CHALMERS | GÖTEBORG UNIVERSITY

MASTER THESIS

Numerical Methods for Infinitely Divisible Distributions

ANTON CRONA

Department of Mathematical Statistics
CHALMERS UNIVERSITY OF TECHNOLOGY
GÖTEBORG UNIVERSITY
Göteborg, Sweden 2010

Thesis for the Degree of Master of Science

Numerical Methods for Infinitely Divisible Distributions

Anton Crona

CHALMERS | GÖTEBORG UNIVERSITY

Department of Mathematical Statistics
Chalmers University of Technology and Göteborg University
SE – 412 96 Göteborg, Sweden
Göteborg, January 2010

Abstract

Finding accurate statistical estimates for observed phenomenon, which underlying mechanisms are difficult to model mathematically, is a frequently reoccurring problem confronting many contemporary engineers from various branches of science. In this thesis we investigate how a numerical method can be implemented to find parameters of the very general Lévy process. The parameters were estimated by minimizing the Kolmogorov-Smirnov distance between the estimated distribution and the empiric distribution of the data by use of a simulated annealing global optimization algorithm. The results obtained from the parameter estimation were used in the Lévy process model which in turn was compared with empiric data from the Dow Jones Industrial Average of the New York Stock Exchange. The results shows a clear improvement over the maximum likelihood fitted Normal Distribution, especially for extreme events.

Acknowledgements

I would like to thank my advisor Patrik Albin not only for his inspiration, help and advice, but also for giving me the opportunity to study this subject. The time I've spent writing this thesis, as well as studding Patriks courses has been the most fruitful period of my time at Chalmers.

I would also like to thank Magnus Önnheim for giving me valuable insights into the subject of heuristic optimization, insights that have contributed significantly to the outcome of this thesis.

Contents

1 Introduction

One of the biggest tasks of stock portfolio managers today is that of finding accurate estimates of the risks involved with a particular investment. Accurately being able to quantify the risk of an investment is necessary in many aspects of portfolio management. For example, estimates are a guard against erroneous hedging strategies. I.e. with accurate risk measures the risk of over/under insuring your assets are reduced.

It is also important for eventual clients investing in a portfolio to be able to make correct judgement about the risks that they are about to take. Put simply, accurate risk estimation is one of the key factors in maintaining a sound portfolio strategy. But how is risk measured?

One way to estimate risk of investment in a particular financial asset, or a portfolio of assets, is to gather historical data of the assets price movements. Usually in the form of stock market closing prices for the asset or portfolio. These data are then subjected to various forms of statistical analysis which serves to estimate the risk of different events occurring. In modern finance, quantitative methods, in the form of statistical analysis, is a inevitable tool for conducting such risk analysis.[1]

In this paper we will implement a method based on the so called Lévy stochastic process. For each portfolio we look at, we will be trying to find the Lévy process that best describes the behaviour of that portfolio. Once that is done, we can extrapolate information in the form of probabilities for certain events. And we will most likely be interested in the extremely negative events. Put in other words, we would like know what the risk is of our investment losing some portion of its value over some period of time. The purpose of this thesis will not only be to describe how this particular Lévy process can be found, but also at what computational cost this estimate can be found.

2 Theoretical foundation

Before describing the implementation of the parameter estimation program we will present in this thesis, we must first develop the underlying theoretical foundations of its mechanisms.

The theoretical chapter of this thesis will present the basic concepts describing which defines the characteristics and properties of the Lévy process. We will also give a brief introduction to the discrete Fourier transform which we will implement by the computationally efficient fast Fourier transform. Further, we will give a short introduction to the subject of computational complexity analysis and finally, a description of the simulated annealing optimization algorithm which we'll use to locate the optimum set of parameters.

2.1 Definition of the Lévy process

The Lévy process is a continuous time stochastic process, named after French mathematician Paul Levy. We will define it in the following way:

$X = \{X_t : t \geq 0\}$ is a Lévy-process given that the following properties of the process holds:

1. The process starts out at the origin. I.e. $X_0 = 0$ holds almost surely.
2. The process is stationary for all increments. $\forall s < t$, $X_t - X_s$ has the same distribution as X_{t-s}.
3. The process is almost surely Cádlág for $t > 0$, i.e. it is is right continous with left limits.
4. All increments are independent identically distributed. More formally, for an arbitrary set of times $0 < t_1 < t_2 < \ldots < t_n < \infty$ the set of $X_{t_2} - X_{t_1}, X_{t_3} - X_{t_2}, \ldots, X_{t_n} - X_{t_{n-1}}$ are all independent.

From the definition, which is a very general one, it follows that the family of Lévy processes is very big including both the Wiener process, where increments are normally distributed, and the Poisson process in which the increments are Poisson distributed.

2.2 Characteristic functions in probability theory

In probability theory and statistics, the characteristic function, $\varphi(\omega)$, of a random variable X with probability distribution function F is defined as the Fourier-Stieltjes transform of that distribution, which is to say

$$\varphi_X(\theta) = \mathbb{E}[e^{i\theta X}] = \int_{-\infty}^{\infty} e^{i\theta x} F(dx).$$

The characteristic function exists for all distributions and it gives a complete definition of the distribution, just as the distribution function, but from a different perspective of frequency content. When investigating the behaviour of a distribution, through its characteristic function, we'll look at its behaviour in the frequency domain instead of in the probability domain where the distribution function is defined.

