NIG-Lévy approximations and copula based bivariate option pricing

by

TORGEIR W. B. HOFFMANN

MASTERTHESIS

for the degree

Master of Science in Modelling and Data Analysis

(Master of Science)

Det matematisk- naturvitenskapelige fakultet
Universitetet i Oslo

March 2009

Faculty of Mathematics and Natural Sciences
University of Oslo

Acknowledgements

Let me start by thanking my supervisor Fred Espen Benth for providing me with an interesting project. His patience and quick response have been critical to the development of the ideas presented herein.

Varm thanks to Paul C. Kettler. Without his good advice and frequent discussions, this thesis may have been very different. I am very grateful for all that he has done.

Last, but not least, I would like to thank my family and friends, who provided me with support when things looked darkest.

March, 2009

Torgeir W. B. Hoffmann

Abstract

We aim to study how copula based dependence modelling of jump sizes in the NIG-Lévy model affect the option price. This leads to the definition of a rejection based approximation algorithm for the NIG-Lévy.

Using small jumps approximation provided by theory from Asmussen and Rosinki, an automatic algorithm is developed. Furthermore, the empirical copula of two dependent processes is extracted and analysed. In general, it can be observed that the small jumps dominate the dependence structure, as well as the behavior of the process. An observation with regards to thin tails is made: The dependent jumps produced by the conditional copula distribution vary a lot even in the 4th or 5th significant digit. Therefore, one can see indications that, for certain parameter sets, copulas with sufficient spread in the tails may produce very large jumps in one marginal process, while the other experience very small jumps.

Finally, a discussion on the effect of the copula on option prices is presented, and compared to the option price of independent processes. An approach to finding the Esscher parameters is presented, and the issues using an Esscher transform on dependent processes are presented.

The findings indicate that copulas on large jumps can have a bigger effect on the option prices if dependency is low, as variation in large jumps can be bigger for the dependent process than the original process. For high dependency, the option prices are generally lower than in the independent case, as dependence restrict the movement of the second process.

All simulations were done using R (http://www.r-project.org). Plotting was done using R and gnuplot (http://www.gnuplot.info).

Contents

Chapter 1

Lévy processes, approximation and simulation

1.1 Leaving the Black-Scholes world behind

The Black-Scholes theory was no doubt one of the most important breakthroughs in modern mathematical finance. It builds on the assumption that the log-returns for a given risky security follow a Gaussian distribution. While it is easy to model and implement, it lacks flexibility and has tails that are far too light to describe the distributions of observed daily log-returns.

The framework is rather inflexible in view of the restrictions such as constant risk-free interest rate, drift and volatility, the absence of transaction costs, and the assumption that one can buy any quantity of a share, no matter how small. However, these are restrictions that one can argue to be necessary if the mathematical model is to have some level of simplicity. On the other hand, the model can be fitted with a heavier tailed distribution. This improves the fit to empirically observed data and risk evaluation.

1.1.1 The fat-tails problem

One of the primary concerns of risk management is heavy tails. Clearly, a model that describes more accurately the tails of a distribution of returns will have big impact on the portfolio of an investor or company. If the tails are too light the risk of big losses is underestimated; the losses will occur more frequently and can lead to very grave results indeed. On the other hand, overestimating the risk will lead to funds not being allocated as efficiently as possible, and the investor can miss opportunities of increased returns on his portfolio.

This calls for a more flexible model, which can more accurately capture the behavior observed. Many heavy tailed distributions exist, among them the Cauchy distribution and Pareto. One should remember that the distribution should be flexible enough to capture skrewness as well. The normal inverse

Gaussian is such a distribution.

1.1.2 Normal inverse Gaussian distribution

The NIG distribution is much more flexible than the commonly used Gaussian distribution, and is able to reflect both semi-heavy tails and skewness in the distribution. Semi-heavy tails are in general classified as being heavier than those of the Gaussian distribution, but lighter than those of α-stable distributions such as the Cauchy.

However, this comes at a cost of more parameters. This is not desireable from a modelling point of view. Even if a model fits very well to the observed returns, the cost of modelling and fitting more parameters might be too high compared to the gain. However, the NIG distribution strikes a good balance with four parameters, and a good fit to empirically observed data. While the NIG was introduced by Barndorff-Nielsen to model the size of gains of sand, it has proven to have very good fit in several other applications, among them financial markets. See Barndorff-Nielsen [5] and the references therein to Shephard [21].

The distribution function is defined as:

$$\mathrm{nig}(x;\alpha,\beta,\delta,\mu) = c\cdot\exp\{\beta(x-\mu)\}\frac{\mathrm{K}_1(\alpha\cdot g(x-\mu))}{g(x-\mu)} \tag{1.1}$$

where $g(x)=\sqrt{\delta^2+x^2}$ and K_1 is the modified Bessel function of 2nd kind order 1[1]:

$$\mathrm{K}_1(x) = \frac{1}{2}\int_0^\infty \exp\{-\frac{1}{2}x(z+z^{-1})\}\,\mathrm{d}z \tag{1.2}$$

and the constant c is given as:

$$c = \frac{\delta\alpha}{\pi}\exp\{\delta\sqrt{\alpha^2-\beta^2}\}$$

Here the parameters of the NIG distribution have the following effect:

- α: tail-heaviness/shape
- β: skewness. $\beta>0$ means skew to the right, $\beta<0$ left, $\beta=0$ symmetric.
- δ: scale parameter, similar to variance, controls pointiness of distribution.
- μ: location

For the distribution to have the desired qualities we need to put some additional restrictions on the parameters [3, 1.3.32]: $\alpha\geq|\beta|\geq 0$ and $\delta>0$.

[1] Referred to as the modified Bessel function of 3rd kind order 1 by some authors, but this is now less common.

The moment generating function of the NIG is easily derived by simple algebra as:

$$M(u) = e^{\mu u + \delta(\sqrt{\alpha^2-\beta^2} - \sqrt{\alpha^2-(\beta+u)^2})}$$

This is interesting and will be returned to in Chapter 3, but it should be noted that it has a very simple form. This is useful from a modelling perspective and especially with respect to option pricing since it is easy to work with.

1.2 Lévy processes

Definition 1.2.1 (Lévy process)**.** Lévy processes are a special class of stochastic processes with the following properties:

- Stochastically continuous
- Stationary with independent increments

1.2.1 Tail integrals and Lévy measures

From Cont and Tankov [9, 5.7] we have:

Definition 1.2.2 (Tail integral)**.** A d-dimensional tail integral is a function $U : [0, \infty] \to [0, \infty]$ such that

1. $(-1)^d U$ is d-increasing, i.e. $\Delta_{x_1}^{x_2} \Delta_{y_1}^{y_2} U(x, y) \geq 0$
2. $U = 0$ if one of its arguments is equal to ∞
3. U is finite almost everywhere, $U(0, \ldots, 0) = \infty$

In other words the tail integral has many similarities to the survival probability function, but the former is more generalized. A survival probability is the tail integral of a probability distribution, and the tail integral plays a very specific role with the Lévy measure. From Cont and Tankov [9, 3.4]:

Definition 1.2.3 (Lévy measure)**.** Let $(X_t)_{t \geq 0}$ be a Lévy process on $\mathbb{R}^d$. Define the Lévy measure ν on $\mathbb{R}$ by:

$$\nu(A) = E[\#\{t \in [0, 1] : \Delta X_t \neq 0, \Delta X_t \in A\}], \qquad A \in B(\mathbb{R})$$

Here we can find an interesting connection. The tail integral of the Lévy measure from a point x is the intensity, or the expected number of jumps with size greater than or equal to x in a unit interval of time. If the Lévy measure is finite, then the intensity is naturally finite everywhere.

1.2.2 Lévy-Itô decomposition

This celebrated result is fundamental when working with Lévy processes, and gives us great insights into the building blocks of the process. Before the theorem, a definition is in order:

Definition 1.2.4 (Poisson random measure)**.** Let $\mathbf{B}_0$ be all Borel sets not containing 0. Let the increment of a Lévy process be $\Delta l(t) = l(t) - l(t^-)$. Then the Poisson random measure of $l(t)$ is defined as:

$$N(t, U) = \sum_{s:0\leq s\leq t} \chi_U(\Delta l(t))$$

From Applebaum [3, 2.4.16] the results can now be stated:

Theorem 1.2.5 (The Lévy-Itô decomposition)**.** *If X is a Lévy process, then there exists $a \in \mathbb{R}^d$, a Brownian motion B_σ with covariance matrix σ and an independent Poisson random measure N on $\mathbb{R}^+ \times (\mathbb{R}^d \setminus \{0\})$ such that for each $t \geq 0$:*

$$X_t = bt + B_A(t) + \int_{|x|<R} x\,\widetilde{\mathrm{N}}(t,\,\mathrm{d}x) + \int_{|x|\geq R} x\,\mathrm{N}(t,\,\mathrm{d}x)$$

R can be arbitrary, but it is common to let $R = 1$.

Here $\widetilde{\mathrm{N}}(t,\,\mathrm{d}x) = \mathrm{N}(t,\,\mathrm{d}x) - t\nu(x)$. This compensation is necessary where the intensity of the small jumps is infinite and for $\widetilde{\mathrm{N}}(t,\,\mathrm{d}x)$ to be a martingale (See Øksendal and Sulem, 2004).

The triplet (a, σ, ν) is commonly referred to as the Lévy triplet, and consists as above of a drift a, covariance matrix σ and a Lévy measure ν.

This means that the basic building blocks of a Lévy process is a drift term and a Brownian motion for the continuous part. Furthermore, we have a term for the small jumps and a term for the big jumps.

1.2.3 Lévy-Khinchin

This theorem is a very important since it tells us that the small jumps in the Lévy-Itô decomposition are actually independent of the big jumps.

Theorem 1.2.6 (Lévy-Khinchin)**.** *Let X_t be a Lévy process with Lévy triplet (a, σ, ν). Then*

$$\int_{\mathbb{R}\setminus\{0\}} \min\{x^2, 1\}\,\nu(\mathrm{d}x) < \infty$$

Furthermore,

$$\mathrm{E}\{e^{iuX_t}\} = e^{t\psi} \tag{1.3}$$

$$\psi = aiu + \frac{1}{2}\sigma^2 u^2 + \int_{\mathbb{R}\setminus\{0\}} \left(e^{iux} - 1 - iux\chi_{|x|<1}\right)\nu(\mathrm{d}x) \tag{1.4}$$

This is important from a modelling perspective. It gives an explicit shape for the characteristic function and determines an important relationship between the small and big jumps: independence. This property will be actively used in modelling. It makes it possible for us to model the dependency between the big jumps of two processes without regard to the connection between the small jumps and vise-versa. This relationship can be confirmed using Kac's theorem [3, 1.1.15].

From this we also get useful notation that can be applied to all Lévy processes: the Lévy triplet[2] (a, σ, ν), where a is a constant, σ is a square matrix and ν is the Lévy measure. This triplet uniquely determines the Lévy process.

1.2.4 NIG-Lévy process

The NIG-Lévy process has its increments distributed as: $X_{t-s} \sim \mathrm{nig}(\alpha, \beta, \delta(t-s), \mu(t-s))$.

From Barndorff-Nielsen [4, 4.11-4.13] we have that the NIG-Lévy process can be represented by a triplet $(a, 0, \nu)$, where

$$a = \mu t + \frac{2\delta t\alpha}{\pi}\int_0^1 \sinh(\beta x)\,\mathrm{K}_1(\alpha x)\,\mathrm{d}x$$

$$\nu(x) = \frac{\delta t\alpha}{\pi|x|}e^{-\beta x}\,\mathrm{K}_1(\alpha|x|) \tag{1.5}$$

Note that the Lévy measure is independent of μ. Important to note is that from the Lévy triplet we have that our NIG-Lévy process is in fact a pure jump process. In other words it has no continuous part represented by a Brownian motion as indicated by the Lévy-Itô decomposition (1.2.5). This will have consequences for our modelling, and in particular for option pricing since a market driven by such processes give rise to an incomplete market. We are dependent on finding a risk neutral probability measure to prevent arbitrage from existing. We will show how to work with this in Chapter 3.

1.2.5 Properties

We will derive some interesting properties of the NIG-Lévy process. For this we will employ several propositions found in Papapantoleon [14]:

[2]Some authors refer to it as the characteristic triplet

Let X_t represent our Lévy process with Lévy measure $\nu(x)$ given as in (1.5). If $\nu(\mathbf{R}) = \infty$, then almost all paths of X_t have an infinite number of jumps on every compact interval [14, Prop. 7.1]. In that case, the Lévy process has infinite activity. This is easily seen from the form of the Lévy measure.

Furthermore, we have that if

$$\int_{|x|\leq 1} |x|\, \nu(\mathrm{d}x) = \infty,$$

then almost all paths have infinite variation [14, Prop. 7.2]. This tells us something about how the small jumps impact on the behaviour of the process, and in particularly that it is dominated by the small jumps. We can also verify this with relative ease as the asymptotic behavior of $\mathrm{K}_1(x)$ is $K_1(x) \sim x^{-1}$ when $x \to 0$. We will look closer at this in our approximation section (1.3), but refer to either Abramowitz and Stegun [1] or Barndorff-Nielsen [4, 4.14] for details.

The final property that we will look at is that of moments. If

$$\int_{|x|\geq 1} |x|^p\, \nu(\mathrm{d}x) < \infty$$

then X_t has finite p-th moment [14, Prop. 7.3]. In our case, this means that since the Lévy measure has no other atoms, and $\lim_{x\to\infty} \nu(x) = 0$ then the NIG-Lévy process has all moments finite. This is very useful, as it means that one can find expectation and variance. Variance and infinite variation need to be distinguished. While the former is commonly known as the second moment, the property of infinite variation means that at every time interval there is a probability of making large enough moves such that one cannot find an upper or lower bound for the increments on any compact interval.

1.3 Approximation of the Lévy processes

Looking at the possiblities of approximating the NIG-Lévy is of interest, as an opproximation on the form of the Itô-Lévy decomposition will enable modelling of the drift, brownian diffusion, small jumps and large jump independently. This allows for great flexibility, in particular in dependence modelling as will be covered in Chapter 2

From the Barndorff-Nielsen representation of the NIG-Lévy process, and that the integral in the Itô-Lévy decomposition can be approximated by a compound Poisson process where the intensity of the jumps are given by the integral of the Lévy measure over its entire support.

This leads to the following expression for the NIG-Lévy using the notation from (1.5):

$$L(t) = at + \sum_{i=1}^{N}(t)\gamma(t)$$

However, it should be noticed that the NIG-Lévy as mentioned has infinite activity in the origin, and hence the theoretical intensity of the Possion process, $N(t)$, is infinite. For the small jumps, it is therefore necessary to take some extra precautions.

1.3.1 Approximation of the jump size distribution

One approach is to approximate the jump size distribution. The idea is simple:

In many cases the Lévy measure is finite, and hence:

$$K = \int_{\mathbf{R}} \nu(\mathrm{d}x) < \infty$$

Therefore, we can define the probability measure of a jump, J, as:

$$P(J \in U) = \frac{\nu(U)}{K}, U \subset \mathbb{R}$$

From this, one can in theory simulate the jumps of a Lévy process. Combined with the power of the Lévy-Itô decomposition, the process has a complete representation.

In general, however, a direct simulation in this way contains several pitfalls. As outlined in 1.2.5 our NIG-Lévy process has infinite activity, so that a direct normalization is not possible.

Define small jumps as increments which lie in the interval $(-\epsilon, \epsilon)$. Jumps outside this interval are considered large jumps. One possibility is simply to remove the small jumps, but in our case Barndorff-Nielsen shows that the small jumps are actually dominating the movement of the process. The result is due to a series representation of the process, and the rate at which it goes to infinity near the origin. Since the process has sufficient mass near the origin, it can be approximated.

Approximation of jumps smaller then ϵ

Instead of removing the small jumps of size ϵ or less, Asmussen and Rosinski[20] propose to incorporate their contribution by a Brownian motion with the following mean and variance:

$$\mu_\epsilon := -\int_{\epsilon \le |x| \le 1} x\, \nu(\mathrm{d}x), \qquad \sigma^2(\epsilon) := \int_{|x|<\epsilon} x^2\, \nu(\mathrm{d}x) \tag{1.6}$$

The Brownian approximation of small jumps exploits the result that expected value of the small jumps divided by $\sigma^2(\epsilon)$ converges in distribution to a standard Brownian motion. For details, see the original article referenced above.

One interesting part to note is that even if the process is a pure jump process, the approximation still gives a Brownian part. The result is valid for the NIG-Lévy process [20, Prop. 2.2]. For all practical purposes we will not be able to

distinguish between the small movements of such a Brownian motion and that of a NIG-Lévy process. The error of this approximation will be dealt with in Section 1.4.1.

Dealing with jumps larger than ϵ

Since the small jumps are now dealt with, we truncate the Lévy measure in a small neighbourhood of the origin $(-\epsilon, \epsilon)$, and since the Lévy measure of the NIG-Lévy has no other atoms, the normalization constant is finite, even if it's still not known. Hence it would be possible to simulate the jumps directly from the the normalized truncated Lévy measure, $\nu_\epsilon(x)$, if we can find the normalizing constant. The constant is not hard to find numerically, but can be quite a challenge analytically. The normalizing constant, from here on referred to as K_ϵ, is given as:

$$K_\epsilon = \int_{\epsilon<|x|<\infty} \nu(\mathrm{d}x) = \int_{\epsilon<|x|<\infty} \frac{\delta\alpha}{\pi|x|} e^{-\beta x} \mathrm{K}_1(\alpha|x|)\,\mathrm{d}(x)$$

Using a Lévy measure truncated around the origin one can avoid discretizing the measure, and hence keep the advantage of the heavy tails. In other words, theoretically one can have draws from the entire support of the jump probability measure, and the only limitation would be that of the computer hardware used for simulation.

Intensity of jumps larger than ϵ

In the general case, the itensity of the jumps are equal to the normalization term in the NIG-Lévy jump measure. This is also the case here where the intensity λ is such that: $\lambda = \nu^\epsilon((-\infty, -\epsilon] \cup [\epsilon, \infty))$.

