r,n,etau,mu,sigma,xm-dx);
if(temp2 < 0.0) temp2 = 0.0;
sum = sum + w[j] * (temp1 + temp2);

}
//printf("f0n integrate test = %lf for n=%d\n",sum*xr,n);
return sum*xr;
}

double fm0Gauss(double s, double r, int m, double etau,
 double mu, double sigma){
int j;
double xr,xm,dx,sum;

double x[] = {0.0,0.095012509837637440185319335,0.281603550779258913230460501,
0.458016777657227386342419442,0.617876244402643748446671764,
0.755404408355003033895101194,0.865631202387831743880467897,
0.944575023073232576077988415,0.989400934991649932596154173};
double w[] = {0.0,0.189450610455068496285396705,0.182603415044923588866763667,
0.169156519395002538189312079,0.149595988816576732081501790,
0.124628971255533872052476308,0.095158511682492784809925107,
0.062253523938647892862843836,0.027152459411754094851780572};

//detection where it makes sense to integrate - it's always just one 'hill'
int points = 8;
double step=0.05;
double a,b,y=0.0;
bool GoFlaga = true, GoFlagb = true;
double tiny = 1E-12;
```

```
double temp1,temp2;

while(GoFlaga || GoFlagb){
if(GoFlaga){
if(fm0int(s,r,m,etau,mu,sigma,y)>tiny){
if(y>step/2.0) a=y-step;
else a=0.0;
y=y+step;
GoFlaga = false;
}
}
//If the a was found
if(!GoFlaga){
if(fm0int(s,r,m,etau,mu,sigma,y)<tiny){
b=y;
GoFlagb = false;
}
}
y = y + step;
//If a was found and 2.5 was reached, make b = 2.5
if(y > 2.5 && !GoFlaga){
GoFlaga = false;
GoFlagb = false;
b=y;
}
//If a was not found and 2.5 was reached, return 0
if(y > 2.5 && GoFlaga){
GoFlaga = false;
GoFlagb = false;
return 0.0;
}
}
//end of detection

xm = 0.5 * (b+a); //middle of the integration interval
xr = 0.5 * (b-a); //radius of the interval
sum = 0.0;
for(j=1;j<=points;j++){
dx = xr * x[j];
temp1 = fm0int(s,r,m,etau,mu,sigma,xm+dx);
if(temp1 < 0.0) temp1 = 0.0;
```

```
temp2 = fm0int(s,r,m,etau,mu,sigma,xm-dx);
if(temp2 < 0.0) temp2 = 0.0;
sum = sum + w[j] * (temp1 + temp2);
}

return sum*xr;
}

double fmnGauss(double s, double r, int m, int n, double etad,
 double etau, double mu, double sigma){
//Different number of abscissas for t and x: x requires more precision
double xt[] = {0.0,0.076526521133497333754640 4,
0.22778585114164507808049 62,0.37370608871541956067254 82,
0.51086700195082709800436 41,
0.63605368072651502545283 67,0.74633190646015079261430 51,
0.83911697182221882339452 91,0.91223442825132590586775 24,
0.96397192727791379126766 61,0.99312859918509492478612 24};
double wt[] = {0.0,0.15275338713072585069808 43,
0.14917298647260374678782 87,0.14209610931838205132929 83,
0.13168863844917662689849 45,
0.11819453196151841731237 74,0.10193011981724043503675 01,
0.08327674157670474872475 81,0.06267204833410906356950 65,
0.04060142980038694133104 00,0.01761400713915211831186 20};
double xx[] = {0.0,0.04830766568773831623481 26,
0.14447196158279649348518 64,0.23928736225213707454460 32,
0.33186860228212764977991 68,
0.42135127613063534536411 94,0.50689990893222939002374 75,
0.58771575724076232904074 55,0.66304426693021520097511 52,
0.73218211874028968038742 67,0.79448379596794240696309 73,
0.84936761373256997013369 30,0.89632115576605212396530 72,
0.93490607593773968917091 91,0.96476225558750643077381 19,
0.98561151154526833540017 50,0.99726386184948156354498 11};
double wx[] = {0.0,0.09654008851472780056676 48,0.09563872007927485941908 20,
0.09384439908080456563918 02,0.09117387869576388471286 86,
0.08765209300440381114277 15,0.08331192422694675522219 91,
0.07819389578707030647174 09,0.07234579410884850622539 94,
0.06582222277636184683765 01,0.05868409347853554714528 36,
0.05099805926237617619616 32,0.04283589802222668065687 86,
0.03427386291302143310268 77,0.02539206530926205945575 26,
0.01627439473090567060517 06,0.00701861000947009660040 71};
```

```
//number of points so I don't have to rewrite in every loop
int points_t = 10;
int points_x = 16;

double sumI, sumJ, sum, t;
double step = 0.2;
double a,b,c,d,dx,xr,xm,dt,tr,tm;
int i,j,k;
double temp1,temp2;

sum = 0;
//Integrate piece by piece for t [-2,2]
for(k=0;k<=(4-step)/step;k++){
sumI = 0.0;
//for step by step movement at
a = -2.0 + step * k;
b = a + step;
//a = -0.4;
//b = -0.2;
tm = 0.5*(b+a);
tr = 0.5*(b-a);

for(i=1;i<=points_t;i++){
dt = tr * xt[i];