The set of available analytical tools for investigating a probability distribution through its characteristic function is different from that of the density function. The model, later developed in this thesis, is constructed using a representation of characteristic functions – the Lévy-Khintchine representation.

2.3 Lévy-Khintchine representation

Since it is possible to characterize all distributions by their characteristic functions this is also true for the distribution of Lévy Processes. The Lévy-Khintchine representation [2] states that if the process X_t is a Lévy process, then its characteristic function $\varphi_{X_t}(\theta)$ can be expressed in the following way:

$$\varphi_{X_t}(\theta) = \mathbb{E}[e^{i\theta X_t}] = \exp\left\{ t\left[ia\theta - \frac{\sigma^2\theta^2}{2} + \int_{\mathbb{R}\setminus\{0\}} (e^{i\theta x} - 1 - i\theta x \mathbf{1}_{\{|x|<1\}}) W(dx) \right] \right\}.$$

Here $a \in \mathbb{R}$ and $\sigma \geq 0$ are constants, while W is the so called Lévy measure on $\mathbb{R} \setminus \{0\}$ that satisfies

$$\int_{\mathbb{R}\setminus\{0\}} \min\{1, x^2\} W(dx) < \infty.$$

Let us decompose the above characteristic function $\varphi_{X_t}(\theta)$ into two factors

$$\varphi_{X_t}(\theta) = \exp\left\{ t\left[i\left(a - \int_{\mathbb{R}\setminus\{0\}} x\mathbf{1}_{\{|x|<1\}} W(dx) \right)\theta - \frac{\sigma^2\theta^2}{2} \right] \right\} \exp\left\{ t\int_{\mathbb{R}\setminus\{0\}} (e^{i\theta x} - 1) W(dx) \right\}.$$

Here the first factor is the characteristic function of a normal $N(\mu t, \sigma^2 t)$ distribution. Now, if we approximate the Lévy measure W by an appropriate discrete point measure

$$V(dx) = \sum_{k=1}^{n} a_k \delta(x - b_k) dx,$$

which can always be done arbitrarily well if $n \in \mathbb{N}$, $a_1, \ldots, a_n \geq 0$ and $b_1, \ldots, b_n \in \mathbb{R}$ are suitably chosen, then the second factor above takes the form

$$\begin{aligned} \exp\left\{ t\int_{\mathbb{R}\setminus\{0\}} (e^{i\theta x} - 1) W(dx) \right\} &\approx \exp\left\{ t\int_{\mathbb{R}\setminus\{0\}} (e^{i\theta x} - 1) V(dx) \right\} \\ &= \exp\left\{ t\sum_{k=1}^{n} (e^{i\theta b_k} - 1) a_k \right\} \\ &= \mathbb{E}\left\{ \exp\left[i\theta \sum_{k=1}^{n} b_k \mathrm{Po}_k(a_k t) \right] \right\}. \end{aligned}$$

Here $\{\mathrm{Po}_k(a_k t)\}_{k=1}^n$ are independent Poisson distributed random variables with parameters $\{a_k t\}_{k=1}^n$.

For the above result it holds that the larger the number of terms in the sum defining the discrete approximating Lévy measure V, the better the approximation of the true Lévy measure

W (and the corresponding true distribution).

We are now ready to infer two important conclusions.

1. Firstly, the distribution of any stochastic process, fulfilling the requirements to be a Lévy process, can be approximated by inverse Fourier transforming the product of the characteristic functions of a normal distributed random variable and a sum of of rescaled Poisson distributed variables.

2. Secondly, by increasing the number of scaled Poisson variables, we will be able to boost the accuracy of our estimation and thus reduce the residual error of our estimate. Since the fundamental assumption of this thesis is that the stock price processes are actually realizations of some unknown Lévy processes, this result implicates that approximations with larger sets of scaled Poisson variables will yield more accurate estimates of the stock price behaviour.

2.4 Daily returns

Since our objective is to use quantitative methods to model stock price movements, we need some mathematical measure to quantify its rather irregular behaviour. We'll will consider the stock price $S = \{S(t), 0 \leq t \leq T\}$ over some period of time $[0, T]$. In the Bachilier-Samuelsson stock price process model, a stock price is modelled as a bank account with an additional noise term. More specifically

$$S(t) = S(0)e^{rt+\lambda W(t)},$$

where $r \in \mathbb{R}$ is a constant and W is a Wiener type of Lévy process with $W(t-s) \sim N(0, t-s)$. Now we want to model the stock price behaviour with the exponent of a more general Lévy process than the Wiener process. As a Lévy process allows for a "built in" drift term rt, we may express our more general stock price model as

$$S(t) = S(0)e^{X(t)},$$

where $X(t)$ is some Lévy process, yet unknown.
Our main focus of study will deal with the movement of the stock prices rather than the actual stock prices themselves, and therefore we will define the daily return of an arbitrary stock S over some arbitrary time period p as