The number of jumps in the compound Poisson process are distributed by a Poisson random variable with the intensity λ given above. The waiting time between each jump is exponentially distributed [9, Def. 2.17]. The increasing sum of exponentially distributed random variables follows a Dirichlet distributed. This can be simulated using a sorted set of uniform draws on $[0, T]$. For more information on this, see Cont and Tankov [9, Prop. 2.10]

Note that the larger jumps are not intended to be approximated. Since the intensity is finite direct or indirect simulation is preferred.

1.4 Simulation of Lévy processes

The expressions for the mean and variance of the Brownian part given in (1.6) are finite integrals that we can easily estimate numerically by employing Riemann integration with a sufficient discretization, or any other optimized integration method in our software.

To draw the large jumps directly using a pseudo-inverse method is not desirable since one cannot easily invert the modified Bessel function K_1 analytically, even if K_1 is strictly decreasing and C^1. A search algorithm (e.g., bisection) on a spline would be inaccurate unless one chooses a large number of knots. In addition, a spline on an unbounded domain will be difficult to implement without imposing artificial boundaries. For high intensity, this method can become very inefficient.

From the definition of the tail integral of the truncated Lévy measure, ν^ϵ

$$\lim_{x\to\infty} \nu^\epsilon(x) = 0$$

Therefore, it is at least bounded as there are no other atoms save the one in the origin. A function that dominates it on this interval can be found. We can find the asymptotes for our Lévy measure by studying the asymptotes of the Bessel function given in Abramowitz & Stegun[1, 9.6.9] and Barndorff-Nielsen[4, 5.1] repectively:

$$\begin{aligned} \nu(x) &\sim \frac{\delta}{\pi x^2}, && x \to 0^+ \text{ or } x \to 0^- \\ \nu(x) &\sim \delta\sqrt{\frac{\alpha}{2\pi|x|^3}} \cdot \exp\{\beta x - \alpha|x|\}, && x \to \pm\infty \end{aligned}$$

We will employ a rejection based sampling methods to draw from our distribution. Using conventional rejection sampling one would need to know either the normalizing constant or at least an upper bound for it[3]. Metropolis-Hastings algorithm, on the other hand, does not require the normalizing constant to be known. Another very appealing feature of the rejection based algorithms is that one can avoid the inversion method described above completely. Two algorithms are presented.

The Accept-Reject algorithm

The first algorithm presented is the conventional Accept-Reject method. It is based on the idea that the height of the proposal distribution, $\mathrm{q}(x)$, and that of the target distribution, $\pi(x)$, is compared for each draw proposed. If the proposal distribution has a lot more mass than the target for a a jump, then there is only a low chance that the jump is accepted as a draw. However, if the distributions resembles one another for some areas in the support chance of acceptance in those regions is higher.

Define α as

$$\alpha = \frac{\pi(y)}{c\mathrm{q}(y)}$$

[3]Hogstad and Omre, A comparison of Rejection Sampling and Metropolis-Hastings Algorithm.

where y is the proposal drawn and c is a majorizing constant such that

$$\pi(y) \leq c\mathrm{q}(y), \quad \forall y \in \operatorname{supp} \pi$$

Since $\alpha \in [0, 1]$, the algorithm rejects with probability $1 - \alpha$. If rejected, the previous jump is kept. Clearly, the chain of accepted draws determines a random walk with a move probability of α. The question that springs to mind is the rate of convergence to target distribution. As can be seen, the algorithm does not take into consideration anything else, save the height of the distributions. Potentially, this could make the rate of convergence slow.

It is critical that one can find a constant c such that the proposal distribution dominates the target over the entire support. Otherwise, areas may be undersampled and one can end up with automatic acceptance in others since $\alpha = 1$. This can lead to extreme undersampling in parts of or the whole target distribution. Still, this may not be enough as some distributions are simply to light tailed to be used efficiently as proposal distributions in some cases. For example, it is very difficult to sample a Cauchy distribution using a Gaussian distribution as proposal because of the light tails in the latter. The converse, on the other hand, works very well.

In addition, it is necessary to know the target distributions normalization factor, or at least an upper bound for it. If not, one cannot guarantee that a c can be found that will allow for optimal convergence of draws to target distribution.

However, there are algorithms that can get around these restrictions, and this is particularly useful for high-dimension problems where the normalization constant is very computationally demanding.

Provided under results, Table 1.2 compares draws generated through the Accept-Reject algorithm with those of a real NIG-Lévy simulation. The results also cover different proposal distributions.

Metropolis-Hastings algorithm

The second rejection based algorithm presented is the Metropolis-Hastings algorithm, which by the construction of a Markov chain can be used to draw realizations of our target distribution , $\pi(x)$, which is the normalized truncated Lévy measure of our NIG process. Let $\mathrm{q}(x)$ denote the proposal distribution. We here employ the same notation as in the excellent paper by Chib and Greenberg [8], but we only consider proposal distributions which are independent of the previous jump. Hence, if two draws are made from $\mathrm{q}(x)$: x_1 and x_2 then the probability mass of x_2 would be $\mathrm{q}(x_2)$. This is slightly different from the original algorithm by Hastings, where the law of the second jump depended on that of the first, hence the Markov chain description. The algorithm is given in Algorithm 1.4.1.

The algorithm is rather simple and evolves around the relative mass (or height of probability distribution) of two draws. The acceptance probability, α, is defined as the fraction of the relative mass in the proposal distribution, $\mathrm{q}(x)$, and $\pi(x)$:

$$\alpha(x_1, x_2) = \min\left(1, \frac{\pi(x_2)\mathrm{q}(x_1)}{\pi(x_1)\mathrm{q}(x_2)}\right)$$

As can be seen, a higher acceptance probability is given if the shape of $\mathrm{q}(x)$ is close to that of $\pi(x)$. Furthermore, if the target distribution is continuous and differentiable then one would benefit slightly from using a proposal distribution with these properties compared to a discontinuous one.

On the other hand, if no proposal distribution with good shape is available , one can instead choose a mixture of continuous functions which fit well on mutually disjoint sets in the support of $\pi(x)$ as long as the support of the mixture contains that of the target distribution.

This will of course only help on accept probabilities of jumps between these areas, and α might still be small for jumps from the same continuous area. However, since Metropolis-Hastings is much more dependent on the overall shape of the distribution, one can be severely punished for choosing a proposal distribution with poor fit. This can in general lead to very slow convergence due to low rates of acceptance.

Algorithm 1.4.1 (Metropolis-Hastings)**.**

procedure METROPOLIS-HASTINGS(π, F, N)
  Draw y from F
  for $n \leq N$ **do**
    Draw x from F and u from U(0,1)
    Define:

$$\alpha = \begin{cases} \min\left(\frac{\pi(x)\mathrm{F}(y)}{\pi(y)\mathrm{F}(x)}, 1\right) & , \pi(y)F(x) > 0 \\ 1 & , \text{otherwise} \end{cases}$$

    if $u < \alpha$ **then**
      Set $y = x$
    end if
  end for
end procedure

In general, one can refer to a Markov Chain generated by the Metropolis-Hastings as running too cold or too hot depending on whether the proposal distribution is too heavy or too light in the tails respectively. If the chain runs too cold then the fraction $\mathrm{q}(x_2)/\mathrm{q}(x_1)$ will be close to one. If the tails are much heavier in the target distribution, this will result in a high number of rejections in the tails, and undersampling of high probability areas. An illustration of the under and oversampling for two Gaussian distributions are given in Figure 1.1.

Vise versa, if the proposal distribution runs too hot it will too often propose draws from high probability areas. It can therefore take a long time for the chain to traverse the support of $\pi(x)$.

Either of these situations can lead to chain converging slowly to a draw from the target distribution. Furthermore, in both these cases high autocorrelation is likely over the values sampled at the end of each chain[8].

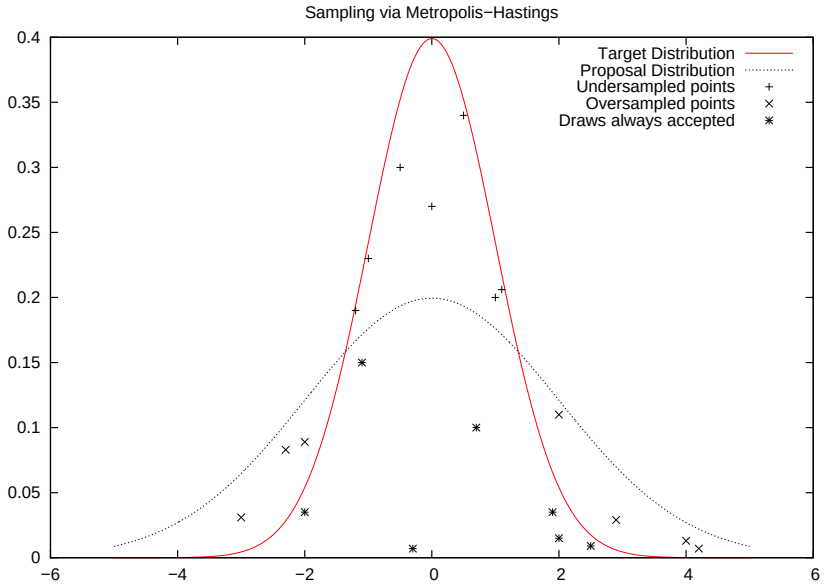

Figure 1.1: Illustration of sampling via Metropolis-Hastings

It is important that the support of the proposal is such that:

$$\text{supp}(\pi) \subseteq \text{supp}(q)$$

If the support of the target distribution is not connected one must ensure that the proposal distribution can at least simulate from the the whole support such that the chain can make the transition from one state to another state in an unconnected part of the support. If this is not possible one may experience mixing problems in the accepted draws.

Clearly, the Metropolis-Hastings algorithm comes at a tradeoff. When using proposal functions with imperfect fit it is highly likely that some areas will be undersampled to some degree or oversampled. Rejection-based algorithms compensate for this run running an increased number of iterations to ensure a good convergence. This is by many still considered an open question although some work has been done in the area, in particular by Raftery and Lewis [15, 16]. A small analysis on this is presented in Section 1.4.3.

In general the chain of accepted draws will be dependent, and therefore we can in principle only use one realization from each chain as an i.i.d. sample [19, p. 234]. This does only in minor degree affect the computational efficiency of our algorithm since few things need to be reinitialized on each run.

Implementation of rejection schemes

The implementation that is quite close to that proposed by Asmussen, Rasmus and Wiktorsson[6]. To analyse the efficiency of the algorithms, two proposal

distributions $\mathrm{J}(x), \mathrm{J}^1(x)$ are chosen. They are probability mixtures as follows:

$$J(x) \sim \begin{cases} w_q \cdot f_q & , x \in [-\omega, -\epsilon] \cup [\omega, \epsilon] \\ w_e \cdot f_e & , x \in (-\infty, -\omega) \cup (\omega, \infty) \end{cases} \tag{1.7}$$

$$\mathrm{J}^1(x) \sim \begin{cases} w_q \cdot f_l & , x \in [-\omega, -\epsilon] \cup [\omega, \epsilon] \\ w_e \cdot f_e & , x \in (-\infty, -\omega) \cup (\omega, \infty) \end{cases} \tag{1.8}$$

We will onwards refer to $\mathrm{J}(x)$ as the inverse quadratic case, and $\mathrm{J}^1(x)$ as the inverse linear case. The distribution functions are given as:

$$\begin{aligned} f_e &\sim (\alpha - |\beta|) \exp(-(\alpha - |\beta|)|x - \omega|) \\ f_q &\sim \left(\frac{1}{\epsilon} - \frac{1}{\omega}\right)^{-1} \frac{1}{x^2} \\ f_l &\sim \log\left(\frac{\omega}{\epsilon}\right) \frac{1}{x} \end{aligned}$$

These can very easily be inverted for simulation. Figure 1.2 illustrates two possible setups with the proposal functions given above. The inverse quadratic distribution and Laplace distribution[4] have weights w_q, w_e respectively. The weight for the inverse linear, f_l, is the same as the former. These can be chosen to be the intensity of the Lévy measure over respective intervals:

$$\begin{aligned} w_e &= \left(\int_{-\infty}^{-\omega} \nu(x)\,\mathrm{d}x + \int_{\omega}^{\infty} \nu(x)\,\mathrm{d}x\right) \cdot K_\epsilon^{-1} \\ w_q &= \left(\int_{|x| \in [\epsilon, \omega]} \nu(x)\,\mathrm{d}x\right) \cdot K_\epsilon^{-1} \end{aligned}$$

Note that manual tuning of the weights is not necessary. In particular, if $J(x)$ is used as a proposal function, these weights are almost natural due to the very small spread. However, with a looser fit in the proposal function as is the case with $\mathrm{J}^1(x)$ proposal function, manual tuning might be necessary due to poor fit in the tails. However, such manual tuning will only result in less draws from the high probability areas of the proposal, and may in many cases then lead to undersampling. Therefore, the same weight is used in both cases.

As for choosing the truncation levels ϵ and ω we will deal with this in (1.4.1).

Furthermore, one is free to use a set of scaling constants $c_1, \ldots, c_4$ to increase the acceptance rate. We choose the constants to be such that the proposal distribution is at least continuous:

[4]Commonly known as Double Exponential Distribution

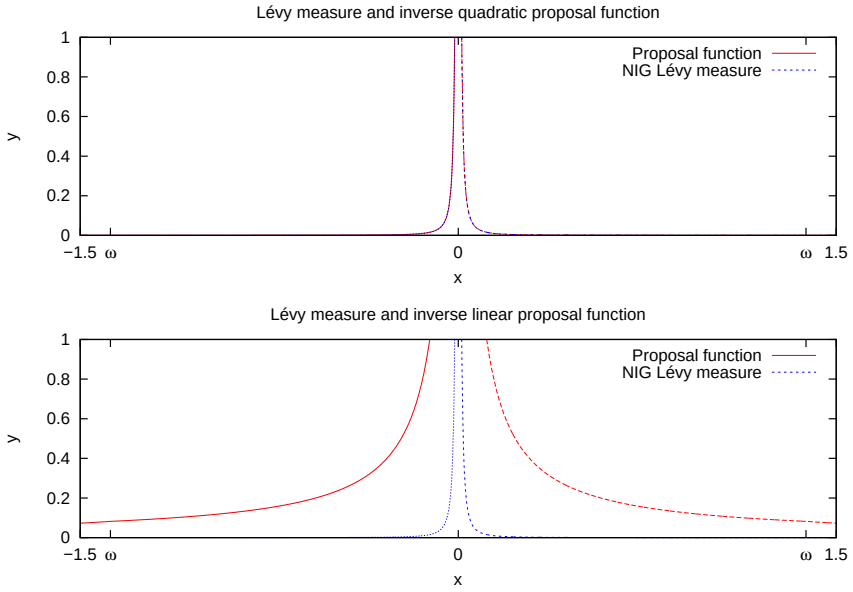

Figure 1.2: The NIG Lévy measure with proposal functions

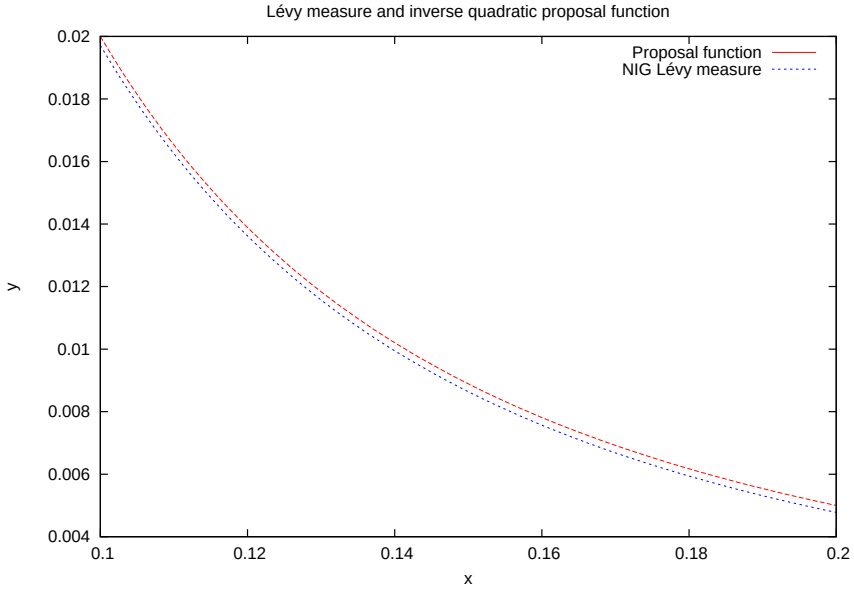

Figure 1.3: A close up look at the NIG Lévy measure with inverse quadratic proposal

$$c_3 = \delta \cdot \alpha$$
$$c_4 = \frac{c_3 \cdot f_q(\omega)}{f_e(\omega)}$$

The constants for the negative half-line can be chosen in similar manner using the negative truncation point $-\omega$. This can have some effect on the acceptance rate for jumps drawn from the Laplace area of the proposal. This can benefit both rejection algorithms. Note that c_3 has as the main purpose to ensure domination of the Lévy measure in the Accept-Reject case, and hence the Metropolis-Hastings will not benefit from this.

On optimal acceptance rates in Accept-Reject

While acceptance rates are not expected to be high when using $\mathrm{J}^1(x)$, they can be extremely high due to very small spread between the Lévy measure and $\mathrm{J}(x)$. As long as $\mathrm{J}(x)$ can adequately sample the entire support of the Lévy measure, there should not be much of an issue.

One should keep in mind that high accept rates in other cases may not be optimal. Roberts, Gelman and Gilks (1994) showed theoretically that if both proposal and target densities are Gaussian, the scale of the former should be tuned such that the rate of acceptance is 0.45 in the one-dimensional case and 0.23 for dimensions over one. However, for the scope of this thesis, and keeping in mind the very nice shape of $\mathrm{J}(x)$ there is little to indicate that maximum acceptance rate is bad.