//one side of tm
t = tm + dt;
c = -2.5;
d = smaller(0,t);
//printf("d=%lf\n",d);
xm = 0.5*(d+c);
xr = 0.5*(d-c);
sumJ = 0.0;
for(j=1;j<=points_x;j++){
dx = xr * xx[j];
//Sometimes the function goes out of bounds and gives
 negative values - that is
//unacceptable and so, such values are zeroed, ergo - ignored
temp1 = fmnint(s,r,m,n,etad,etau,mu,sigma,t,xm+dx);
if(temp1 < 0.0) temp1 = 0.0;
temp2 = fmnint(s,r,m,n,etad,etau,mu,sigma,t,xm-dx);
```

```
if(temp2 < 0.0) temp2 = 0.0;
sumJ = sumJ + wx[j] * (temp1 + temp2);
}
sumI = sumI + wt[i]*sumJ*xr;

//second side of tm
t = tm - dt;
c = -2.0;
d = smaller(0,t);
//printf("d=%lf\n",d);
xm = 0.5*(d+c);
xr = 0.5*(d-c);
sumJ = 0.0;
for(j=1;j<=points_x;j++){
dx = xr * xx[j];
temp1 = fmnint(s,r,m,n,etad,etau,mu,sigma,t,xm+dx);
if(temp1 < 0.0) temp1 = 0.0;
temp2 = fmnint(s,r,m,n,etad,etau,mu,sigma,t,xm-dx);
if(temp2 < 0.0) temp2 = 0.0;
sumJ = sumJ + wx[j] * (temp1 + temp2);
}
sumI = sumI + wt[i]*sumJ*xr;

}
sumI = sumI*tr; //tr * sumI
//printf("%lf %lf %lf\n",sumI,a,b);
sum = sum + sumI;
}
//printf("fmn integrate test = %lf and m=%d, n=%d\n",sum,m,n);
return sum;

}

//Integrals multiplied by poisson coefiicient
double f0n(double s, double r, int n, double lambdad,
 double etad, double mu, double sigma){
double integral;
integral = f0nGauss(s,r,n,etad,mu,sigma);

return poi(n,lambdad) * integral;
}
```

```
double fm0(double s, double r, int m, double lambdau,
 double etau, double mu, double sigma){
double integral;
integral = fm0Gauss(s,r,m,etau,mu,sigma);

return poi(m,lambdau) * integral;
}

double fmn(double s, double r, int m, int n, double lambdad,
 double lambdau, double etad, double etau, double mu, double sigma){
double integral;
integral = fmnGauss(s,r,m,n,etad,etau,mu,sigma);

return poi(n,lambdad)*poi(m,lambdau)*integral;
}

//Infinite summations with termination criterion
double NSumf0n(double s, double r, double tolerance, double lambdad,
 double etad, double mu, double sigma){
double result=0, next;
int n=1;
bool GoFlag = true;

while(GoFlag){
result = result + f0n(s,r,n,lambdad,etad,mu,sigma);
next = f0n(s,r,n+1,lambdad,etad,mu,sigma);
if(2.0*fabs(next) <= tolerance * (fabs(result)+fabs(result+next))){
GoFlag = false;
result = result + next;
}
n++;
}

return result;
}

double NSumfm0(double s, double r, double tolerance, double lambdau,
 double etau, double mu, double sigma){
double result=0, next;
int m=1;
```

```
bool GoFlag = true;

while(GoFlag){
result = result + fm0(s,r,m,lambdau,etau,mu,sigma);
next = fm0(s,r,m+1,lambdau,etau,mu,sigma);
if(2.0*fabs(next) <= tolerance * (fabs(result)+fabs(result+next))){
GoFlag = false;
result = result + next;
}
m++;
}

return result;
}

double NSumfmn(double s, double r, double tolerance, double lambdad,
 double lambdau, double etad, double etau, double mu, double sigma){
double result=0.0,next;
int m=0,n=0;
bool GoFlag=true;

while(GoFlag){
m++;
for(n=1;n<=m;n++){
next = fmn(s,r,m,n,lambdad,lambdau,etad,etau,mu,sigma);
if(2.0*fabs(next) <= tolerance * (fabs(result) + fabs(result+next))){
result = result + next;
GoFlag = false;
return result;
}
else result = result + next;
}
//condition to avoid a neverending loop
if(m > 11){
GoFlag = false;
return result;
}
}
}