$$R(t) = \frac{S(t) - S(t-p)}{S(t-p)}$$

2.5 De-volatilizing daily returns by σ-balancing

Since the volatility of the stock market is changing with time we will use the method of $\sigma - balancing$ [3] to rescale the returns to returns of an equal volatility. This is done in the following way. We group the returns $R(t)$ into bins. Each of size 20. I.e. we stratify the returns into groups, each of size 20. By calculating the mean volatility as

$$\sigma_{mean} = \frac{20}{T} \sum_{i=1}^{N} \sigma_i$$

where σ_i is the volatility of the i:th bin and T the total number of returns. Now we balance the volatility by multiplying each return $R(t)$ of bin i with $\frac{\sigma_{mean}}{\sigma_i}$. This will cancel out all eventual differences in volatility and will in turn simplify the parameter fitting since our proposed model is based on a constant volatility rate over the whole period which we work with.

2.6 Quantifying goodness of fit

Since the idea behind this thesis is that stock price movements are driven by some unknown Lévy process, our main focus of attention will be of estimating the parameters of this unknown process by iteratively comparing the empiric data we have from daily stock returns with candidate Lévy processes. To do this, we need some measure of how "good" a particular candidate actually is. As quality measure of some arbitrary candidate, we'll use the Kolmogorov-Smirnov (KS) distance which is defined as:

$$D_{ks} = \sup_x |F_c(x) - F_n(x)|$$

Where $F_c(x)$ is the cumulative distribution function (cdf) of our candidate Lévy process & $F_n(x)$ is the empirical distribution function (edf), of our observed daily returns, defined by:

$$F_n(x) = \frac{1}{n} \sum_{t=1}^{n} I(R(t) \leq x)$$

Where I(X) is the indicator function of event X and n the total number of observations.

2.7 Optimization theory

Together with the previously presented theory, we are now almost ready to make a complete problem formulation for our Lévy parameter fitting. But we will first introduce some basic theory on optimization.

2.7.1 Local & global optimality

Faced with the problem to

$$\begin{gathered}\text{minimize } f(\mathbf{x}) \\ \text{subject to } \mathbf{x} \in S\end{gathered}$$

Where S is some arbitrary set of dependent variables, we define $\mathbf{x}^*$ as the global minimum of f over S if $\mathbf{x}^*$ yields the lowest value of f over S.

Facing the same problem, we say that ${\mathbf{x}_l}^*$ is a local minimum of f over the set S if

$$\exists \epsilon > 0 \text{ such that } {\mathbf{x}_l}^* \leq \mathbf{x}, \mathbf{x} \in S \bigcap B_\epsilon(\mathbf{x}^*)$$

Where $B_\epsilon({\mathbf{x}_l}^*)$ is the set of spheres with radius ϵ centred at ${\mathbf{x}_l}^*$. That is, ${\mathbf{x}_l}^*$ is the global minimum of f in some arbitrary small proximity of ${\mathbf{x}_l}^*$.[4]

2.8 Computational traceability

There exists a wide spread misconception that recent performance increases of modern computers are of such magnitude that almost any computational problem can be solved efficiently. This is of course not true. Execution time does matter.

Since we will develop a numeric approximation algorithm for parameter fitting, we would like to define some sort of measure of performance for a proposed algorithm. We would like this measure to be able to tell us weather a proposed algorithm has a good running time or not.

2.8.1 Asymptotic upper bound of growth

Let $T(n)$ denote the worst case running time for some particular algorithm, i.e. $T(n)$ is the largest number of fundamental comparisons a computer must evaluate in order to return the result of the particular algorithm.

Given a computational complexity function $C(n)$ we define the asymptotic upper bound of $T(C(n))$ to be $O(C(n))$ if there exists some constants $n_0 > 0$ and $K > 0$ such that $KC(n) \geq T(n), \forall n \leq n_0$.
(In spoken language we say: "$T(n)$ is of order $C(n)$")

2.8.2 Definition of computational efficiency

In this thesis we will use the definition of computational efficiency proposed by Kleinberg & Tardos[5] which says:

"A computational algorithm is considered efficient if it runs in polynomial time"

There are of course cases where polynomial time algorithms have long running times, either due to large constant factors or big exponents. However, in most cases, this definition is a reasonable.

2.9 Discrete Fourier transform

The problem of analytically estimating the KS distance from a set of characteristic functions requires the use of the Fourier transform. In this context however, since we use a computer to generate candidates for the optimal solution, the discrete Fourier transform (DFT) will be a more suitable choice of method for transforming between characteristic functions and density functions. We will therefore give a brief introduction of the method.

Given a sequence of N complex of complex numbers $x = \{x_0, x_1, \ldots, x_{N-1}\}$. The DFT is defined as the process of converting the elements of x into the sequence of transformed complex numbers $T = \{t_1, t_2, \ldots, t_{N-1}\}$ using the following formula: [6]

$$T_k = \sum_{n=0}^{N-1} x_n e^{-\frac{2\pi i}{N} kn}$$

Where i is defined as the complex unit. Since we, among other things, are interested in finding out how computationally difficult the DFT is to calculate, we'll rewrite the N-point DFT as a $N \times N$ matrix multiplication problem $\mathbf{T} = \mathbf{W}\mathbf{x}$ where $\mathbf{x}$ is the N-point original input and $\mathbf{W}$ the transformation matrix.