1.4.1 Accuracy of Approximation

There are two quantities that are of particular interest when we use this approximation of the NIG-Lévy process. First, how good the approximation is. Second, how computationally intensive the approximation is. If X^ϵ is the truncated process, define the measure of accuracy for $t = 1$ as:

$$D(\epsilon) = \sup_{x \in \mathbb{R}} |\mathbb{P}(X^\epsilon(1) \leq x) - \mathbb{P}(X(1) \leq x)|$$

While the details are in Asmussen and Rosinski[20], it can then be showed that this has an upper bound has follows

$$D(\epsilon) \leq (0.7975)\sigma^{-3}(\epsilon) \int_{|x|<\epsilon} |x|^3 \, \mathrm{Q}(\mathrm{d}x)$$

With this upper bound for the error, the truncation level can be chosen based on maximum acceptable error. Given the shape of the Lévy measure of the NIG, it can be very computationally intensive if we choose a truncation level too small. Or vise versa it will not be very accurate if the Brownian motion is

ϵ	2e-05	2e-04	2e-03
Error	2.8e-03	8.8e-03	2.8e-02
Intensity	31830	3182	317

Table 1.1: Maximum Error Levels of Brownian approximation

left to do most of the jumps. Let $\delta(\epsilon) = D(\epsilon)/(0.7975)$. Then for the NIG case we have [20, Theorem 3.1]:

$$\delta(\epsilon) \sim \sqrt{\frac{\pi}{8\delta} \cdot \epsilon}$$

Using this we can see the maximum error level for some selected values of ϵ, given $\delta = 1$ is as in Table 1.1. One should also notice that the intensity of the jumps for the chosen values are proportional to the value of ϵ, the error level is not. The choice of truncation level also needs to take into consideration the computational effort required.

Clearly from the formula, with high δ we can choose a bigger ϵ and thus lower the computational effort required as the intensity decreases. The errors calculated are the maximum, and hence we can expect the approximation to perform better than this much of the time.

From this we can simply invert the above formula for the error of the Brownian approximation and thus ϵ can automatically be set using the formula:

$$\epsilon = \frac{8\delta^2(\epsilon) \cdot \delta}{\pi} \tag{1.9}$$

For the second truncation there is a slightly different approach. The first thing that should be observed is that f_q is lighter in the tails than the target Lévy measure for some values of α and β. Hence it will at some point cross the graph of the Lévy measure. There is a possibility of undersampling if ω is chosen too large. On the other hand, choosing ω too small will result in inefficient sampling, as the second term in the proposal, f_e has very poor resemblence to the normalized Lévy near the origin. This in turn leads to poor acceptance rates, and slow convergence.

However, the following equation uniquely determines ω, and furthermore this solution for ω maximizes the acceptance rate. It is based on the method of S. Rasmus [18, Theorem 3.2], but differ due to using a simpler proposal function since f_e in this case is symmetric around the origin. This will simplify the solution by quite a bit compared to the one referred.

Proposition 1.4.2. *The value of ω that maximizes the accept probability in the Accept-Reject algorithm solves the equation:*

$$\beta \mathrm{S}(\omega) - \frac{2}{\omega}\mathrm{C}(\omega) - \alpha\frac{\omega}{|\omega|}\frac{\mathrm{K}_0(\alpha|\omega|)}{\mathrm{K}_1(\alpha|\omega|)}\mathrm{C}(\omega) = -(\alpha + |\beta|)\left(\frac{\epsilon}{\omega}\right)^2 \mathrm{C}(\epsilon) \tag{1.10}$$

where

$$\mathrm{S}(\omega) = \frac{\sinh(\beta\omega)\mathrm{K}_1(\alpha\omega)}{\omega}, \quad \mathrm{C}(\omega) = \frac{\cosh(\beta\omega)\mathrm{K}_1(\alpha\omega)}{\omega}$$

Proof. Define the majorizing contant c such that $\pi(x) \leq c\mathrm{q}(x)$. Write c in term of ω as:

$$\begin{aligned} c(\omega) &= \frac{\nu^\epsilon(-\omega)}{f_e(\omega)} + \frac{\nu^\epsilon(-\epsilon)}{f_q(\epsilon)} + \frac{\nu^\epsilon(\epsilon)}{f_q(\epsilon)} + \frac{\nu^\epsilon(\omega)}{f_e(\omega)} \\ &= \frac{\nu^\epsilon(-\omega) + \nu^\epsilon(\omega)}{\alpha + |\beta|} + \frac{\nu^\epsilon(-\epsilon) + \nu^\epsilon(\epsilon)}{f_q(\epsilon)} \end{aligned}$$

The aim is to find $c(\omega)$ such that the sum of fractions is as big as possible with the above restriction. Differenciate w.r.t to ω:

$$c'(\omega) = \frac{\partial_\omega \nu^\epsilon(\omega)}{\alpha + |\beta|} + \frac{\partial_\omega \nu^\epsilon(-\omega)}{\alpha + |\beta|} - (\nu^\epsilon(\epsilon) + \nu^\epsilon(-\epsilon))\frac{\partial_\omega f_q(\epsilon)}{f_q(\epsilon)^2}$$

where

$$\begin{aligned} \partial_\omega \nu^\epsilon(\omega) &= \left[\beta - \frac{2}{\omega} - \alpha\frac{\omega}{|\omega|}\frac{\mathrm{K}_0(\alpha|\omega|)}{\mathrm{K}_1(\alpha|x|)}\right]\nu^\epsilon(\omega) \\ \partial_\omega f_q(\epsilon) &= -\left(\frac{\epsilon}{\omega}\right)^2 f_q(\epsilon)^2 \end{aligned}$$

Exploiting the following relationship between $\mathrm{S}(\omega)$, $\mathrm{C}(\omega)$ and $\nu^\epsilon(\omega)$ [18, A.5, A.6]:

$$\begin{aligned} \nu^\epsilon(\omega) - \nu^\epsilon(-\omega) &= \frac{2\delta\alpha}{\pi K_\epsilon}\frac{\sinh(\beta\omega)\mathrm{K}_1(\alpha\omega}{\omega} \\ \nu^\epsilon(\omega) + \nu^\epsilon(\omega) &= \frac{2\delta\alpha}{\pi K_\epsilon}\frac{\cosh(\beta\omega)\mathrm{K}_1(\alpha\omega}{\omega} \end{aligned}$$

Substitution for the above expressions:

$$\begin{aligned} \frac{4\delta\alpha}{\pi\mathrm{K}_\epsilon}c'(\omega)(\alpha + |\beta|) = &\left[\beta - \frac{2}{\omega} - \alpha\frac{\omega}{|\omega|}\frac{\mathrm{K}_0(\alpha|\omega|)}{\mathrm{K}_1(\alpha|x|)}\right](\mathrm{S}(\omega) + \mathrm{C}(\omega)) \\ &+ \left[\beta + \frac{2}{\omega} + \alpha\frac{\omega}{|\omega|}\frac{\mathrm{K}_0(\alpha|\omega|)}{\mathrm{K}_1(\alpha|x|)}\right](\mathrm{S}(\omega) - \mathrm{C}(\omega)) \\ &+ 2\mathrm{C}(\epsilon)\left(\frac{\epsilon}{\omega}\right)^2(\alpha + |\beta|) \end{aligned}$$

Simplifying the above expression and setting $c'(\omega) = 0$ gives the desired equation.

□

A similar calculation can be done in the inverse linear case, but the equation is somewhat more complex since f_l has a different partial derivative w.r.t. ω: $\partial_\omega f_l \neq -C \cdot f_l^2$. Furhtermore, this choice for ω is valid for both the Metropolis-Hastings as well as for the Accept-Reject algorithm, as the majorizing constant will simply be eliminated in the former due to the fraction. However, the choice for ω may be bigger in the case of $\mathrm{J}^1(x)$ owing to a lower derivative.

As can be noticed, the α and β are the ones that influences the second truncation ω the most of the NIG-parameters. This is expected, as δ and μ have little effect on the tails themselves.

1.4.2 Implementation issues

Some issues did come up during implementation. In particular, when implementing programmatically it is difficult to determine exactly in which interval a solution to Equation 1.10 exists. This is necessary for solving the equation numerically. Some initial tests showed that the solution was approximately inversely proportional to the value of α times a constant. The choice was therefore made to search in the interval between ϵ and $15 \cdot \alpha^{-1}$. For extreme values of α, manual tuning may be necessary, but this is a programmatic concern, and therefore it has not been analysed further.

There were no other implementation issues worth mentioning, but it should be noted that the algorithm is rather heavy computationally for large numbers of jumps. Large quanta of high speed memory are therefore recommended for efficient simulation.

1.4.3 Simulation of the normal inverse Gaussian

The first issue to deal with in Markov Chain Monte Carlo is the number of iterations required for convergence. As suggested by Raftery and Lewis [15], a small analysis on sample variance is performed.

A method of determining how many iterations the chain must take to produce a valid draw is to study the variance of the draws as the number of iterations is increased. This can be a time-consuming job, but it can give an indication on the minimum of iterations required. Results from 3 independent runs are available in Figure 1.4.

In all of them, the algorithm shows very good rates of convergence, and simulation indicates that one can get away with as few as 500–1000 iterations. This can greatly decrease computation time when generating huge number of draws.

The following plot shows a sample simulation of the approximation via Metropolis-Hastings compared with that of a NIG-Lévy process with the same parameters. As can be observed in Figure 1.5, the simulations have similar range and from a first look seems similar.

Table 1.2 shows that a sample run from the approximation yields quite good

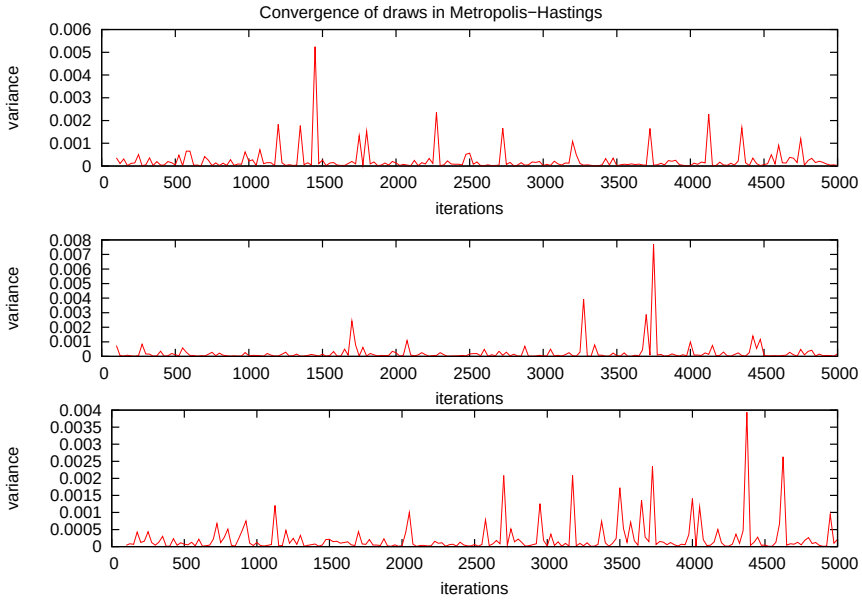

Figure 1.4: Variance from 3 random starting points simulations via Metropolis-Hastings.

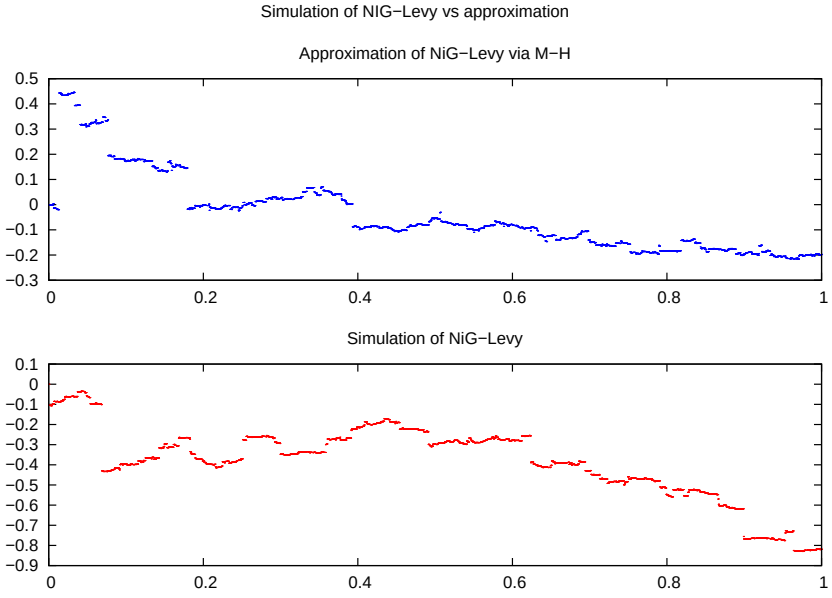

Figure 1.5: Simulation of NIG-Levy and its approximation

	M-H		A-R		NIG-Levy
	x^{-1}	x^{-2}	x^{-1}	x^{-2}	
$\#(J_{>\epsilon})$	2445	2235	2368	2273	2388
Accept %	0.5001	0.9997	0.05818	0.9997	-
mean(J)	-3.99e-02	8.09e-06	6.31e-04	6.74e-05	3.92e-05
var(J)	2.76e-02	6.27e-05	2.09e-04	4.56e-05	1.67e-05
mean$(\lvert J\rvert_{>\epsilon})$	-1.63e-01	2.89e-05	2.65e-03	2.96e-04	2.19e-04
var$(\lvert J\rvert_{>\epsilon})$	9.28e-02	2.8e-04	8.8e-04	2e-04	1.72e-04

Table 1.2: Statistics for approximations and NIG-Lévy simulation

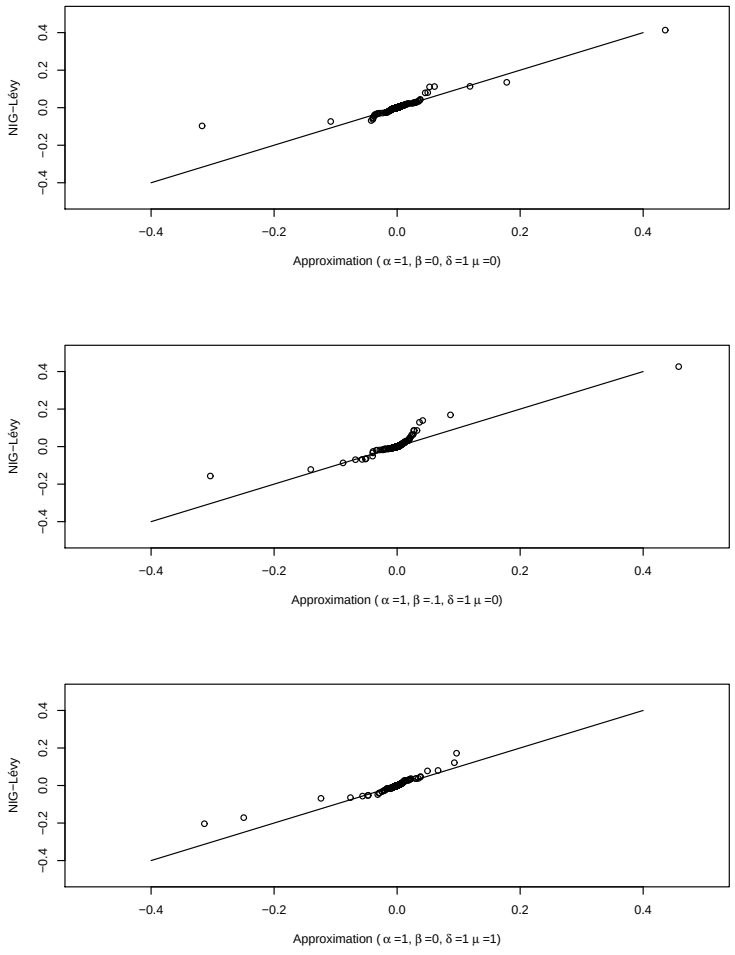

Figure 1.6: Quantile-Quantile plots of approximation and NIG-Lévy

results compared to a simulation directly from NIG-Lévy. However, one should notice that these simulations were done under specific conditions. First, ω was fixed to 1.38, but this is mainly to see how the different proposals perform under similar conditions. Furthermore, for simulations in Table 1.2 the following parameters were chosen for one time step, where each time step has 10000 points in the simulation grid: $\alpha = 1, \beta = 0, \mu = 0, \delta = 1$. The accept rates and other statistics using both proposal functions were compared against each other, and all simulations where done with 1500 iterations in the rejection based algorithms. It can be confirmed that the approximation works well by looking at the quantile-quantile plots for different parameters. From the results in Figure 1.6 one can observe that the approximation does quite well. For some outliers it may merely depend on the specific run, as the heavy tails can cause some results to be off, and there is also a small contribution in the error by the Brownian approximation of the small jumps.

Both methods performed very well with the f_q as part of the proposal, yielding over 99% acceptance rate on average. While this in general situations could be a reason for concern, it is natural in this setting as the spread between the proposal and the target density is very small. Using f_l as a part of the proposal on the other hand did not produce very convincing results. As can be expected from Figure 1.2, using this proposal would lead to very slow convergence, and there are some signs of this in the results when comparing to the results produced by the NIG-Lévy process and the theoretical values.

All in all, the approximation seem to work very well, and there is some indications that Metropolis-Hastings with a proposal including f_q will produce very satisfying results with fast convergence. This is therefore the preferred method of choice in the chapters that follow. Although it will be stated explicitly which parameters are used, we can already here say that one can run Metropolis-Hastings for less than 500 iterations and still get good approximations.

Chapter 2

Dependence between Lévy Processes

The study of dependence between stochastic processes has a long history in statistics, from the use of simple bivariate distributions which share some common parameters, to non-linear measures of dependence and measures of concordance.

This chapter extends the previous chapter by introducing dependence between the jumps in NIG-Lévy approximations. Restricting the model to two dimensions, the effect of dependence between the small jumps and dependence between large jumps is studied.