//Final calculation of f(r)
```

```
double f(double s, double r, double lambdad, double lambdau, double etad,
 double etau, double mu, double sigma){
double f00_part, f0n_part, fm0_part, fmn_part;

f00_part = exp(-(lambdad+lambdau)) * f00(s,r,mu,sigma);
f0n_part = exp(-lambdau) * NSumf0n(s,r,FTOL,lambdad,etad,mu,sigma);
fm0_part = exp(-lambdad) * NSumfm0(s,r,FTOL,lambdau,etau,mu,sigma);
fmn_part = NSumfmn(s,r,FTOL,lambdad,lambdau,etad,etau,mu,sigma);
return f00_part + f0n_part + fm0_part + fmn_part;
}

//Calculation of the likelihood function
double likelihood(double lambdad, double lambdau, double etad, double etau,
 double mu, double sigma){
int i;
double sum=0.0;
double temp,prev;
long counter=0;

prev = log(f(periods,returns[0],lambdad/100.0,lambdau/100.0,etad,etau,
mu/1000.0,sigma/2000.0));
sum += prev;

for(i=1;i<data_amount-1;i++){
if(returns[i] == returns[i-1]){
sum += prev;
counter++;
}
else{
temp = log(f(periods,returns[i],lambdad/100.0,lambdau/100.0,etad,etau,
mu/1000.0,sigma/2000.0));
prev = temp;
sum += temp;
}
}
printf("Calculations saved: %ld    ",counter);
return sum;
}

//object used to communicate with find_max_bobyqa
class likelihood_call{
```

```
public:

//empty constructor
likelihood_call(){}

double operator() (const column_vector& arg) const{
double result;
//increment the global variable counter
counter++;

result = likelihood(arg(0),arg(1),arg(2),arg(3),arg(4),arg(5));
cout << "counter: " << counter << endl;
cout << "lambdad = " << arg(0) << endl;
cout << "lambdau = " << arg(1) << endl;
cout << "   etad = " << arg(2) << endl;
cout << "   etau = " << arg(3) << endl;
cout << "     mu = " << arg(4) << endl;
cout << "  sigma = " << arg(5) << endl;
cout << "Value   = " << result << endl;

fprintf(output,"\nRemember about rescaling! Results:\n");
fprintf(output,"Counter: %d\n",counter);
for(int j=0;j<6;j++){
fprintf(output,"%lf\n",arg(j));
}
fprintf(output,"Value: %lf\n",result);

return result;
}
};

//main function
int main(){
    try{
        //make column vectors of length 6 to work with find_max_bobyqa
column_vector starting_point;
column_vector lower_bounds, upper_bounds;

starting_point.set_size(6);
lower_bounds.set_size(6);
```

```
upper_bounds.set_size(6);

printf("Data file name: ");
scanf("%s",&input_name);
printf("Output file name: ");
scanf("%s",&output_name);
output = fopen(output_name,"w");

time_t time_begin,timer_begin;
time_t time_end,timer_end;
struct tm * timeinfo;

time_begin = time(&timer_begin);
timeinfo = localtime(&timer_begin);
fprintf(output,"Start time and date: %s\n",asctime(timeinfo));

data_amount = read_data_file(input_name);

if(data_amount == -1){
printf("The data file could not be found or read or something.\n");
getch();
return 0;
}

calculate_returns(data_amount);
//sorting the returns array
qsort(returns, data_amount, sizeof(double), compare);

starting_point = init_point[0],
init_point[1],
init_point[2],
init_point[3],
init_point[4],
init_point[5];

cout << "init_point: " << starting_point << endl;
//The variables have been rescaled to have the same distance
//between lower and upper bounds
lower_bounds = 0.1, 0.1, 0.1, 0.1, -250, 0.05;
upper_bounds = 500, 500, 500, 500, 250, 400;
```

```
double minimum_value = find_max_bobyqa(
likelihood_call(),
                        starting_point,
                        13,    // number of interpolation points
                         - best is 2*dim + 1 (13 in our case)
                        lower_bounds,  // lower bound constraint
                        upper_bounds,   // upper bound constraint
                        50,    // initial trust region radius
                         (0.1 of the interval length)
                        0.01,  // stopping trust region radius
                         (desired precision)
                        500    // max number of
                         objective function evaluations
        );

time_end = time(&timer_end);
timeinfo = localtime(&timer_end);
fprintf(output,"\n\nFinal time and date: %s\n",asctime(timeinfo));
fprintf(output,"Time elapsed: %ld seconds\n",time_end-time_begin);
printf("\n\nFinal time and date: %s\n",asctime(timeinfo));
printf("Time elapsed: %ld seconds\n",time_end-time_begin);
cout << "Value: " << minimum_value << " in: " <<
 endl << starting_point << endl;

//Printing results to a file
fprintf(output,"Remember about rescaling! Results:\n");
fprintf(output,"Periods: %lf\n",periods);
for(int j=0;j<6;j++) fprintf(output,"%lf\n",starting_point(j));
fprintf(output,"Value: %lf",minimum_value);
fclose(output);
printf("\n\n\nTHE PROGRAM HAS FINISHED! YOU CAN TURN IT OFF NOW :)");
printf("\noutput file: %s",output_name);
getch();
    }
    catch (exception& e){
        cout << e.what() << endl;
    }
}
```