This representation might seem annoyingly complicated at a first glance. And that might very well be true! But, this representation will significantly simplify our quest of establishing a asymptotic upper bound of execution time once we get to that. For now, lets focus on the matrix problem!

The transformation matrix $\mathbf{W}$ is defined as follows:

$$\mathbf{W} = \frac{1}{\sqrt{N}} \begin{bmatrix} 1 & 1 & 1 & \ldots & 1 \\ 1 & \omega & \omega^2 & \ldots & \omega^{N-1} \\ 1 & \omega^2 & \omega^4 & \ldots & \omega^{2(N-1)} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & \omega^{N-1} & \omega^{2(N-1)} & \ldots & \omega^{(N-1)(N-1)} \end{bmatrix}$$

Where ω is defined as the primitive N:th root of unity $\omega = e^{-\frac{2\pi i}{N}}$

2.10 Fast Fourier transform

If we recall from basic linear algebra, solving the matrix-problem $\mathbf{T} = \mathbf{W}\mathbf{x}$ for $\mathbf{T}$ where $\mathbf{W}$ is of size $N \times N$ will require N^2 operations. And this is just as true for the DFT-matrix problem. This means that a algorithm calculating the DFT of a n-length sequence, by solving the DFT-matrix problem, will be $O(n^2)$.

The $O(n^2)$ asymptotic upper bound certainly fulfils our definition of efficient algorithms, so maybe we should be satisfied at this point and just let the computer solve this problem? This is of course a highly rhetorical question and this is also the point of where the fast Fourier transform (FFT) comes into play.
Because if we would choose the input sequence in a clever way, we may be able to further reduce the asymptotic upper bound.
Now lets choose a input sequence $\mathbf{x}$ of length n where $n = 2^N$, under this assumption, its possible to factorize $\mathbf{W}$ in the following way: [6]

$$\mathbf{W_{2^N}} = \begin{bmatrix} \mathbf{I_{2^{N-1}}} & \mathbf{D_{2^{N-1}}} \\ \mathbf{I_{2^{N-1}}} & -\mathbf{D_{2^{N-1}}} \end{bmatrix} \begin{bmatrix} \mathbf{W_{2^{N-1}}} & 0 \\ 0 & \mathbf{W_{2^{N-1}}} \end{bmatrix} \mathbf{P}$$

Where $\mathbf{I}$ is a identity matrix, $\mathbf{P}$ is the permutation matrix that rearranges the input vector $\mathbf{x}$ so that element of even order are placed at the top, and conversely, elements with an uneven order are stacked from the bottom. The diagonal matrix $\mathbf{D}$ is defined as

$$\mathbf{D_{2^{N-1}}} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & \omega_{2^N} & 0 & 0 & 0 \\ 0 & 0 & 0(\omega_{2^N})^2 & 0 & 0 \\ 0 & 0 & 0 & \ddots & 0 \\ 0 & 0 & 0 & 0 & (\omega_{2^N})^{2^{N-1}-1} \end{bmatrix}$$

We can now again factorize the matrix $\mathbf{W_{2^N}}$ according to the analogy above. The FFT will repeat this process until the matrix $\mathbf{W}$, (which was of size $N \times N$) is factored into N simpler matrices. Applying this method (the FFT), will yield a asymptotic upper bound of $O(nlog(n))$. This is a major reduction of the number of necessary computational steps. Especially in the application we are about to use it for where the size of n will be very large, and the precision of our algorithm is dependent upon how many transforms we will be able to calculate. In our program we will also be concerned with calculating the inverse of the Fourier transform. This is done by solving the inverse of the FFT matrix problem.

2.11 Simulated annealing

Simulated annealing (SA), is a probabilistic meta heuristic algorithm for global optimization. Its a beneficial method to use for problems, where solutions are located within a large non-linear,non-convex search space. Properties that can all be shown to hold for our problem. The

name annealing is an analogy that originates from metallurgy.

In metallurgy, annealing is a technique used to enlarge the size, and reduce the defects, of crystals in a material by first heating the material and then slowly cooling it. The heating of the material increases the internal energy of the material, causing the molecules of the material to leave their current state (by analogy a local minimum) and move around randomly through states of higher energy (by analogy non local minimum). By slowly cooling the material, the molecule movements will be through states of lower internal energy, and eventually converging to a state of lower internal energy than that of the initial state.

2.11.1 The iteration

At each step of the iteration, the SA algorithm considers some randomly choosen neighbour of the current state. In our implementation, we will always redefine the current state to be that of the neighbour if the neighbour yields a better solution than the current state. However, if the neighbour yeild a worse solution to the problem, the algorithm might still choose to move. This decision is based on the acceptance probability.

2.11.2 Acceptance probability

The acceptance probability $P(e, e', T)$ is defined as the probability of moving from a current state to the neighbouring state. It is a function of the energy of the current state e, the energy of the neighbouring state e' and the temperature T of the system.

In most applications, the probability of changing state will be larger if $e - e'$ and T are large. The most important property however is that $P(e, e', T) = 1$ when $e - e' > 0$ implicating that we will always move to a lower state of energy if we find one.