2.1 Conventional measures of dependence

The most known and commonly used tool employed in modelling dependence is the correlation, relating to Pearson's product-moment correlation coefficient, ρ. This is defined as

$$\text{corr}(X, Y) = \frac{\text{covar}(X, Y)}{\sqrt{\text{var}(X)\text{var}(Y)}}$$

In this thesis, the term correlation is reserved for this particular linear dependence measure. In general the term dependence will be used, while some measures are referred to as measures of concordance. The latter do not in the same way measure the dependence, but focus on the probability of joint movements up or down. Two famous examples are Kendall's τ and Spearman's ρ_s. In this thesis, however, the connection between these and the dependency structure, copulas, is not in focus. More information on this connection can be found in Frees and Valdez [10, Table 3], or for an introduction, see Nelsen [13, 5.1.1,5.1.2].

2.2 Copulas

Copulas are not a new concept; they were first introduced by Sklar (1959). However, there has been significant progress in the field, especially from a practitioner's point of view. With the ultimate goal to make as good a model as possible fitted to market data, the concept of dependence is even more central than before, especially when looking at structured products (e.g. derivatives) that cover more than one underlying.

Definition 2.2.1 (Copula)**.** A 2-dimensional copula is a function $C : [0,1]^2 \to [0,1]$ such that:

- C is grounded, i.e. $C(x,y) = 0$ if $x \cdot y = 0$
- C is 2-increasing
- $\forall (x,y) \in \mathrm{dom}(C)$ we have $C(u,1) = u$ and $C(1,v) = v$

Furthermore, if the copula satisfies certain smoothness conditions, the copula density is given as:

$$C(U = u, V = v) = \frac{\partial C(u,v)}{\partial u \partial v}$$

The existence of the copula density can in some cases become vital for simulation. To understand the flexibility and power that can be gained by using copulas for dependence structure, Sklar's famous result must be introduced.

2.2.1 Sklar's Theorem

Theorem 2.2.2 (Sklar's Theorem)**.** *Let C be a copula, and F and G margins. Let H be a multivariate density. Then*

$$H(x,y) = C(F(x), G(y)) \tag{2.1}$$

If F and G are continuous, then C is unique. Conversely, given a multivariate distribution H and margins F and G, there exists a copula such that (2.1) holds. Again, if F and G are continuous, then C is unique.

The result is a very elegant, and shows that copulas can be used to model the dependence independently from the margins. This gives a whole different kind of flexibility compared to the linear correlation. For example, one can keep the dependence structure constant, while changing all or a few of the margins of the underlyings.

Furthermore, it follows that there is an equally interesting relationship that will be very useful in studying how the overall copula changes when the processes are a mixture of several multivariate distributions each with their own dependence structure.

$$C(u,v) = H(F^{-1}(u), G^{-1}(v))$$

2.2.2 Archimedean copulas

This particular class of copulas is worth special attention. They have a structure that make them easy to work with and extend over several dimensions. In common they have that they can be written in terms of a generator function.

Definition 2.2.3 (Generator function of a copula)**.** The (additive) generator function of a copula, $\phi(x)$, is a strictly decreasing function such that:

$$C(u_1, \ldots, u_n) = \phi^{-1}(\sum_{i=1}^{n} \phi(u_i))$$

All copulas of the Archimedean class have the property that they can be written in terms of their generator function and its inverse [13, theorem 4.1.4]. This will be a very useful fact in simulation with copulas. It should also be noted that a copula of Archimedean class is symmetric with repect to its arguments.

In this thesis the copulas Clayton, Frank and Gumbel are studied, and sample pairs generated from these copulas are presented in Figure 2.2.

Clayton copula

Definition 2.2.4 (Clayton family)**.** A one parameter family of copulas given as:

$$C(u, v; \theta) = \left(u^{-\theta} + v^{-\theta} - 1\right)^{-1/\theta}, \theta > -1$$

The copula is asymmetric with higher dependence in the lower tail than in the upper. The parameter of this family controls the dependence in the lower tail, and the higher the θ is, the higher the dependence. For the limiting case of $\theta \to 0$ the copula is the indepence copula.

An example with different parameters is given in Figure 2.1.

Frank copula

Definition 2.2.5 (Frank family)**.** A symmetric one-parameter copula given as:

$$C(u, v; \theta) = -\frac{1}{\theta} \log \left[1 + \frac{(e^{-\theta u} - 1)(e^{-\theta v} - 1)}{e^{-\theta} - 1}\right]$$

The copula is symmetric around the point (1/2, 1/2), and hence any change in the parameter θ will influence dependence in the tails equally.

Gumbel copula

Definition 2.2.6 (Gumbel family)**.** An asymmetric one-parameter copula given as:

$$C(u, v; \theta) = \exp\left(-\left[(-\log u)^{\theta} + (-\log v)^{\theta}\right]^{1/\theta}\right)$$

The Gumbel has more dependence in the upper tail than in the lower, but it does not have the same spread as the Clayton. This makes it very suitable for problems where the dependence relationship is asymmetric, but still has high dependence in both tails.

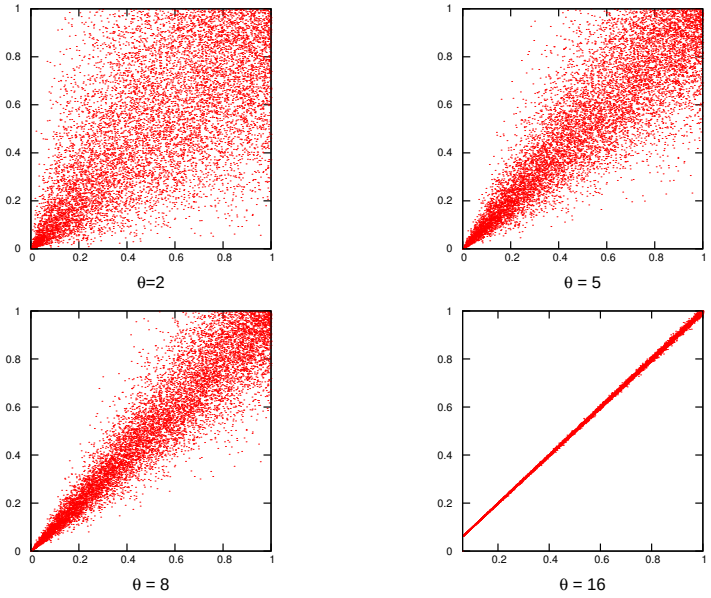

Figure 2.1: Simulation of the Clayton copula with different θ

2.2.3 Simulation via copulas

The presented methods are more concisely described by other authors, for example Frees and Valdez [10]. The two methods which will be employed in the simulations are the following:

First, the conditional law of a copula is given by the following expression as long as the copula is smooth enough to be differentiable:

$$\frac{\mathrm{d}C(u,v)}{\mathrm{d}u} = P(V \leq v | U = u)$$

This conditional law is one of the very basic tools that can be employed in order to simulate a pair of dependent processes. A basic algorithm will look like the following:

Algorithm 2.2.7 (Simulation of dependent pair using conditional law)**.**

1: Let u a be value of the first process' CDF.
2: Draw V_1 from $\mathrm{U}(0,1)$
3: Define v as:

$$v = \frac{\mathrm{d}C(u,v)}{\mathrm{d}u}^{-1}\left(V_1\right)$$

return (u, v)

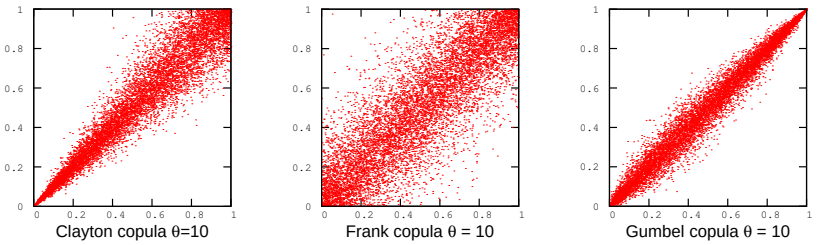

Figure 2.2: Simulation of Clayton, Frank and Gumbel copula

A second approach to simulating from a copula is through its generator. This method generates a pair of dependent variables, but it can be modified to generate a second dependent variable given the first. Exploiting the relationship between the inverse of the generator and Laplace transform for common distributions gives rise to the following algorithm first mentioned in Marshall and Olkin (1988), and employed by Frees and Valdez [10] among others:

Algorithm 2.2.8 (Simulation through inverse generators)**.**

1: Simulate a variate X where the Laplace transform of the distribution function is equal to the inverse of the generator, ϕ^{-1}.
2: Simulate two independent variate $V_1, V_2 \sim \mathrm{U}(0,1)$.
3: Define:

$$u = \phi^{-1}(-\log(V_1)/X), \quad v = \phi^{-1}(-\log(V_1)/X)$$

return (u,v)

Clayton copula

The Clayton copula has a simple structure and is easy to simulate through Algorithm 2.2.7 since it is easy to differentiate and invert:

$$\frac{\mathrm{d}C(u,v)}{\mathrm{d}u} = -\theta^{-1}(u^{-\theta} + v^{-\theta} - 1)^{-(1/\theta)-1}(-\theta)u^{-\theta-1}$$

Inverting around $V_1 \sim \mathrm{U}(0,1)$ through simple algebra, the resulting conditional density is:

$$v = (u^{\theta} y^{-\theta/(\theta+1)} + u^{\theta} - 1)^{1/\theta}$$

It should be noted that since Clayton is an Archimedean copula, it could equally well be simulated via Algorithm 2.2.8. On the other hand, there is no reason to apply a more sophisticated method when it can easily be differentiated and inverted.

Gumbel copula

The Gumbel copula on the other hand can be simulated from using the second method. Let the single parameter of the Gumbel be θ. The inverse of its generator function is the Laplace transform of positive stable variate [10, Table 2] with the following parameters:

$$\alpha = 1/\theta,\ \beta = 1,\ \gamma = (\cos(\pi/(2\theta)))^{\theta},\ \delta = 0$$

Hence, if $y \sim \mathrm{St}(\alpha, \beta, \gamma, \delta)$, and $u \sim \mathrm{U}(0, 1)$ then calculate:

$$v = \exp\left(-\frac{\log(V_2)}{X}\right)$$

The pair (u, v) is then a realization from the Gumbel copula. However, in applications where one dependent draw is generated from another draw, one already have one realization from the jumps. Therefore, one can choose which variable one is interested in simulating, and which one that should be calculated based on the first draw. Choosing V_1 to be the drawn variable in Algorithm 2.2.8, X can be calculated as:

$$X = \frac{-\log(V_1)}{(-\log(u))^{\theta}}$$

A dependent v is found by substitution in the expression given above.

Frank copula

Similarly to the Gumbel, the Frank copula can be simulated using the second method. Here the inverse of the generator function is the Laplace transform of a logarithmic series distribution on the positive integers [10, Table 2]. However, the distribution is not implemented in software directly, nor did Frees and Valdez give the required argument.

Fortunately, using the same methods as with the Gumbel copula one can avoid the simulating from the logarithmic series distribution, and simply draw a random standard uniform number. If u is the first random number drawn, and V_1 is as in Algorithm 2.2.8, then the following gives the second dependent realization:

$$v = -\theta^{-1} \log(1 + \exp(\log(V_2) X^{-1})(e^{-\theta} - 1))$$

where

$$X = \frac{\log(V_1)}{\log(e^{-\theta u} - 1) - \log(e^{-\theta} - 1)}, \qquad V_1, V_2 \sim \mathrm{U}(0,1)$$

2.3 Lévy-copulas

While the concept of conventional copulas is well-explored and documented, they do not guarantee that structure is preserved. In other words, it is not clear in general if the resulting measure of a Lévy process passed through a copula will produce another Lévy process. However, there are examples of this. Most known is the Gaussian copula joining two Brownian motions, which produces a 2-dimensional Brownian motion. However, this is not the case for any other copula joining two Brownian motions. This issue calls for somewhat similar, but different concept.

Tankov defines an equivalent concept for Lévy processes named not surprisingly Lévy copulas [9, Def. 5.12]. They share many properties with conventional copulas.

Definition 2.3.1 (Lévy copula)**.** A function $F(x, y) : [-\infty, \infty]^2 \to [-\infty, \infty]$ is a 2-dimensional Lévy copula if:

- F is 2-increasing
- $F(0, x) = F(x, 0) = 0$, $\forall x$
- $F(x, \infty) - F(x, -\infty) = F(\infty, x) - F(-\infty, x) = x$

As can be seen, they do not share the domain or range with conventional copulas since they act on the tail integral of the Lévy measure and not the CDF of the probability measure. Since the Lévy measure contains all the information of the stochastic jump behavior of the Lévy process, one is guaranteed that a Lévy process is the result when joining two Lévy processes with a Lévy copula.

2.3.1 Sklar's Theorem for Lévy copulas

From Cont and Tankov [9, theorem 5.6] there exist a similar result to Sklar's Theorem for Lévy copulas:

Theorem 2.3.2 (Sklar's Theorem for Lévy copulas)**.** *Let U be an d-dimensional tail integral for a d-dimensional Lévy process with positive jumps. Furthermore, $U_1, \ldots, U_d$ be tail integrals for one-dimensional Lévy processes. Then there exist a Lévy copula F such that*

$$U(x_1, \ldots, x_d) = F(U_1(x_1), \ldots, U_d(x_d))$$

If $U_1, \ldots, U_d$ are continuous then F is unique, otherwise it is unique on $\operatorname{Ran} U_1 \times \operatorname{Ran} U_d$.

Conversely, if F is a d-dimensional Lévy copula and $U_1, \ldots, U_d$ are tail integrals with Lévy measures on $[0, \infty)$, then the function U above is the tail integral of a d-dimensional Lévy process with positive jumps having marginal tail integrals $U_1, \ldots, U_d$.

Notice that the theorem is specified for Lévy processes with positive jumps. The general case can be achieved by treating each quadrant or orthant seperately.

2.3.2 Some examples

Example 2.3.3. Independence Lévy copula

$$F_{\perp}(x_1, \ldots, x_d) := \sum_{i=1} x_i \prod_{j \neq i} 1_{\{x_j = \infty\}}$$

Example 2.3.4. Complete Dependence Lévy copula

$$F_{\parallel}(x_1, \ldots, x_d) := \min(|x_1|, \ldots, |x_d|) 1_K(x_1, \ldots, x_d) \prod_{i=1}^{d} \operatorname{sgn}(x_i)$$
$$K := \{x \in \mathbb{R}^d : \operatorname{sgn}(x_1) = \cdots = \operatorname{sgn}(x_d)\}$$

Example 2.3.5. Clayton-like Lévy copula (2-dim)

$$F(u, v) := (|u|^{-\vartheta} + |v|^{-\vartheta})^{-1/\vartheta} (\eta 1_{\{uv \geq 0\}} - (1 - \eta) 1_{\{uv < 0\}})$$

2.4 Bivariate distributions with NIG marginals

The copulas provide high flexibility in modelling and here several samples from bivariate distributions are generated. The marginals are the processes approximating the NIG-Lévy developed in Chapter 1. Some of the copulas introduced will be used, and the results analysed. The copulas mentioned will be applied to jumps larger than ϵ. Only correlation (Gaussian copula) will be employed on the Brownian part of each process to ensure that the resulting 2-dimensional process is indeed a Lévy process. However, in view of the dependence concepts introduced earlier, three possible approaches are discussed: one using Lévy copulas, one using conventional copulas and one hybrid concept. After analysing strengths and weaknesses, a method will be chosen.

Direct Lévy-copula method

The simplest method for simulation purposes is without doubt simply applying a Lévy copula directly to our two processes. The NIG-distributions that they follow will of course have the possibility of using different parameters, and a model based on this will still retain the parameters, making it easier to calibrate these to market data.

Since the NIG-Lévy process is a pure jump process with infinite activity, its Lévy measure will be infinite at the origin. However, this produces a 2-dimensional

tail integral and not a CDF as with conventional copulas. Working with a tail integral is in the simulation case not very different from working on the result of a conventional copula, however there is a catch. Since the Lévy copulas are defined on $[-\infty, \infty]^2$, one actually need a total of four tail integrals given as $U^{++}, U^{--}, U^{+-}, U^{-+}$ where the superscript indicate a negative or positive increment for for each of the two processes in two dimensions. [9, theorem 5.7]. This does give flexibility as one can define a different Lévy copula on each set, but it can easily become challenging to implement.

Passing that hinderance, some conditions on the smoothness must be introduced for the Lévy-copula and the margins. The Lévy density is then found by differentiation [9, p. 148]:

$$\nu(x_1, x_2) = \left.\frac{\mathrm{d}^2 F(y_1, y_2)}{\mathrm{d}y_1\,\mathrm{d}y_2}\right|_{y_1=U_1(x_1), y_2=U_2(x_2)} \nu_1(x_1)\nu_2(x_2) \tag{2.2}$$

Inverting the law numerically around a uniform variable, simulation can in some cases be straight forward. However, since we are left with a tail integral with an atom in the origin, problems can occur when we want to price a basket of two NIG-Lévy processes joined with a Lévy copula. The tail integral is formally infinite in the origin, and from that respect there may be problems evaluating the expectation near the origin directly. Another argument against the use of Lévy copulas are their limited number of structures. There are more than over 20 known Archimedean conventional copulas [13, Tabel 4.1], while there is a lot fewer established Lévy copulas.

This leads to the idea of a potential hybrid solution where the Lévy measure is truncated and finite.

Combined copula and Lévy-copula approach

Using approximation of the Lévy process we can also try a joint approach: A two-dimensional Brownian motion to the small jumps after the Lévy measures have been joined with a Lévy copula.

While one could argue that this falls between two chairs, it gives the modeller a chance to employ the power of Lévy copulas while shying away from the issues discussed above. It should be possible to simulate dependent pairs from this, as long as the smoothness condition mentioned in the previous section holds. For option pricing one can avoid values that are infinite, since the two-dimnesional Lévy measure is truncated, but one must still find a Lévy copula for each quadrant.

This all builds on the assumption that the normal approximation is valid, in two dimensions, and especially that it is unique. Asmussen and Rosinski only dealt with the one-dimensional case, and to this author it is unclear how one would choose, e.g., the correlation parameter once two Lévy processes are joined with a Lévy copula. Future research may prove this to be a viable solution, but at this time it is too uncertain to use this in the thesis.

Conventional copula approach

Using an approximation we can simply use a copula on both the Brownian motions and the compound Poisson. This approach requires many of different quantities for the price processes $L_k^{(n)}$:

- The distributions $\gamma_i^{(k)}$ for all k processes
- A copula for each pair of γ_i^1, γ_i^2 (2-dimensional case)
- Intensity for $N(t), M(t)$ and copulas for the Brownian motions

The gain is that one is now in the realms of conventional copulas as opposed to Lévy copulas. The power of the Sklar's theorem for conventional copulas can now be unleashed to produce multivariate distributions for each jump. This takes it a step further by not simply employing a copula on the Brownian parts letting the large jumps be completely independent. Although this is a common method among the practitioners, there is empirical evidence of common shifts within similar market segments and the market as a whole.

This model will quickly become complicated, and empirically it is difficult to even observe, e.g., dependence in jump intensity. It is also a challenge as to how one should decide the final law of the multivariate process if the intensities are anything else than completely dependent or independent. Therefore, in this thesis, only the latter two are assumed. In addition, there is no dependence between the jumps in the same process, i.e., no stochastic volatility or volatility clustering as this would possibly break the attribute of stationary increments.

2.5 The model

The underlying model from Chapter 1 is unchanged.:

Definition 2.5.1 (Our model)**.**

$$L_t^{(1)} = \mu_\epsilon^{(1)} t + \sigma_1(\epsilon) W_t + \sum_{i=0}^{N(t)} \gamma_i^{(1)} \tag{2.3}$$

$$L_t^{(1)} = \mu_\epsilon^{(2)} t + \sigma_2(\epsilon) W_t + \sum_{i=0}^{N(t)} \gamma_i^{(2)} \tag{2.4}$$

$$\tag{2.5}$$

where

$$\begin{aligned} \mu_\epsilon, \sigma(\epsilon) &\text{ as in (1.6)} \\ N(t) &\sim \text{Poisson}(\lambda) \\ \gamma_t^{(1)}(u) &\sim \nu^\epsilon(u) K_\epsilon^{-1} \\ \gamma_t^{(2)}(v) &\sim \text{G}_2^{-1}(C(V = v | u = \text{G}_1(\gamma_t^{(1)}))) \end{aligned}$$

Where G_1 is the density function of $\nu^{\epsilon}(\alpha_1, \beta_1, \delta_1)K_{\epsilon}^{-1}$ and G_2^{-1} is its inverse with parameters $(\alpha_2, \beta_2, \delta_2)$

The model itself is rather flexible with regard to calibration of either parameters. However, it should be noted that the model does not allow mass on the axis of the joint Lévy measure, i.e. all jumps occur at the same time. The choice of this implementation is based on that there is not a clear way to choose how much of the intensity will be independent jumps, unless one were to say that jumps of a certain size are independent. Similarily, dependence between jump intensities is not introduced, as it is very difficult to determine this type of relationships empirically. Therefore, a simplified model such as the one above is chosen.

2.6 Simulation of dependent NIG-Lévy approximations

Simulation is done by first simulating a path for the approximation. For each jump in the compound Poisson, a dependent jump is generated from the conditional law of the copula. The second process is then constructed as in Equation (2.5). In Figures 2.3, 2.4 and 2.5 the former process is marked as "NIG-Lévy approx", while latter process is marked as "Dependent approx".

see the effect of copulas on high dependency. Clearly, with such high tail dependency in the lower (upper) tail there will also be reasonable high dependence in the upper (lower) tail as well using the Clayton copula.

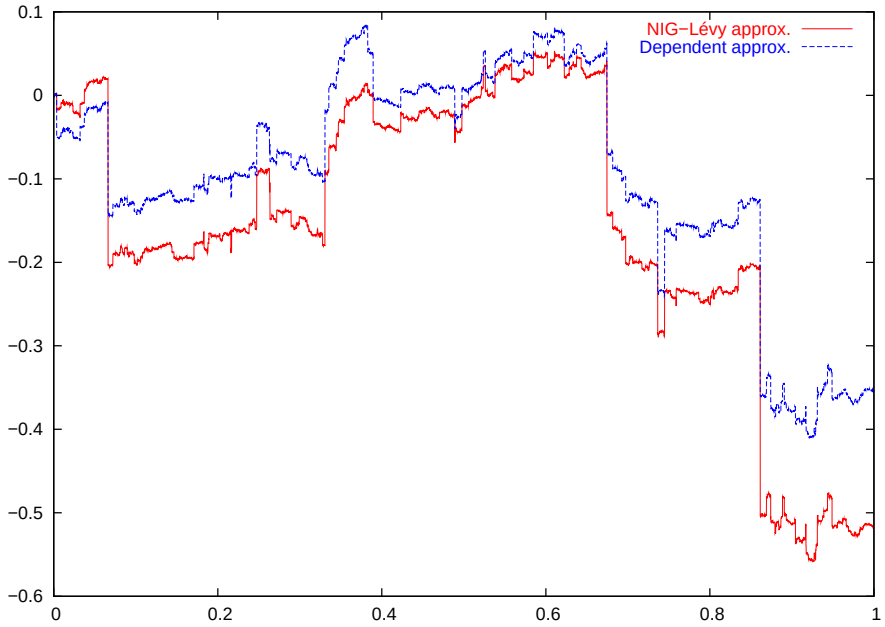

Figure 2.3: Gumbel with $\theta = 20$, notice higher dependency in upper than lower tail.

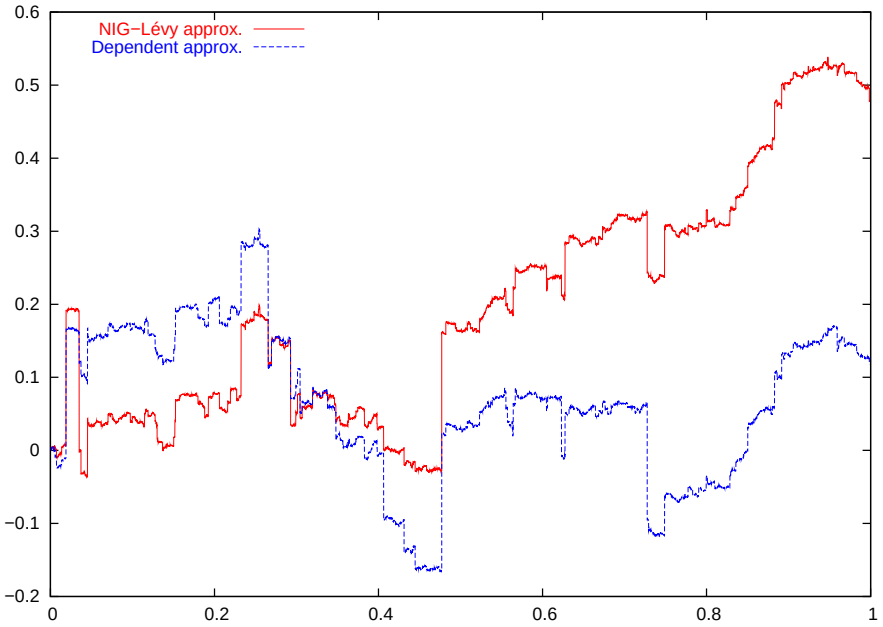

Figure 2.4: Gumbel with $\theta = 10$, notice higher dependency in upper than lower tail.

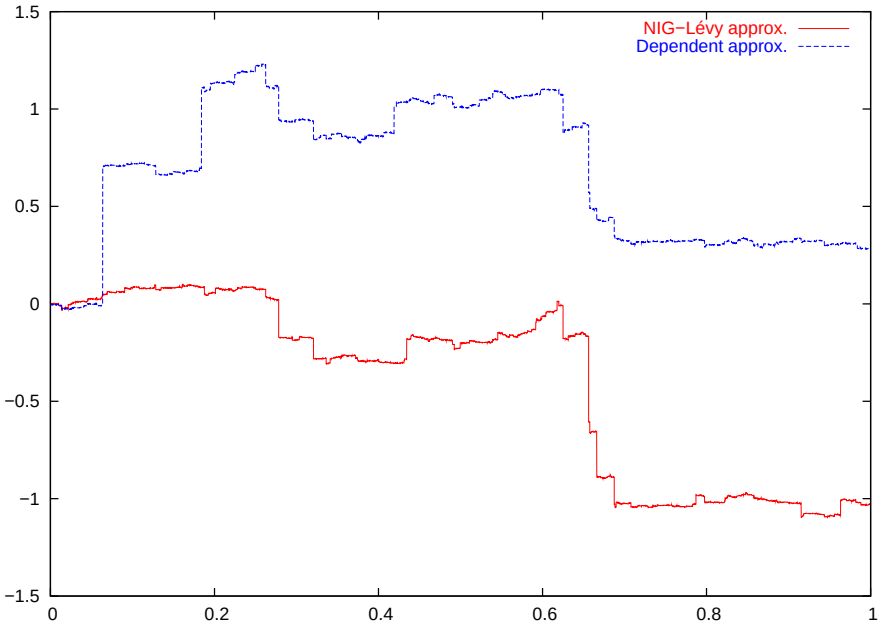

Figure 2.5: Clayton simulation with $\theta = 2$, higher dependency in lower tail.

2.6.1 Implementation issues

There are some issues that should be assessed. First, using the same truncation on each of our dependent processes may not result in both marginal processes approximating the NIG-Lévy equally well. This stems from the direct effect that different values of δ have on the error of the Brownian approximation. Furthermore, choice has to be taken with regards to dependency in the large jumps. Since the dependence in the general case is not perfect, the copula may produce jumps that would lie inside $(-\epsilon, \epsilon)$, if one use the quantiles of the NIG-Lévy increments. Since the jumps are drawn from the truncated Lévy jump measure,ν^ϵ, a jump in this inteval can be translated to either a negative jump or a positive jump if the first jump is close to ϵ or $-\epsilon$. This can change the intensities somewhat, but not in a degree that will let the process be vastly different from a NIG-Lévy. This may not the behavior one expect the when introducing dependency in the jump sizes, but this is a direct consequence of the approximation. Approximation error of the dependent process may be slightly higher than the first due to the reasons mentioned above.

One alternative is to round this jump to the minimum size ϵ. This on the other hand breaks the dependency structure, while maintaining the Lévy measure of the processes. A second alternative is to fit e.g. a NIG-distribution to the random variates drawn from a jump measure and use the builtin quantile mechanisms. However, this will as mentioned allow for values which are less than ϵ. In either case, these implementation issues must be conquered on the cost of flexibility. Working with ν^ϵ directly may give unexpected effects, but it does not break the dependency structure.

Another issue that should be noted is that of round-off errors in the algorithm. This can give very strange results, especially with the Frank copula, where one can have large variations due to round-off errors in the tails as the tails of the Lévy jump probability measure are rather light.[1] Therefore, it is necessary to implement a sanity check since the algorithm seems sometimes to experience numerical instability. In certain cases, the quantile produced by the copula has been rounded to 1 or 0 for extreme events in the tails. This gives jumps that seem unreasonable. The numerical implementation is therefore critical. Small variation in the quantile numbers may in the tail produce large differences in actual jump size. This is often observed far out in the tail where the jump sizes are often determined in 4th or 5th significant digit.

2.6.2 Copula choice for financial applications

In simulation, large spread in some parts of the copula can give unexpected results. For example, for the Clayton with $\theta = 2$ Figure 2.1 shows that it has low dependence in the upper tail, while still quite a bit in the lower tail. Because of the high spread in the upper tail, one can sometimes observe that the dependent processes that are generated experience some negative drift. This is due to the way ν^ϵ is defined, as with sufficient spread, a jump close to ϵ may generate dependent jump of size $-\epsilon$ or less, or visa versa. One can have large

[1] An initial jump of 0.38 corresponds to 0.9998 quantile ,while the returned result from the copula may be as low as 0.9991, which gives a dependent jump of 0.15.

overreactions in the dependent process such as in Figure 2.5. This is sometimes so strong that the small jumps of the process are unable to make it recover. Due to the relationship in the upper tail, big jumps generated by a process are not necessarily implying big positive jumps in the dependent process. However, as θ is increased, this effect will fade. This leads to the question of whether, e.g., Clayton is an optimal copula in financial applications in low dependence settings.

The other copulas presented do not have such a large asymmetry, and it is therefore possible that these are better choices in many cases. In particular Gumbel has higher dependency in both tails.

2.6.3 The empirical copula

This chapter aims not only to implement copula-based dependency structures, but also to study how much dependency between the larger jumps affect the overall dependency structure of the NIG-Lévy process.

The dependency via approximation of two NIG-Lévy processes can be viewed as a mixture of copulas, since the Browian parts have their dependency specified via a Gaussian copula (correlation matrix), while the jumps are dependent through a copula such as the Clayton or Gumbel copula.

From Chapter 1 it is argued that the behavior of the process is dominated by the behavior of the small jumps, and it is therefore natural to assume that the overall dependency structure is not affect to a great degree. From Nelson [13, 5.6.1] we have the definition:

Definition 2.6.1 (Empirical Copula)**.** The empirical copula is the dependency structure extracted a sample of points from two processes. Let $S = \{(x_k, y_k)\}_{k=1}^n$ denote a sample of size n from a bivariate distribution. The empirical copula is the function defined by:

$$C_n\left(\frac{i}{n}, \frac{j}{n}\right) = \frac{\#(x,y) \in S : (x,y) \leq (x_{(i)}, y_{(j)})}{n}$$

where $x_{(i)}, y_{(j)}$ are the order statistics from the sample.

Using simulations with the methods in Chapter 1 with the Clayton copula with $\theta = 2, 8, 16$ and $\rho = 0, 0.5, 0.9, 1$, the results are found in Figures 2.6, 2.7 and 2.8 for $\theta = 2, 8, 16$ respectively.

However, the reader should keep in mind that the shape of the empirical copula will be very dependent on the error level chosen, as this directly influences the intensity of the jumps above ϵ. Also, it is important to have a very high number of data points for an efficient study of the empirical copula since the NIG has very high mass in the small jumps. For this reason, only plots for one copula is included in this thesis, since on the current ϵ-level chosen by the algorithm, there will be only minor differences to the plots of different copulas.

It is rather difficult to observe much difference with different values of θ. The reason is not surprising, there is simply too much noise from the small jumps,

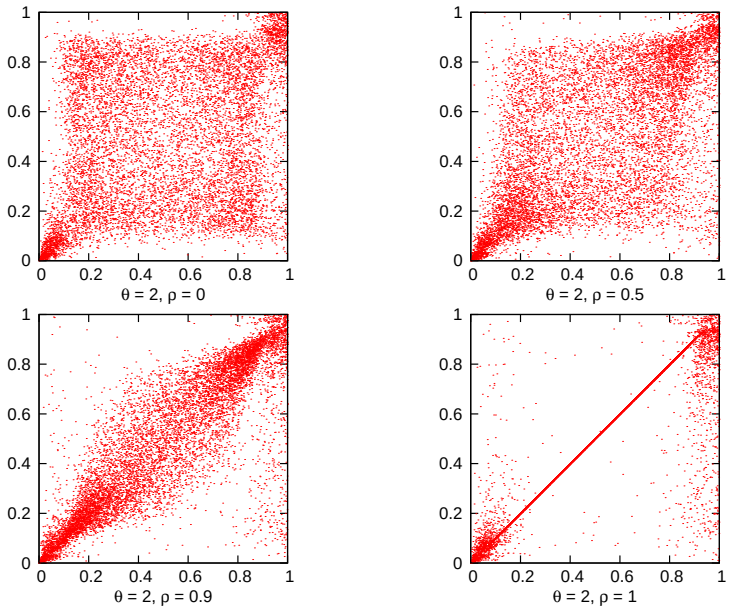

Figure 2.6: Empirical copulas from simulations with Clayton $\theta = 2$

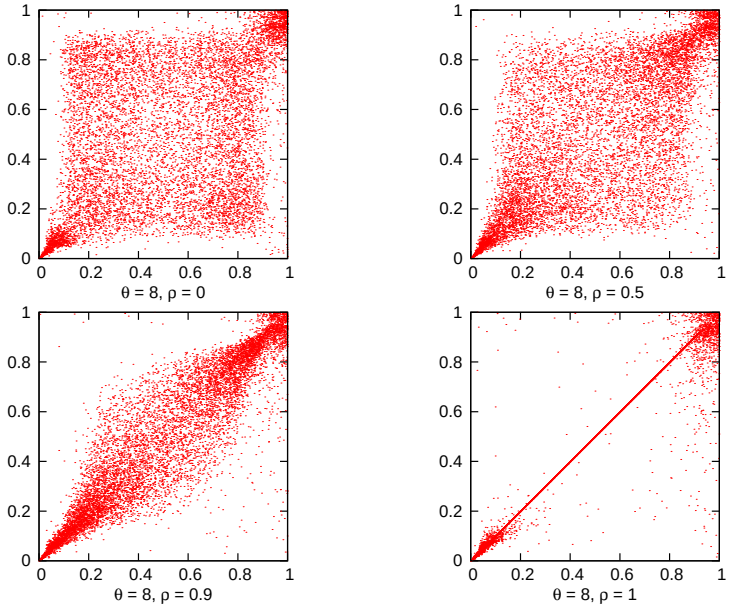

Figure 2.7: Empirical copulas from simulations with Clayton $\theta = 8$

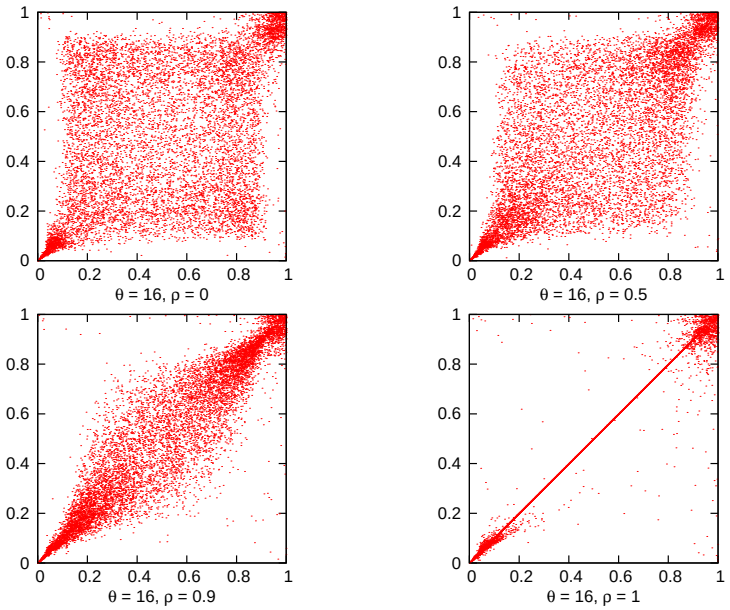

Figure 2.8: Empirical copulas from simulations with Clayton $\theta = 16$

which have substantial mass near the origin However, as ϵ becomes smaller one can expect that the empirical will look more and more like the copula used to join jumps greater than ϵ.