2.11.3 Cooling schedule

The coolings schedule of the SA algorithm defines how we change the temperature T as the algorithm progresses. In the typical application T will start with a large value, making the algorithm move around among alot of potential solution candidates of various quality. However, as the number of iterations approch its predefined maximum, T will approach 0, and the closer the algorithm gets to the end of its run, the acceptance probability of accepting neighbouring states of higher energy will become smaller, thus forcing the algorithm to move towards some local (and hopefully global) optimum.

2.11.4 Annealing in real life: an illustrating example

The previous description might seem vague and abstract, so lets conceptualize with a small example. Picture a newly filled bag of potato chips leaving the factory in a truck. When the truck speeds up on the highway, vibrations originating from engine, wind drag, and inconsistencies in the road will all propagate into the cargo compartment, and thus shake our bag of chips.

Shaking the bag is in fact a increase of its internal energy, (certainly not by heat since the cargo compartment is likely to be very cold!) but with kinetic energy making the chips jump around randomly in the bag. When the truck slows down, the vibrations will decrease, thus reducing the internal (kinetic) energy of the bag making the chips jump around less, and eventually stop moving totally finding some new (local minima) when the bag arrives at the shelf of your local grocery store!

When you open the bag in your kitchen, you'll find that the chips are stacked in a way that can almost surely not have not been as a result of just randomly dropping them into the bag. Which was the way they were put into the bag at the factory. The reason for this is simply that the truck ride is a example of a annealing process! Actually many producers of groceries that come in large containers of many small randomly ordered elements, such as coffee beans, flour and potato chips have disclaimers on their packaging, informing critical consumers that the reason for why the containers might look only half full is due to shaking of the container during transport! The minimal size of the container is limited by the volume necessary for filling the bag in a random fashion at the factory. This volume might be considerably larger than the necessary volume after annealing the potato chips!

2.11.5 Simulated annealing for parameter fitting

Shaking potato chips in a truck is certainly very far from the subject of minimizing KS distances of stock returns fitted Lévy processes, so how can this optimization algorithm be of any use to us? Well, in our case, the parameters of the Lévy process we want to fit will be the molecules in our system i.e. the "potato chips". The KS distance will be our measure of the internal energy of the system. We will increase this energy, i.e. raising the temperature of the system, by giving the parameters normally distributed random pushes. The standard deviation of these random pushes will be proportional to the temperature.

3 Structure of implementation

We have covered a lot of ground, and we have reached the point where it is time to start "sketching" the algorithm implementation. We will begin by defining our optimization problem, where we will measure the quality of a set of parameters in terms of its yielded KS distance with respect to the edf. We will then show how the KS distance is calculated. Further we will show how the simulated annealing algorithm is implemented, and finally make a short analysis of the implementation as a whole.

3.1 Formulation of optimization problem

The attentive reader will probably already have understood that our approach towards finding "good" parameters will be that of iteratively trying different candidate sets of parameters, and evaluating the "goodness" of these sets by calculating the KS distance of them with respect to the edf of the observed data. Thus we are ready to formulate our quest for "good" parameters as a global optimization problem!

Lets denote the set of daily returns of a set of stock prices over some time interval $[0, T]$ as

$$R_T = \{R(t), t \in [0, T]\}$$

And further we'll define a parameter set for a candidate Lévy process as

$$C_k = \{\mu_k, \sigma_k, a_{k,1}, \lambda_{k,1}, \ldots, a_{k,n}, \lambda_{k,n}\}$$

where n is the number of rescaled Poisson variables in the approximation. We can now formulate the KS distance as a function of the empiric data and a candidate Lévy process as

$$D_{ks} = f(R_T, C_k)$$

Since we are interested in finding the "best" fit in terms of KS distance, we will consider the problem to

$$\begin{gathered} \text{minimize } f(R_T, \mathbf{x}) \\ \text{subject to } \mathbf{x} \in S \end{gathered}$$

Where $S = \{C_k\}$ is the set of feasible parameters for a Lévy process and

$$f : \{\mathbb{R}^n, \{\mathbb{R}, \mathbb{R}^+, \{\mathbb{R}^+{}_i, \mathbb{R}^+{}_i\}_1^m\}\} \to \mathbb{R}$$

. Where n is the number of observed daily returns and m is the number of Poisson variables in the approximation.

Now we have not yet made any claims as to what $f(R_T, \mathbf{x})$ might look like, in this thesis we will only use numerical approximations to calculate the values of f, finding a analytical expression will be considered beyond the scope of this paper.