For the big jumps, some change can be observed as θ is increased. In particular, note that the shape towards the lower tail is more pointed if θ is increased.

One possible reason for this shape is that the Clayton copula allows for considerably less dependency in the upper tail than in the lower, and visa versa, if rotated. Hence, for many jumps they are free to lie between very generous upper and lower bounds of the copula when θ is low. This can create unexpected effects in simulations and option pricing. However, since there are very few really large jumps to analyse from in these cases, this might indicate that the big jumps do not occur frequently enough to explain the need for dependency in large jumps.

In conclusion, one can say that a slight misuse of copulas to join distributions with light tails can have disasterous effects on simulation. As demonstrated in Figure 2.5, the process had little chance to recover even with independent Brownian parts, however the dependency pattern is clearly visible elsewhere in the path. This is an indication that one can expect, in low dependency settings, that the large jumps of the dependent processes are more volatile, owing to the structure of the copulas. As mentioned, the copulas can alter the itensity of the jumps, so there is a possibility that the dependent process is not exactly NIG-distributed, but in most cases it will be very close. In addition, the rounding errors on numbers of more than 6 significant digits can change the size of the dependent jumps. Hence, big accuracy is required for good simulations.

Chapter 3

Option pricing with Dependent Lévy processes

Option pricing in the Lévy process framework is very much centered through the existence of an equivalent martingale measure for arbitrage free pricing. In this chaper a possible method of obtaining this for two dependent exponential NIG-Lévy processes is sketched. Furthermore, option prices for a two-asset basket with dependent underlyings are obtained and contrasted with baskets on independent underlyings.

3.1 Asset price model

It is assumed that the log-returns are NIG-Lévy distributed. This has the advantage that asset prices cannot be negative. The model is defined as:

$$\begin{cases} S_0(t) = S_0(0)e^{rt}, & r > 0(\text{constant}) \\ S_0(0) = 1 \end{cases}$$

$$\begin{cases} S_n(t) = S_n(0)e^{L_n(t)} \\ S_n(0) = s_n, & s_n > 0, n = 1, 2 \end{cases}$$

where r is the interest rate, and $L_n(t)$ is a Lévy process for all $t > 0$. The exponential Lévy model is arbitrage free as long as it is not almost surely increasing or almost surely decreasing. In addition it guarantees that we can find an equivalent martingale measure [9, Prop. 9.9].

3.2 Risk Neutral Measures

Owing to the fact that the NIG-Lévy process is a pure jump process as indicated in Chapter 1, this does in general give rise to an incomplete market [7, Thm

3.4]. Therefore there is a whole continuum of risk neutral measures. Depending on the choice of risk neutral measure, the option price can be any value in the range between the lower hedging price (buyer's price) and the upper hedging price (seller's price), as long as it satisfies the no-arbitrage requirement.

Finding a risk-neutral measure or equivalent martingale measure is imperative to option pricing as it allows the modeller to transform the underlying probability measure of the process to an equivalent measure under which the process is a martingale. In the classic Black-Scholes framework an equivalent martingale measure was found using the Girsanov theorem. Lacking a diffusion part in the Lévy triplet for the NIG-Lévy process imposes certain restrictions, in particular one cannot alter the drift freely to create a martingale [9, Prop. 9.8].

Choosing a class of risk-neutral measures which are easy to work with is prioritized. Two alternatives string to mind: Merton's approach and the Esscher transform. Merton's approach assumes that there is no risk-premium on the jumps and that jumps in assets are independent of jumps in other assets. Therefore, it concerns itself with altering the drift so that the process becomes a martingale. However, in the setting of this thesis and in the markets in general, assets do not have independent jumps. This can be observed in downward markets where whole indexes can experience negative shocks[9, 10.1]. The next class of equivalent martingale measures generated by the Esscher transform, which should in particular be noted for its structure preserving property. That is, if the process is a Lévy process under the objective measure P, it will also be a Lévy process under the risk-neutral measure Q. This is a good property to have when implementing, simulating, and pricing options under Q in markets driven by Lévy processes. It does not suffer from the same prerequisites as Merton's approach. In the following, the Esscher transform for independent processes is treated for the NIG-case, and not the approximation. This is because the approximation is only necessary when one want to introduce dependency between the jump sizes. The Esscher transform for the approximation is explored in Section 3.2.3.

3.2.1 The Esscher transform

The Esscher transform was introduced by the Swedish actuary Fredrik Esscher, and has quickly become a very popular way of finding an equivalent martingale measure. It's use in finance was, by name, first introduced by Gerber and Shui[11] in 1994.

The idea is simply to multiply the underlying probability measure by an exponential function of the variable scaled by a constant, h. Then the measure is normalized to form a probability measure:

$$\mathrm{dQ}^h = \frac{e^{hx}}{\int_{-\infty}^{\infty} e^{hx}\,\mathrm{dP}}\,\mathrm{dP} = \frac{e^{hx}}{\mathrm{mgf}_t(h)}\,\mathrm{dP}$$

Where $\mathrm{mgf}_t(h)$ is the moment generating function of $L_1(t)$. Since the moment generating function of the NIG was established in 1.1.2, this is rather easy to derive by algebra:

$$\mathrm{dQ}^h = \mathrm{nig}(x; \alpha, \beta + h, \delta t, \mu t)\,\mathrm{mgf}_t(h)^{-1}$$

Since the exponential function is non-negative and smooth results from basic measure theory give that the measures are absolutely continuous with respect to each other. Hence, the measures P and Q are equivalent probability measures.

The last requirement is that the process is a martingale under the modified measure. To show this, it is sufficient that $\mathrm{E_Q}[e^{L_1(t)}] = 1$:

$$\begin{aligned}\mathrm{E}_{Q^h}[e^{L_t}] &= \mathrm{E}[e^{L_1(t)} e^{hL_1(t)}\,\mathrm{mgf}_t^{-1}(h)] \\ &= \mathrm{E}[e^{(h+1)L_1(t)}]\,\mathrm{mgf}_t^{-1}(h) \\ &= \frac{\mathrm{mgf}_t(h+1)}{\mathrm{mgf}_t(h)} \end{aligned} \tag{3.1}$$

Clearly, for it to be a martingale the parameter h must satisfy the equation:

$$\mathrm{mgf}_t(h+1) = \mathrm{mgf}_t(h) \tag{3.2}$$

Since there is an explicit form for the moment generating function of the NIG, the equation can be solve to find the parameter. However, one should take some care with respect to parameters in the NIG-case. In particular, from (1.1.2) the mgf can at most exist in an interval $[-\alpha - \beta, \alpha - \beta]$.

Furthermore, certain precaution has to be made to ensure that there actually is a solution to the equation (3.2). This is due to the fact that such a solution can only exist if a function is at least locally convex or concave. Furthermore, due to convexity of the moment generating function, it can be established that there can only be at most one solution to this equation [17]. In particular, if $\alpha < 1/2$ or if $|\mu| \geq \delta\sqrt{\alpha^2 - 1}$, then no solution exists. This is further explored in [12, 4.1.1].

For the NIG-Lévy case, we then have the following equation if the price process is discounted by the continuously compounded interest rate:

$$\exp(-rt)\exp\left(\mu t + \delta t\sqrt{\alpha^2 - (\beta + h)^2} - \delta t\sqrt{\alpha^2 - (\beta + h + 1)^2}\right) = 1$$

This equation can have at most one solution given that the prerequisites above are satisfied[17, Lemma 1.9]. Through some rather long algebra one can solve this equation analytically with the following result:

$$h = -\beta - \frac{1}{2} - \sqrt{\frac{(\mu - r)^2\alpha^2}{\delta^2 + (\mu - r)^2} - \frac{(\mu - r)^2}{4\delta^2}} \tag{3.3}$$

From the definition of the Esscher measure dQ^h the exponential process under the new measure will then have the β-parameter replaced.

3.2.2 Esscher transform in two dimensions

In the previous section the Esscher transform for a single risky asset was established. The natural extension is then to markets driven by independent Lévy processes and in particular independent NIG-Lévy processes. Let the L_1, L_2 be such that:

$$L_1(t) \sim \text{nig}(\alpha_1, \beta_1, \delta_1 t, \mu_1 t)$$
$$L_2(t) \sim \text{nig}(\alpha_2, \beta_2, \delta_2 t, \mu_2 t)$$

Due to independence, the underlying probability measure is defined as:

$$\text{dP} = \text{dP}_1 \cdot \text{dP}_2 = \text{nig}_1(x_1)\text{nig}_2(x_2)\,\text{d}x_1\,\text{d}x_2$$

where dP_1, dP_2 are the underlying probability measures of L_1, L_2 respectively. The corresponding moment generating function with parameter $h = (h_1, h_2) \in \mathbb{R}^2$ in the NIG-case for $T = 1$ is then:

$$\begin{aligned}\text{mgf}(h) &= \text{E}[\exp\{(h_1, h_2)'(L_1, L_2)\}] \\ &= \exp\{\mu_1 h_1 + \delta_1\sqrt{\alpha_1^2 - \beta_1^2} - \delta_1\sqrt{\alpha_1^2 - (\beta_1 - h_1)^2}\} \\ &\cdot \exp\{\mu_2 h_2 + \delta_2\sqrt{\alpha_2^2 - \beta_2^2} - \delta_2\sqrt{\alpha_2^2 - (\beta_2 - h_2)^2}\} \qquad (3.4)\end{aligned}$$

For ease of notation the indication of time has been omitted. Since the process have independent margins, it is sufficient that each margin is a martingale by solving the equation:

$$\begin{aligned}\text{E}_Q[e^{L_1+L_2}] &= E[e^{L_1+L_2}e^{h_1L_1+h_2L_2} \cdot \text{mgf}^{-1}(h)] \\ &= \frac{\text{mgf}(h + (1,1))}{\text{mgf}(h)} \\ &= \begin{pmatrix} 1 \\ 1 \end{pmatrix}\end{aligned}$$

Clearly, this is not much different from the one-dimensional case.

3.2.3 Esscher transform of Dependent Lévy processes in two dimensions

As outlined in Chapter 2, applying copulas to the Brownian diffusion parts and the compound Poisson parts will ensure that the structure is preserved, so that both dependent processes are still Lévy processes. However, the underlying probability law of the process obtained through the conditional probability

distribution via the copula is not necessarily NIG-distributed anymore, as intensities may change slightly and the law does not have the simple structure of the NIG.

Clearly, if two processes are dependent, one will have impact on the Esscher transform of the other. This is not surprising, since arbitrage opportunities might still exist via the second asset even if the probability measure of the first is transformed to an equivalent martingale measure via an Esscher transform.

The law of the second process can be obtained directly given a copula which satisfies the smoothness requirement. Let the first simulation be a NIG-Lévy approximating process with the law:

$$L_1 \sim \zeta_1 t + \sigma_1(\epsilon) W_t + \sum_{i=1}^{N(t)} \gamma_i^{(1)}$$

The conditional probability measure for the increment in the second process is then:

$$\begin{aligned} L_2 \sim & \zeta_2 t \\ & + \sigma_2(\epsilon)\Phi^{-1}\left(C^{Ga}\left(v|u = \Phi(\sigma(\epsilon)W_t)\right)\right) \\ & + \sum_{i=1}^{N(t)} G_2^{-1}\left(C\left(v|u = G_1(\gamma_i^{(1)})\right)\right) \end{aligned} \tag{3.5}$$

where C^{Ga} is a Gaussian copula and C is a copula. The law of $\gamma_i^{(k)}, k = 1, 2$ is the NIG-Lévy jump measure truncated at ϵ with the respective NIG-parameters of L_1, L_2. ζ_1, ζ_2 are in similar manner drifts given in the Barndorff-Nielsen representations in addition to the Asmussen-Rosinski drift ($\mu_k(\epsilon)$). The $\sigma_k(\epsilon), k = 1, 2$ is as defined in (1.6). If $l_1 \sim L_1, l_1 < \epsilon$, the third term (CPP) will not contribute, and hence neither will the third term in L_2. G_1 and G_2^{-1} refer to the density function of ν^ϵ and its inverse with parameters of L_1 and L_2 respectively. This is necessary since the copula is defined on the unit square.

For the approximation, one can find a new measure $\mathrm{dP}_1^{h_1}(x)$ via the Esscher transform on the law of the approximation as:

$$\mathrm{dP}_1^{h_1}(x) = \frac{\exp\{\frac{\xi^2 - (\zeta_1 t)^2}{2\sigma_\epsilon^2 t}\} \cdot \mathrm{N}(x; \xi, \sigma_\epsilon^2 t)}{A} + \frac{\sum_{i=1}^{N(t)} \nu^\epsilon(x; \alpha_1, \beta_1 + h_1, \delta) K_\epsilon^{-1}}{A} \tag{3.6}$$

where

$$\begin{aligned} \xi &= \zeta_1 t + \sigma_1^2(\epsilon) h_1 t \\ A &= \exp\left\{\frac{\xi^2 - (\zeta_1 t)^2}{2\sigma^2(\epsilon)t}\right\}(\zeta_1 t + \sigma^2(\epsilon) h_1 t) + K_{\epsilon, \beta_1 + h_1} \end{aligned}$$

The entire algebra can be found in A.1.

For the second tilted probability measure in (3.5), one runs into trouble. The law of the dependent process is conditioned on the value of each jump of the other. Hence, one h_2 must be determined for each jump size, since the law will be slightly different. One Esscher parameter for each distribution can give rise to many different Esscher transforms, this makes it very difficult to find one unique pair of Esscher parameters in the dependent 2-dimensional case. While there has been some work in the field done by Albrecher and Predota[2], this was done on Asian option pricing. This is a multi-asset option, but these assets are comonotonic and hence the Esscher parameter is the same for each process. In the paper, the law of each process under the tilted measure is easily derived due to the simple structure of the normal inverse Gaussian moment generating function. The law of the approximation, on the other hand, has a moment generating function which is not that simple. One can observe that, not surprisingly, the Esscher transform has an effect on the normalized Lévy jump measure similar to that of the NIG. The Brownian term in the approximation yields a more complex result given in (3.6). Therefore, one can at best with conventional Esscher transforms write the second Esscher parameter in terms of the first: $h_2(h_1)$. To the best of knowledge, no paper has yet been published in this area.

This is the reason why an Esscher parameter is not found for the dependent processes in this thesis. Instead, the option pricing assumes a priori that one operates under risk-neutral conditions. Using the Esscher parameters from the two-dimensional independent processes will no doubt introduce arbitrage. Hence, leaving the Esscher transform out is a better choice.

The simulations are based on dependency specified via a Gumbel copula with parameters $\theta = 10, 20$, and the prices are contrasted with the independent price for a basket of exponential NIG-Lévy processes. Prices for several strikes are considered. The reasons for this restrictive approach is the high computation time required, and that the Gumbel is a flexible copula as it has dependency in both tails. The algorithm is also more numerically stable using Gumble copula than the others - where the instability can be accounted for due to reasons mentioned in Section 2.6.1.

3.3 Option pricing

We use the Monte Carlo method for pricing a basket option with two dependent underlyings. This is not a very efficient method, with expected runtime for several hours for each price. In this respect the result should be viewed more as a concept for the study of copulas' effect pricing of basket options with dependent underlyings under the exponential NIG-Lévy model.

There is a specific reason to the choice of the Monte Carlo method, since aggregating the exponential NIG-Lévy processes will not result in a Lévy process itself, and therefore more efficient methods such as Fourier transform cannot be used directly [9, 11.5].

A natural comparison is the price of a basket option in a market driven by

$K\backslash\theta$	0 (independent)	10	20
1	2.41	1.86	1.90
3.8	1.08	0.89	0.79
5.40	0.885	0.756	0.57
7.06	0.691	0.629	0.44
10	0.5399	0.5407	0.31

Table 3.1: Option prices under the Gumbel copula and different strikes, $\rho = 0$

independent exponential NIG-Lévy processes. The expression that will need to be evaluated for the $t = 0$ price is the following:

$$p(0) = \mathrm{E}_Q[e^{-rT}(e^{L_1(t)} + e^{L_2(t)} - K)^+]$$

Assume $P = Q$. In the Monte Carlo framework this means:

$$p^n(0) = \frac{e^{-rT}}{n} \sum_{i=1}^{n} \max(e^{L_1(T)} + e^{L_2(T)} - K, 0)$$

and hence $p^n(0) \to p(0)$ as $n \to \infty$.

The setup is $T = 1$ where T is weeks and $r = 1.05$ annually with a total trading days of 252 per year. The underlyings are sampled per minute, which gives a total of about $n = 10000$ steps in each path simulation. The number of path simulations is 4500. The convergence seems good at this number, but it has limited the number of pricings possible. Due to long computation times, pricing is only done on the independent basket and on baskets with the Gumbel dependency structure with $\theta = 10, 20$. One can see that the price converges in Figure 3.1. The resulting option prices can be found in Table 3.1.

3.4 Concluding remarks

As can be seen, the dependency structure can have an effect on the option price. The spread between the option prices decreases for far out-of-the-money options. This makes sense under the Gumbel copula since the dependence is strong in the upper tail. However, it's important to note that one need to be very far out in the tails to see that the price of the dependent basket is higher than that of the independent basket. The simulations $\theta = 20$ may come as a surprise, but one explanation is that with such high dependence the first process has to produce a large jumps for the second process to have a large jump. This will certainly limit the paths for the pairs produced by simulation, and in this particularly case 4500 simulations may not be enough to ensure that the price is close to the theoretical option price. When looking at the required computation time, one may ask whether the use copulas joining large jumps have enough impact on the option price. While more computations with different copulas would certainly be necessary to establish a pattern, there are some indications

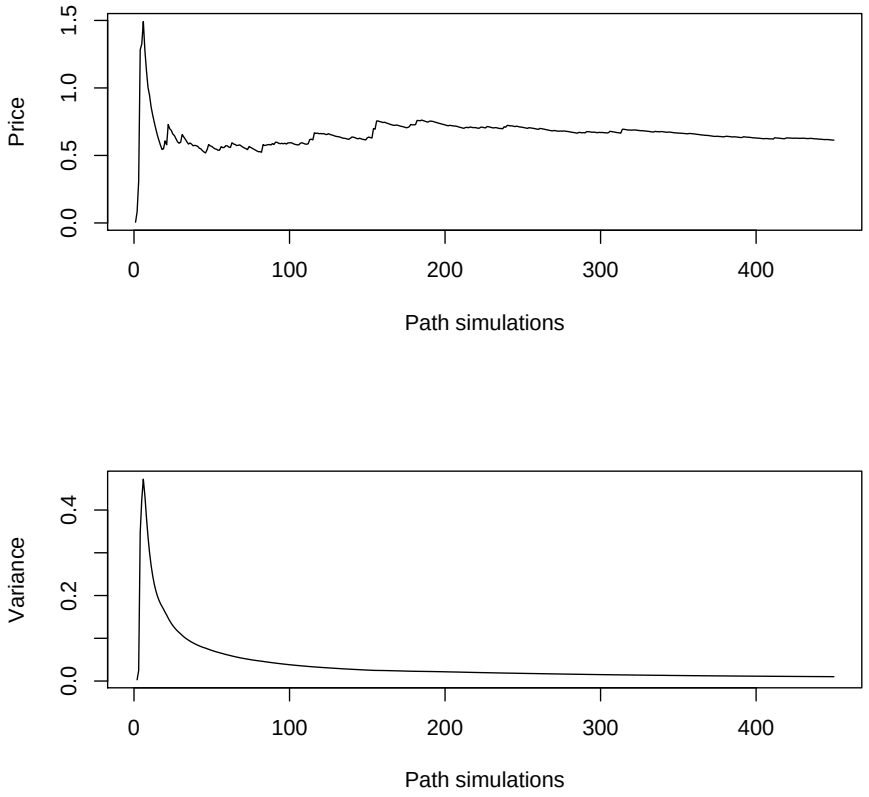

Figure 3.1: Convergence of option price as number of path-simulations (multiple of 10) increase.

that copulas have most effect in the tails where they can in some cases produce higher option prices than in the independent case. However, with the round-off errors present in this implementation, copulas are a concept that should be applied with care.

Chapter 4

Conclusion

The NIG-Lévy process can be approximated well by a rejection based sampler, but it is computationally demanding. The algorithm using the Metropolis-Hastings and Accept-Reject algorithms both give very good results as long as the proposal distribution has a good fit to the target. However, the Metropolis-Hastings has better acceptance rate for proposals which have worse fit, and hence this is the preferred algorithm. The implementation largely due to results published by Sebastian Rasmus seem to gone mostly unnoticed. From the results presented in this thesis, it provides a very elegant solution with fast convergence.

The application of copulas to large jumps of the NIG-Lévy has certain interesting properties. In particular, care needs to be taken when applying copulas to quantiles very close to 0 or 1. There are some round-errors that can introduce big variation in the dependent jumps, and the size of the jumps can change much based on the 4th significant digit in the quantile returned by the copula. This is an indication that copulas must be applied with care when joining distributions with thin tails. Likewise, an implementation in software care must be taken to produce numerically stable results. Finding the quantiles for each jump in the jump measure in particularly demanding and this is the main bottleneck in the algorithm.

Applying copulas to the large jumps over ϵ has indicated that the copulas in some cases have direct effect on the option prices.The jumps generated by the dependent process do in general have more variance than the jump in the original process, and one may take this as an indication that option pricing by Monte Carlo simulations will be very demanding. Due to restrictions on hardware, simulations were done with 4500 samples for each option price, but this may not be enough to ensure a sufficiently small Monte Carlo error. Correlation in the small jumps less than ϵ seems to have less effect on the overall location of the process at T, and hence on the option price. The large jumps will in many cases determine the end value of the process at T.

In summary, copulas to join two approximated NIG-Lévy processes cannot be recommened for option pricing as one will have difficulty determining an equivalent martingale measure. In addition, the numerical instability that can occur in standard implementations may prove to be another issue. While Lévy copulas

were found problematic in this thesis, they can possibly lead to less complicated expressions, and maybe in the future, these will prove to be an efficient solution to the above problems.

Appendix A

A.1 Esscher transform for the NIG-Lévy approximation

Define:

$$\begin{aligned}\mathrm{dP}_1(x) &\sim \zeta_1 t + \sigma_1(\epsilon)\mathrm{N}(0,t) + \sum_{i=1}^{N(t)} \nu^{\epsilon}(x) K_{\epsilon}^{-1} \\ &\sim \mathrm{N}(\zeta_1 t, \sigma^2(\epsilon)t) + \sum_{i=1}^{N(t)} \nu^{\epsilon}(x) K_{\epsilon}^{-1}\end{aligned}$$

The tilted measure is then defined as $\mathrm{dQ}^h(x) = e^{hx}\,\mathrm{dP}(x)$ for the Esscher parameter h.

The Brownian part

$$\begin{aligned}e^{hx}\mathrm{N}(\zeta_1 t, \sigma^2(\epsilon)t) &= \frac{1}{\sqrt{2\pi t}\sigma(\epsilon)}\exp\left\{\frac{-(x-\zeta_1 t)^2 - 2\sigma^2(\epsilon)thx}{2\sigma^2(\epsilon)t}\right\} \\ &= \frac{1}{\sqrt{2\pi t}\sigma(\epsilon)}\exp\left\{\frac{-(x^2 - 2x\zeta_1 t + (\zeta_1 t)^2 - 2\sigma^2(\epsilon)thx)}{2\sigma^2(\epsilon)t}\right\} \\ &= \frac{1}{\sqrt{2\pi t}\sigma(\epsilon)}\exp\left\{\frac{-(x^2 - 2x(\zeta_1 t + \sigma^2(\epsilon)th) + (\zeta_1 t)^2)}{2\sigma^2(\epsilon)t}\right\} \\ &= \frac{1}{\sqrt{2\pi t}\sigma(\epsilon)}\exp\left\{\frac{-(x-(\zeta_1 t + \sigma^2(\epsilon)))^2 - (\zeta_1 t)^2 + ((\zeta_1 t + \sigma^2(\epsilon))^2}{2\sigma^2(\epsilon)t}\right\} \\ &= \exp\left\{\frac{-(\zeta_1 t)^2 + (\zeta_1 t\sigma^2(\epsilon)th)^2}{2\sigma^2(\epsilon)t}\right\}\mathrm{N}(\zeta_1 t + \sigma^2(\epsilon)th, \sigma^2(\epsilon)t)\end{aligned}$$

The compound Poisson part

This can simply be written as:

$$e^{hx}\nu^{\epsilon}(x;\alpha,\beta,\delta)K_{\epsilon}^{-1} = \nu^{\epsilon}(x;\alpha,\beta+h,\delta)K_{\epsilon}^{-1}$$

The normalization constant

$$\mathrm{E}_P[e^{hx}] = \exp\left\{\frac{-(\zeta_1 t)^2 + (\zeta_1 t + \sigma^2(\epsilon)ht)^2}{2\sigma^2(\epsilon)t}\right\}(\zeta_1 t + \sigma^2(\epsilon)ht) + K_{\epsilon,\beta+h}$$

where $K_{\epsilon,\beta+h}$ stems from the expectation of the compound Poisson, which is the intensity K_ϵ times the expected jump size. This results in the tilted measure:

$$\mathrm{dQ}^h(x) = \frac{\exp\{\frac{\xi^2-(\zeta_1 t)^2}{2\sigma_\epsilon^2 t}\}\cdot \mathrm{N}(x;\xi,\sigma_\epsilon^2 t)}{A} + \frac{\sum_{i=1}^{N(t)}\nu^{\epsilon}(x;\alpha_1,\beta_1+h,\delta)K_{\epsilon}^{-1}}{A}$$

where

$$\begin{aligned}\xi &= \zeta_1 t + \sigma_1^2(\epsilon)ht\\ A &= \exp\left\{\frac{\xi^2-(\zeta_1 t)^2}{2\sigma^2(\epsilon)t}\right\}(\zeta_1 t + \sigma^2(\epsilon)ht) + K_{\epsilon,\beta_1+h}\end{aligned}$$

Appendix B

B.1 Source code: approximation

B.1.1 NIG-Levy2.R

```
#Simulation of NIG-Levy processes vs approximation
library(fBasics);
library("HyperbolicDist");
library("fCopulae");

source("Copula-functions.R");
source("MonteCarlo-functions.R");
source("NIG-functions.R");

getAsmussenRosinskiMu <- function(epsilon, alpha, beta, delta, mu){
        muFn <- function(x){return(param_mu(x,alpha,beta,mu,delta))}
        muEpsilon <- integrate(muFn, -1, -epsilon)$value;
        muEpsilon <- -1*(muEpsilon+integrate(muFn, epsilon,1)$value);

        return(muEpsilon);
}

getAsmussenRosinskiSigma <- function(epsilon, alpha, beta, delta, mu){
        sigmaFn <- function(x){return(param_sigma(x, alpha,beta,mu,delta))}
        sigmaEpsilon <- integrate(sigmaFn, -epsilon, 1)$value - integrate(sigmaFn,
             epsilon, 1)$value;

        return(sigmaEpsilon);
}

getBarndorffDrift <- function(alpha, beta, delta, mu){
        gamma <- function(x){
                return(sinh(beta*x)*besselK(alpha*x,1));
        }

        drift <- 2*delta*alpha/pi*integrate(gamma, 0, 1)$value + mu;

        return(drift);
}

nigLevy <- function(p){
#nigLevy <- function(n, alpha,beta,mu, delta, epsilon, omega){
        #Here we'll have the NIG-LEVY simulation.

        alpha <- p$alpha;
        beta <- p$beta
        mu <- p$mu
        delta <- p$delta
        epsilon <- p$epsilon
        omega <- p$omega;
        n <- p$n;

        if(p$verbose){
                show("--NIG parameters are:--");
                show(c("Alpha:", alpha));
                show(c("Beta:", beta));
                show(c("Mu:", mu));
                show(c("Delta:", delta));
                show(c("Epsilon:", epsilon));
                show(c("n:", n));
        }

        theta <- c(-0.5, alpha, beta, delta, mu);
        nig <- rghyp(n-1, theta);
        #nig <- rnig(n-1, alpha = alpha, beta = beta, delta = delta, mu = mu);

        if(p$verbose){
                show(c("Length nig:", length(nig)));
```

```
        }

        numGeqEpsilon <- 0;

        geqEJumps <- nig[abs(nig) >= epsilon];
        numEGeqEpsilon <- length(geqEJumps);
        geqWJumps <- nig[abs(nig) >= omega];
        numWGeqEpsilon <- length(geqWJumps);

        if(p$verbose){
                show(c("NIG abs incr. max:", max(abs(nig))));
                show(c("NIG abs incr. min:", min(abs(nig))));
                show(c("NIG incr. var:", var(nig)));
                show(c("NIG incr. mean:", mean(nig)));
                show(c("Estimated intensity of big epsilon jumps:", numEGeqEpsilon))
                    ;
                show(c("Estimated intensity of big omega jumps:", numWGeqEpsilon));
                show(c("Max of abs big jumps:", max(abs(geqEJumps))));
                show(c("Var of big jumps:", var(geqEJumps)));
                show(c("Mean of abs big jumps abs:", mean(abs(geqEJumps))));
        }

        X <- cumsum(c(0, nig))

        result <- list(X=X, incr=nig);

        return(result);
}

approxNuNiG <- function(params){
        #alpha, beta, mu, delta: NiG parameters
        #lambda: Jump intensity
        #The jump distribution is the normalized NIG Levy measure
        #-----------------------------------------------------------------

        #These can be taken directly from the parameters:
        alpha <- params$alpha;
        beta <- params$beta;
        mu <- params$mu;
        delta <- params$delta;
        maxErr <- params$maxErr;
        T <- params$T;
        t <- params$t;
        n <- params$n;

        rho <- params$rho;

        #Defining the levy measure function:
        nu <- function(x){
                val <- nu_nig(x, alpha, beta, mu, delta);
                return(val);
        }

        #Determining truncation levels: (epsilon, omega):
        epsilon <- 8*maxErr^2*delta/pi;

        params$epsilon <- epsilon;

        #Determining intensity: (lambda)
        lambda <- integrate(nu, -Inf, -epsilon)$value;
        lambda <- lambda + integrate(nu, epsilon, Inf)$value


        #We determine omega (It is determined based on the change in derivative):
        if(params$omegaOverride){
                omega <- params$omegaOverride;
        }else{
                S <- function(w){
                        v <- sinh(beta*w)*besselK(alpha*w,1)/w;

                        return(v);
                }

                C <- function(w){
                        v <- cosh(beta*w)*besselK(alpha*w,1)/w;

                        return(v);
                }

                eqn <- function(w){
                        v <- beta*S(w) - 2/w*C(w)-alpha*w/abs(w)*besselK(alpha*w,0)/
                             besselK(alpha*w,1)*C(w)  + (alpha+abs(beta))*epsilon^2/
                             w^2*C(epsilon);
                        return(v);
                }

                omega <- uniroot(eqn, c(epsilon, 15*alpha^-1))$root;

        }

        #The lambda is for one time unit, so multiply with time units:
        lambda <- lambda*T;

        params$lambda <- lambda;

        #Determining Brownian approximation parameters: (mu, sigma)
        muEpsilon <- getAsmussenRosinskiMu(epsilon, alpha, beta, delta, mu)
```

```
sigmaEpsilon <- getAsmussenRosinskiSigma(epsilon, alpha, beta, delta, mu);

#This is the drift according to Barndorff-Nielsens representation:
drift <- getBarndorffDrift(alpha, beta, delta, mu);

##############################################
#NOW WE HAVE FOUND THE NECESSARY PARAMETERS#
##############################################

if(params$verbose){
        show("--Approx parameters are--");
        show(c("Alpha", alpha));
        show(c("Beta", beta));
        show(c("Mu", mu));
        show(c("Delta", delta));
        show(c("Epsilon", epsilon));
        show(c("Omega", omega));
        show(c("Lambda", lambda));
        show(c("muEpsilon", muEpsilon));
        show(c("sigmaEpsilon", sigmaEpsilon));
        show(c("drift", drift));
}

X <- matrix(NA, n);

#We let both the brownian and compound possion start in 0.
Bm <- 0;
Cpp <- 0;

#Number of jumps
N <- rpois(1, lambda);

show(c("Number of jumps", N));

#When does the jumps occur?
U <- runif(N, 0, T);

#Sort them increasingly, this makes them exponential.
jTimes <- sort(U);

#The jump sizes:
jSizes <- c();

rCand <- function(n){
        val <- rCGF2(n, alpha, beta, mu, delta, epsilon, omega, params$
            rProposal);
        return(val);
}

pCand <- function(x){
        val <- pCGF2(x, alpha, beta, delta, epsilon, omega, params$dProposal
            );

        return(val);
}

dCand <- function(x){
        c <- nu(epsilon)/pCand(epsilon);

        val <- pCand(x)*c;

        return(val);
}



MHn <- params$MHn
x_0 <- rCand(N);

args <- list(N=N, target=nu, candidate=pCand, rcandidate=rCand, x_0=x_0, max
    _n=MHn, debug=FALSE, verbose=FALSE);


if(params$ARmethod == "AcceptReject"){
        args$candidate <- dCand;
}

mh <- do.call(params$ARmethod, args);

jSizes <- mh$values;

#Some statistics:
if(params$verbose){
        show(c("Mean acceptance rate:", mean(mh$ar)));
        show(c("Rate of alpha=0:",mean(mh$a0)));
        show(c("jSizes max:", max(abs(jSizes))));
        show(c("jSizes min:", min(abs(jSizes))));
        show(c("jSizes var:", var(jSizes)));
        show(c("jSizes mean:", mean(abs(jSizes))));
        show(c("Number of jumps over Omega:", length(jSizes[abs(jSizes) >=
            omega])));
}

#Draw the increments of the Brownian Motion. It's simply easier to draw all
    at once, even if no dependent
#series is to be generated.
brownians <- rnorm2d(n, rho);
```

```
BmIncr <- sqrt(sigmaEpsilon*step)*brownians[,1];


#vector of jump times
#vector of all timesteps
#sum up all jumps below each time

times <- array(t);

cpps <- apply(times, 1, function(x) return(sum(jSizes[jTimes <= x])));

bms <- cumsum(c(0,BmIncr[1:n-1]));

X <- (drift+muEpsilon)*t + bms + cpps;

mar <- mean(mh$ar);

################################################
#GENERATING THE DEPENDENT PROCESS              #
################################################


#If param says to generate two dependent realizations, just do it!
if(params$dep){
        #Retrieve parameters:
        alphaDep <- params$alphaDep;
        betaDep <- params$betaDep;
        muDep <- params$muDep;
        deltaDep <- params$deltaDep;

        #Get the first param of triplet from Barndorff-Nielsen
            representation:
        driftDep <- getBarndorffDrift(alphaDep, betaDep, deltaDep, muDep);

        #Get Asmussen-Rosinski mu and sigma:
        muEpsilonDep <- getAsmussenRosinskiMu(epsilon, alphaDep, betaDep,
            deltaDep, muDep);
        sigmaEpsilonDep <- getAsmussenRosinskiSigma(epsilon, alphaDep,
            betaDep, deltaDep, muDep);


        if(params$verbose){
                show("--Approx DEP parameters are--");
                show(c("Alpha", alphaDep));
                show(c("Beta", betaDep));
                show(c("Mu", muDep));
                show(c("Delta", deltaDep));
                show(c("muEpsilonDep", muEpsilonDep));
                show(c("sigmaEpsilonDep", sigmaEpsilonDep));
                show(c("driftDep", driftDep));
        }

        #Get the Normal approximation increments:
        BmIncrDep <- sqrt(sigmaEpsilonDep*step)*brownians[,2];


        #Get large jumps from Copula:
        if(params$theta == 0){
                show("Theta == 0");

                rCandDep <- function(n){
                        val <- rCGF2(n, alphaDep, betaDep, muDep, deltaDep,
                            epsilon, omega, params$rProposal);
                        return(val);
                }

                pCandDep <- function(x){
                        val <- pCGF2(x, alphaDep, betaDep, deltaDep, epsilon
                            , omega, params$dProposal);

                        return(val);
                }

                dCandDep <- function(x){
                        c <- nu(epsilon)/pCandDep(epsilon);

                        val <- pCandDep(x)*c;

                        return(val);
                }


                MHn <- params$MHn
                x_0 <- rCandDep(N);

                argsDep <- list(N=N, target=nu, candidate=pCandDep,
                    rcandidate=rCandDep, x_0=x_0, max_n=MHn, debug=FALSE,
                    verbose=FALSE);


                if(params$ARmethod == "AcceptReject"){
                        args$candidate <- dCandDep;
                }

                mh <- do.call(params$ARmethod, argsDep);

                jSizesDep <- list();
```

```
                        jSizesDep$Y <- mh$values;
                }else{
                        jSizesDep <- depNIGLevyCopula(jSizes, params);
                }


                #Generate the path:
                bmsDep <- cumsum(c(0,BmIncrDep[1:n-1]));
                cppsDep <- apply(times, 1, function(x) return(sum(jSizesDep$Y[jTimes
                      <= x])));

                Y <- (driftDep+muEpsilonDep)*t + bmsDep + cppsDep;



                nigLevy <- list(X=X, Y=Y, jumpTimes = jTimes, jumpSizes = jSizes, Fv
                     =jSizesDep$Fv, u=jSizesDep$u, epsilon=epsilon, omega=omega, ar
                     = mar);
        }else{
                nigLevy <- list(X=X, jumpTimes = jTimes, jumpSizes = jSizes, epsilon
                     =epsilon, omega=omega, ar = mar);
        }

        return(nigLevy);
}

#Common variables
T <- 1;
n <- T*10000;
step <- T/(n-1);
t <- seq(0,T,step);

#TODO: Approximation arguments by list:
approxArgs <- list(alpha = 1,
                   beta = 0,
                   delta = 1,
                   mu = 0,
                   T = T,
                   t = t,
                   n = n,
                   theta = 2,
                   rho = 0,
                   dep = TRUE,
                   alphaDep = 1,
                   betaDep = 0,
                   deltaDep = 1,
                   muDep = 0,
                   step = step,
                   MHn = 1000,
                   maxErr = 0.01,
                   #omegaOverride = 1,
                   omegaOverride = FALSE,
                   verbose=TRUE,
                   #ARmethod="AcceptReject",
                   ARmethod="MetropolisHastingsVI",
                   #rProposal="rInvX2",
                   #dProposal="dInvX",
                   rProposal="rQuadFn2",
                   dProposal="pQuadFn",
                   copulaMtd="clayton.gen");

nigArgs <- list(alpha = 1,
                    beta = 0,
                    delta = 1*step,
                    mu = 0*step,
                    n = n,
                    verbose=TRUE);

run <- function(withNIG = TRUE){

        #The approximation of the NiG using M-H on jump measure:
        system.time(appr <- approxNuNiG(approxArgs), gcFirst=TRUE);

        write.table( matrix(c(t[1:n],appr$X), nc=2), file="NIG-Levy-approx-hyp.data"
             , row.names=FALSE, col.names=FALSE);

        appr$incr <- appr$X[2:length(appr$X)]-appr$X[1:(length(appr$X)-1)];
        appr$q <- quantile(appr$incr,seq(0,1,0.001));

        if(approxArgs$dep){
                fname <- paste(c("NIG-Levy-dependent-",approxArgs$copulaMtd,"-",
                     approxArgs$theta,".data"), collapse="");
                fdata <- matrix(c(t[1:n], appr$X, appr$Y), nc=3);
                write.table(fdata, fname, row.names=FALSE, col.names=FALSE);
                appr$incrY <- appr$Y[2:length(appr$Y)]-appr$Y[1:(length(appr$Y)-1)];
        }

        value <- list(appr=appr)

        e <- appr$epsilon;
        w <- appr$omega;

        nigArgs$epsilon <- e;
        nigArgs$omega <- w;

        if(withNIG){
                #Doing a simulation with the builtin NIG function:
                realNIG <- nigLevy(nigArgs);
                write.table(matrix(c(t[1:n],realNIG$X),nc=2), "NIG-Levy-nig-hyp.data
```

```
                    ", row.names=FALSE, col.names=FALSE);
                realNIG$q <- quantile(realNIG$incr,seq(0,1,0.001));

                value$real <- realNIG;
        }


        return(value);
}
```