3.2 Calculation of Kolmogorov Smirnov distance

We will calculate the KS distance as a function of a candidate set of distribution parameters with respect to the edf of the daily returns. The algorithm looks as follows:

```
ndisc = with mu,sigma discretize normal density in n pieces
ndisc = Fast Fourier transform (ndisc)

For i in [1,m]
    pdisc[i] = with scale[m],lambda[m] discretize poisson in n pieces
    pdisc[i] = Fast Fourier transform pdisc[i]
    ndisc    = ndisc * pdisc[i]

L\'evy density      = inverse fast Fourier transform(disc)
L\'evy distribution = calculate cumulative sum (L\'evy density)
KS distance       = max difference (L\'evy distribution, empiric distribution)
return KS distance
```

Recall the formulation of the global optimization problem:

$$\begin{gathered}\text{minimize } f(R_T, \mathbf{x}) \\ \text{subject to } \mathbf{x} \in S\end{gathered}$$

Where $S = \{C_k\}$ is the set of feasible parameters for a Lévy process and

$$f : \{\mathbb{R}^n, \{\mathbb{R}, \mathbb{R}^+, \{{\mathbb{R}^+}_i, {\mathbb{R}^+}_i\}_1^m\}\} \to \mathbb{R}$$

. Where n is the number of observed daily returns and m is the number of Poisson variables in the approximation.

Even thou we can not formulate any analytical expression, the pseudo code above will be our f. It takes a candidate set of parameters and generates a KS distance. This function can be shown to be non-linear and non-convex. As a result of this, all optimization methods based on linearity properties and/or the convexity property is of no use to us, we must search for some other, global optimization method. Rather than using exhaustive search, we will turn our attention to the simulated annealing algorithm.

3.3 Algorithm pseudo code

We will illustrate how the algorithm works by describing its pseudo code:

```
params   := candidate set of parameters
ks       := KS distance of s
returns  := Daily returns to which we want to fit L\'evy process
```

```
bestfit  := Smallest KS Distance so far
pbest    := Best set of parameters so far
imax     := Maxiumum number of iterations
pnew     := newest candidate set of parameters
ksnew    := KS distance of pnew & returns
tk       := Temperature randomization scale

Algorithm:

params  = initial candidate;
ks      = KS distance of (params, returns)
pbest   = params
bestfit = ks
k       = 0

while k < imax and ks > ksmax

  pnew       = params +random bounce ~ N(0,sigma=tk*(kmax-k/kmax))
  ksnew      = KS distance of (params, returns)

  if ksnew < bestfit then
    bestfit = ksnew
    pbest   = pnew

  if P(ks, ksnew, temperature(k/kmax)) > uniform random number in [0,1]
    ks = ksnew; params  pnew

  k  k + 1

return bestfit and pbest
```

3.4 Algorithm behaviour analysis

The algorithm basically tests candidates of a normally distributed random distance from the current candidate set of parameters. This random distance will almost surely decrease in magnitude as the number of iterations grow. Since $\sigma \to 0$ as $k \to k_{max}$. That means that the algorithm will be very "wild" in its initial trial of candidates. During this phase of the algorithm, we are hoping to find some "nice" neighbourhood of the search space S with a lot of good solutions in it. Solutions that we can investigate more closely as $\sigma \to 0$.

The most interesting step however is the "if P > uniform random number in [0,1]", where

we define the acceptance probability as

$$P(KS_p, KS_c, T_c) = e^{-(\frac{KS_p - KS_c}{T_c})}$$

where KS_p is the KS distance of the current set of parameters, KS_c the KS distance of the new candidate and T_c the current temperature of the annealing process. This is the key step, it assures us that even if the randomly generated candidate generates a worse KS distance than the current one, we might still choose to use it as a new base of search for that "nice" neighbourhood of many good solutions.

4 Choice of software

We have now described the theoretical foundation for our search of good parameter fits. However the world around us is real, and even thou algorithms might look very appealing from a theoretical point of view, in order to produce good practical results, there is a lot of consideration to be made about implementation details. We can think of the theoretical background as a road map for finding our solution, now we must also walk it.

The mountain of K2 is located in the Karakourum mountain range on the border of Pakistan and China, it is widely considered as one of the toughest mountains in the world to climb with its summit at 8611 meters above sea level. From a theoretical point of view, 8611 meters is in all not that far, at K2 you might spend the rest of your life trying to cover this distance, and still not make it!

Computer implementations of mathematical problems can behave in just the same way, even the most stringent and clean looking mathematical problem formulations might take a universe lifetime (or 10) to calculate. Certainly we must assure that this will not be our destiny!

4.1 Requirements

So what requirements should we set so that we can find a descent solution without having to spend time we don't have? At the same time, we don't have all the time in the world to write this program, so we can't just write binary code manually. And since we are in the business of advancing the science of engineering we would like to use some modern software that takes advantage of the latest developments in computer engineering. Finally, it would also be good if as many users as possible could be able to harvest the fruit of all the work we put in to our program.

Lets formulate all these requirements in a more concrete way

1. Performance: we want our program to have fast execution time for reasonable input sizes.

2. Software reuse: we want to be able to combine already efficient implementations of existing code to a new powerful program. And hence save developer time (which is highly expensive compared to computing time).

3. Expansion and modification: We want a system capable of object oriented programming.

4. Fault identification & debugging: we want to use a programming language with strong debugging features so that maintainable will be kept easy.

5. Platform independence: Unlike the monumentally huge international soft ware empire, we would like to make our software available for anyone, not judging users by their choice of hardware or operating system.

4.2 Python

After taking all these requirements into account, the choice of implementing this optimization program fell on the Python programming language due to the following reasons:

1. Performance: Python supports the usage of compiled C-code. Using such code will almost certainly guarantee fast execution.

2. Software reuse: Python has a large set of pre-compiled functions available for free use, and in our case, the Numpy package (which we will introduce in just a moment) is of particular interest.

3. Expansion and modification: Python supports object oriented programming.

4. Fault identification & debugging: Since Python is a high level language, faults can easily be debugged and corrected.

5. Platform independence: Python code can be run on any platform for which there exists a interpreter. Even for users of "international empire"-software.

4.3 Scientific computing in Python: The Numpy package

The Numpy package is a open source library which adds support for large multi-dimensional arrays and matrices to Python. In addition to the added functionality of large arrays, Numpy contains a highly diverse set of mathematical algorithms for working with these arrays. One of these functions is the fast Fourier transform which we use to calculate our estimates. The program built in this thesis uses almost exclusively the data-structures provided by Nympy.

4.4 Performance boosting: Psyco

Psyco is a "Just-in-time"-compiler for Python which analyses Python-code and compiles it at run time. Algorithmic applications are more likely to benefit from the use of Psyco since they tend to be more processor-intensive. In our implementation the only drawback will be the extended use of RAM that Psyco causes. In our program, this will not be any problem since the amount of ram used by the program will be relatively small.

5 Result

There are two different type of results we would like to investigate, firstly we would like to investigate the actual computing performance of the algorithm implementation both from a theoretical and a experimental perspective. But more importantly, we would like to know the quality and precision of the produced results.

5.1 Program performance

We will start our analysis of program performance by deriving the asymptotic upper bound of its execution time, this will give us valuable clues as to what parts of the program that are "expensive" to execute, information crucial when deciding how to spend our effort when trying to improve performance.

However, in the end, for algorithms implemented as programs, experienced performance i.e. running times in seconds, is the ultimate measure of performance, and therefore we will also analyse some examples of real-life program execution.

5.1.1 Asymptotic upper bound

We will first establish the asymptotic upper bound of the algorithms execution time. And we will start by defining the total number of executions in terms of subroutine calls and their corresponding asymptotic upper bounds.

Well do this by an top down approach, recursively adding all different components and finally summarizing them in one complete formula from which we'll derive the complete asymptotic upper bound.

Let's start by denote the total number of comparisons required to complete one run of the program with $T(n)$. Where n corresponds to the size of input, i.e. the number of daily returns. Now, when initializing the algorithm we give a maximum number of iteration it might use to find a result. So in the simplest form:

$$T(n, Iteration_{max}) = k1 * Iteration_{max} * SA(n)$$

Where k is some constant representing the number of comparisons conducted during one iteration, $SA(n)$ is the time required to complete one iteration of the Simulated Annealing algorithm.

Now, this formula looks very appealing, but the $SA(n)$ does a lot of complicated calculations. And we must establish the cost of these:

$$SA(n) = k_2 * fft(n)$$

Where k_2 is some constant representing the number of comparisons conducted during one iteration of $SA(n)$, and $fft(n)$ is the cost of one FFT. Now, recall the asymptotic upper bound previously derived for the fft:

$$O(fft(n)) = O(nlog(n))$$

Hence we can now put together a comprehensive expression for the running time of the algorithm as follows:

$$T(n, Iteration_{max}) == k_1 * Iteration_{max} * k_2 * n * log(n)$$

The form of the expression above is somewhat unfortunate, since it is a function of two variables. And since we will alter the size of $Iteration_{max}$ in the empiric trials that we are about to undertake, we can not group it together with the other constant factors.

However, lets just for now assume that we will terminate the algorithm if it finds a KS distance $\leq$ some $KS\epsilon$, then for some sufficiently large p:

$$\forall Iteration_{max} \geq p : f(data, Iteration_{max}) \leq KS_{\epsilon}$$

holds almost surely.

If the above statement holds, there will be no need to use any larger number of iterations than p. At that point, $Iteration_{max}$ becomes constant, and we can group it into the other constant factors, yielding the following execution time:

$$T(n, Iteration_{max}) = k_{total} * n * log(n) \Rightarrow O(T(n, Iteration_{max})) = O(nlog(n))$$

However, in reality, we don't know what the number p might be, and lets instead assume that the it is of the somewhat same magnitude as n, we will define $m^2 = n * p$ then the bound looks as follows:

$$T(n, Iteration_{max}) = k_1 * k_2 * m^2 * log(m) \rightarrow O(T(n, Iteration_{max})) = O(m^2)$$

Which is a polynomial time upper bound. So, we have two different asymptotic upper bounds depending on how we choose to look at the $Iteration_{max}$ term. Now since we are interested in establishing a worst case execution time, we will finalize this statement by concluding that the asymptotic upper bound of growth for our algorithm is $O(n^2)$, however, do not despair! By our definition of efficient algorithms, the asymptotic upper bound of the algorithm is not at all that bad.

Inspecting $T(n)$ gives rise to a few interesting observations. Firstly, we can note that the $O(nlog(n))$ term will likely be our biggest cause of headache. I.e. the computation of the FFT. Since, in our application, this n happens to represent the size of the discretization mesh for our probability density functions, increasing mesh size will yield execution time increases of order $O(nlog(n)$. Increasing the number of maximum iterations, or the number of Poisson variables, will only cause execution time increases of order $O(n)$.

Poisson Variables	*750 days*	*1500 days*	*2500 days*	*5000 days*
2	0.0565	0.0994	0.2045	0.4597
4	0.0599	0.1034	0.2120	0.5258
6	0.0631	0.1079	0.2239	0.5920
8	0.0663	0.1108	0.2331	0.6580
10	0.0691	0.1148	0.2437	0.7192

Table 1: Execution time (in seconds) of 1 Simulated annealing iteration from the Dow Jones industrial average index

5.1.2 Time trial

Table 1 shows the execution time for each iteration of the simulated annealing algorithm. All tests has been conducted on a Intel Core2Duo dual core processor at 1.6GHz with 2Gb RAM. The operating system used was Darwin-BSD and Python 2.6 interpreter with Psyco JIT. In the tests, we have chosen to use a discretization mesh of the same size as the number of daily returns. The result shows clearly what we've expected from our analysis of the asymptotic upper bound. The size of the mesh is the main cause of increased execution time. Adding Poisson variables does seem to have some impact, but it does not cause increased execution time of the same magnitude as the increase of mesh size does.

5.2 Precision of estimates

Table 2 shows the KS distances achieved at different maximum iterations of the simulated annealing process. The presented values are the average of 10 runs. By using a cleverly picked initial candidate and 10 Poisson variables, we can start out at a point which is at least not too far away from the optimal fit.

The first line in the table, with only 10 iterations allowed corresponds to a "reasonable" initial guess. With 10 iterations, it is rather unlikely that we will be able to find some good set of candidates, and further, since we give the variables normally distributed random pushes, proportional to the temperature of the system, the parameters in the candidate set will not be able to move that far away from the initial guess.

What can also be established from the table is that very large set of data does not seem to improve the result of the fit. And more importantly, what is really the key factor in finding a

SA iterations	*750 days*	*1500 days*	*2500 days*	*5000 days*
10	0.0735	0.0750	0.0732	0.0691
100	0.0646	0.0688	0.0691	0.0675
200	0.0453	0.0466	0.0437	0.0473
500	0.0390	0.0267	0.0271	0.0294
1000	0.0164	0.0120	0.0134	0.0250
2000	0.0137	0.0096	0.0099	0.0146

Table 2: Kolmogorov Smirnov distance of estimates

good set of parameters is to let the algorithm work through a lot of feasible solutions. If we compare the 2000 iterations fit of 750 days of returns, the fit is of almost the same magnitude as that of 2500 & 5000 days. But comparing the entries of table 1, we see that the earlier result took less than 10% of the time to calculate!

6 Conclusions

6.1 Improving performance

By looking at the time trials and comparing these to the precision of the estimates, we can clearly see that increased resolution in the discretization of the density functions impact performance in a highly negative way. At the same time however, such an increase does not seem to result in any obvious reduction in size of the Smirnov distances.

So if trying to improve an algorithm like this one, it would seem reasonable to spend energy on improving the optimization algorithms used to find the approximations. For example, if finding a better way of locating candidate parameter sets, the generated "spare" time will have a much larger impact, on the result of the calculation, if used to increase the number of iterations in the simulated annealing algorithm. It will almost surely yield a better "pay-off" in terms of preciser estimates of the KS distance than using the available-made time for increasing the resolution of the discretization.

6.2 The usability of high level programming languages

With the aid of compiled C-code, from the Numpy scientific computing library for Python, we have seen that even a high level programming language can actually be used for numeric approximation. Certainly this program would most likely have been out performed by a similar implementation built directly in C, but then we would also have lost a lot of the functionality made available through the platform independent alternative.

The results calculated in this Python program could, for example, with ease have been connected to a variety of other software, in which this estimation method could have contributed with alternative risk estimates.

7 Future research topics on the subject

There are of course a lot of open topics left uninvestigated, which could be the subject of future research. We will present some suggestions in a comprehensive list:

- Search space: The search space, i.e. the feasible set of solutions for the distribution parameters. This space might contain interesting characteristics that could be used to build more efficient algorithms for finding optimum fits. Maybe the problem can even be reformulated as a convex one!
- Global optimization: There might be more efficient algorithms that can identify optimal solutions in a more efficient way. Harmony search is a interesting example.
- Choice of Program Language: This algorithm could be implemented in some other language in order to compare performance.

References

[1] Duffie, D. and Pan J. (1997) *An overview of value at risk,* The journal of Derivatives 3, 7-49.

[2] Schoutens W. (2003). *Pricing Financial Derivatives Lévy Processes in Finance*. Wiley, Chichester.

[3] Trolle, U. (2005). *On semi-parametric modelling of stock prices with Lévy processes*. Master's thesis, Department of Mathematical Statistics, Chalmers University of Technology.

[4] Andreasson, N. ,Patriksson, M. and Evgrafov, A. (2005). *An introduction to continuous optimization*. Studentlitteratur, Lund.

[5] Kleinberg J. and Tardos E. (2006). On asymptotic upper bounds. *Algorithm Design*. Pearson, New York.

[6] Zill, D.G. and Cullen, M.R. (1999). *Advanced Engineering Mathematics, 2nd edition* 750–764. Jones and Bartlett. Boston