B.1.2 NIG-functions.R

```
###This file contains all functions associated with levy measure, ###
###limit functions of levy measure, riemann integration and       ###
###calculation of mu/sigma parameters for small jump approximation###


nu_nig <- function(x, alpha, beta, mu, delta){

        y = delta*alpha/(pi*abs(x))*exp(beta*x)*besselK(alpha*abs(x),1);

        y[is.infinite(y)] <- 0;
        y[is.nan(y)] <- 0;


        return(y);
}

#exp_fn <- function(x, alpha, beta, epsilon, omega){
#       y = (alpha-beta)*exp(-(alpha-beta)*(abs(x)-omega));
#       return(y);
#}
#
#quad_fn <- function(x, epsilon, omega,delta=pi){
#       y = delta/(pi*x^2);
#       return(y);
#}
#
#intRiemannNIG <- function(lower, upper, N, fn, alpha, beta, mu, delta){
#       step = (upper-lower)/N;
#
#       steps = seq(lower, upper, step);
#
#       vol = 0;
#
#       print(grid);
#
#       for(x in steps){
#               vol = vol + step*fn(x, alpha, beta, mu, delta);
#       }
#       return(vol);
#}

param_mu <- function(x,alpha,beta,mu,delta){
        return(x*nu_nig(x,alpha,beta,mu,delta));
}

param_sigma <- function(x, alpha,beta,mu,delta){
        return(x^2*nu_nig(x,alpha,beta,mu,delta));
}
```

B.1.3 MonteCarlo-functions.R

```
source("NIG-functions.R");

MetropolisHastingsVI <- function(N, target, candidate, rcandidate, x_0, max_n, debug
    =FALSE, verbose=FALSE){
        x_test <- c(x_0);

        #Draw the full vector of random numbers:
        uni <- runif(max_n * N);
        rcands <- rcandidate(max_n * N);

        #This is the complete vectors of draws:
        draws <- c(x_0, rcands);

        #Look up the density height of all draws, incl. start value.
        probCand <- candidate(draws);
        probTarget <- target(draws);

        #We need some indexes to keep track of which values we are using:
        idxStart <- 1;
        idxCurrent <- 1;
        idxProposal <- length(x_0) + 1 ;

        #This is debug stuff:
        n_accept <- 0;
        n_reject <- 0;
        n_alpha <- 0;
```

```
        n_x2yIs0 <- 0;

        n <- 0;

        #This is the vector of draws:
        z <- c();

        while(TRUE){
                n <- n +1;

                #Get the uniform and proposal draws by index.
                #Recall that draws has x_0 at the beginning.
                u <- uni[idxProposal - N];
                y <- draws[idxProposal];

                y2x <- probTarget[idxProposal] * probCand[idxCurrent];
                x2y <- probTarget[idxCurrent] * probCand[idxProposal];

                if(x2y > 0){
                        alpha <- min(1, y2x/x2y);
                }else{
                        alpha <- 1;
                        n_x2yIs0 <- n_x2yIs0 + 1;
                }

                if(alpha == 1) n_alpha <- n_alpha + 1;

                if(u < alpha){
                        #We set the index of the current draw to the accepted
                            proposal
                        idxCurrent <- idxProposal;

                        n_accept <- n_accept + 1;
                        y_0 <- draws[idxProposal];
                }else{
                        #If not accepted, the index of the current draw stays the
                             same,
                        #so we do nothing except some statistics

                        n_reject <- n_reject + 1;
                        y_0 <- draws[idxCurrent];
                }

                #x <- append(x, y_0);

                if(n %% max_n == 0){
                        #This is the final draw, so append to draw vector:
                        z <- append(z, y_0);

                        #update which initial value is to be used:
                        idxStart <- idxStart + 1;

                        #Change current index to next starting value.
                        idxCurrent <- idxStart;
                }

                if(n < max_n * N){
                        idxProposal <- idxProposal + 1;
                }else{
                        break;
                }
        }

        result <- list(values=z, ar=n_accept/(n_accept + n_reject), a0=n_alpha/(n_
             accept + n_reject), x2y0=n_x2yIs0/(n_accept + n_reject));

        return(result);
}

AcceptReject <- function(N, target, candidate, rcandidate, x_0, max_n, debug=FALSE,
     verbose=FALSE){
        x_test <- c(x_0);

        #Draw the full vector of random numbers:
        uni <- runif(max_n * N);
        rcands <- rcandidate(max_n * N);

        #This is the complete vectors of draws:
        draws <- c(x_0, rcands);

        #Look up the density height of all draws, incl. start value.
        probCand <- candidate(draws);
        probTarget <- target(draws);

        #We need some indexes to keep track of which values we are using:
        idxStart <- 1;
        idxCurrent <- 1;
        idxProposal <- length(x_0) + 1;

        #This is debug stuff:
        n_accept <- 0;
        n_reject <- 0;
        n_alpha <- 0;
        n_x2yIs0 <- 0;
```

```
        n <- 0;

        #This is the vector of draws:
        z <- c();

        while(TRUE){
                n <- n +1;

                #Get the uniform and proposal draws by index.
                #Recall that draws has x_0 at the beginning.
                u <- uni[idxProposal - N];
                y <- draws[idxProposal];

                alpha <- probTarget[idxProposal]/probCand[idxProposal];

                if(alpha >= 1) n_alpha <- n_alpha + 1;

                if(u < alpha){
                        #We set the index of the current draw to the accepted
                             proposal
                        idxCurrent <- idxProposal;

                        n_accept <- n_accept + 1;
                        y_0 <- draws[idxProposal];
                }else{
                        #If not accepted, the index of the current draw stays the
                             same,
                        #so we do nothing except some statistics

                        n_reject <- n_reject + 1;
                        y_0 <- draws[idxCurrent];
                }

                if(n %% max_n == 0){
                        #This is the final draw, so append to draw vector:
                        z <- append(z, y_0);

                        #update which initial value is to be used:
                        idxStart <- idxStart + 1;

                        #Change current index to next starting value.
                        idxCurrent <- idxStart;
                }

                if(n < max_n * N){
                        idxProposal <- idxProposal + 1;
                }else{
                        break;
                }
        }

        result <- list(values=z, ar=n_accept/(n_accept + n_reject), a0=n_alpha/(n_
             accept + n_reject), x2y0=n_x2yIs0/(n_accept + n_reject));
        return(result);
}

#Candidate Generating Density Functions

#alpha, beta: NiG parameters. omega: truncation/shift.
#u: uniform variate, pos: 1 is positive jump, -1 negative
rExpFn <- function(n, alpha, beta, omega){
        #Notice that our implementation is symmetric

        #We use absolute value of beta to be sure we are dominating
        #the biggest tail, hence also the lowest.

        uMax <- (alpha-abs(beta));

        u <- runif(n, max=uMax);

        x <- -1*(log(u) - log(uMax))/uMax + omega;

        if(prod(abs(x) > omega) == 0){
                show(c("Value less than omega drawn in Exp!!",x));
                show(c("Are you sure your parameters are right, epsilon < omega?"));
        }

        return(x);
}

rQuadFn2 <- function(n, epsilon, omega){

        u <- runif(n);

        x <- (epsilon*omega)/(omega - u*(omega-epsilon));

        return(x)
}

pExpFn <- function(x, alpha, beta, omega){
        #Since the density is symmetric we eval x in the pos. density
        #There is no need to normalize the result since we only care for the height
        #of the density, unnormalized or not.

        p <- rep(NA, length(x));
```

```
        idxLeqOmega <- (abs(x) < omega);
        p[idxLeqOmega] <- 0;

        p[!idxLeqOmega] <- (alpha-beta)*exp(-(alpha-abs(beta))*(abs(x[!idxLeqOmega])
            - omega));

        return(p);
}

pQuadFn <- function(x, epsilon, omega){
        #Since the density is symmetric:
        absX <- abs(x);

        p <- rep(NA, length=length(x));
        idxIntv <- (absX > abs(omega) | absX < abs(epsilon));
        p[idxIntv] <- 0;
        p[!idxIntv] <- omega*epsilon/(abs(omega-epsilon)*x[!idxIntv]^2);

        #This is our value, since we only want the height of the density:
        return(p);
}

rCGF2 <- function(n, alpha, beta, mu, delta, epsilon, omega, rDist){
        #An optimized version for any n!
        #-------------------------------

        #Draw n uniform variables:
        u <- runif(n);

        #Pseudo-function:
        new_nu <- function(x) nu_nig(x, alpha, beta, mu, delta);

        #Find the draw-weights:
        mExp1 <- integrate(new_nu, omega, Inf)$value;
        mExp2 <- integrate(new_nu, -Inf, -omega)$value;
        mQuad1 <- integrate(new_nu, -omega, -epsilon)$value;
        mQuad2 <- integrate(new_nu, epsilon, omega)$value;

        pExp1 <- mExp1/(mExp1+mExp2+mQuad1+mQuad2);
        pExp2 <- mExp2/(mExp1+mExp2+mQuad1+mQuad2);
        pQuad1 <- mQuad1/(mExp1+mExp2+mQuad1+mQuad2);
        pQuad2 <- mQuad2/(mExp1+mExp2+mQuad1+mQuad2);

        #This is our vector of draws:
        x <- rep(0, length=n);

        #Which indexes will take draws from which area:
        idxLowerExp <- (u < pExp1);
        idxLowerQuad <- (u >= pExp1 & u < pExp1 + pQuad1);
        idxUpperQuad <- (u >= pExp1 + pQuad1 & u < pExp1 + pQuad1 + pQuad2);
        idxUpperExp <- (u >= pExp1 + pQuad1 + pQuad2);


        #Find the number of realizations to make from each area:
        n_lower_exp <- length(u[idxLowerExp]);
        n_lower_quad <- length(u[idxLowerQuad]);
        n_upper_quad <- length(u[idxUpperQuad]);
        n_upper_exp <- length(u[idxUpperExp]);

        #Draw the respective number of realizations:
        #(While one can further optimize it to draw only once from each,
        # the author finds it clearer to leave 4 invocations in case one '
        # later wants the CGF to be asymmetric).

        #Draws from the exponential area.
        drawsLowerExp  <-  -1*rExpFn(n_lower_exp, alpha, beta, omega);
        drawsUpperExp  <-  rExpFn(n_upper_exp, alpha, beta, omega);

        randArgsLower <- list(n=n_lower_quad, epsilon=epsilon, omega=omega);
        randArgsUpper <- list(n=n_upper_quad, epsilon=epsilon, omega=omega);

        drawsLowerQuad <- -1*do.call(rDist, randArgsLower);
        drawsUpperQuad <- do.call(rDist, randArgsUpper);

        #Assign the draws in the right order to the draws vector:
        x[idxLowerExp] <- drawsLowerExp;
        x[idxLowerQuad] <- drawsLowerQuad;
        x[idxUpperQuad] <- drawsUpperQuad;
        x[idxUpperExp] <- drawsUpperExp;


        #Check that all the draws are sane:
        if(prod(abs(x) >= epsilon) == 0){
                show("Value less than epsilon drawn in optimized version!!");
        }

        #Return the draws:
        return(x);
}

pCGF2 <- function(x, alpha, beta, delta, epsilon, omega, dDist){
        #A vector capable version of pCGF.

        p <- rep(0, length(x));

        #We scale the exponential probability to make our proposal
```

```
        #continuous, but not differentiable:
        c_quad <- alpha*delta;
        distArgs <- list(x=omega, epsilon=epsilon, omega=omega);
        c_exp <- c_quad*do.call(dDist, distArgs)/pExpFn(omega, alpha, beta, omega);

        idxExpNeg  <- (x < -omega);
        idxQuadNeg <- (x <= -epsilon & x >= -omega);
        idxQuadPos <- (x >= epsilon & x <= omega);
        idxExpPos  <- (x > omega);

        #Exponential function is alwasy used in tails:
        p[idxExpNeg] <- pExpFn(x[idxExpNeg], alpha, beta, omega)*c_exp;
        p[idxExpPos] <- pExpFn(x[idxExpPos], alpha, beta, omega)*c_exp;

        distArgsNeg <- list(x=x[idxQuadNeg], epsilon=epsilon, omega=omega);
        distArgsPos <- list(x=x[idxQuadPos], epsilon=epsilon, omega=omega);

        p[idxQuadNeg] <- do.call(dDist, distArgsNeg);
        p[idxQuadPos] <- do.call(dDist, distArgsPos);

        return(p);
}

dInvX <- function(x, epsilon, omega){
        absX <- abs(x);

        p <- rep(NA, length=length(x));
        idxIntv <- (absX > abs(omega) | absX < abs(epsilon));
        p[idxIntv] <- 0;
        p[!idxIntv] <- 1/(log(omega/epsilon)*x);

        return(p);
}

rInvX2 <- function(n, epsilon, omega){
        u <- runif(n);

        x <- exp(u*log(omega/epsilon))*epsilon;

        return(x);
}
```

B.1.4 Metropolis-Hastings.R

```
source("MonteCarlo-functions.R");
source("NIG-functions.R");

nu <- function(x){
        val = nu_nig(x, alpha, beta, mu, delta);

        return(val);
}

rCand <- function(n){
        val = rCGF2(n, alpha, beta, delta, epsilon, omega);

        return(val);
}

pCand <- function(x){
        val = pCGF2(x, alpha, beta, delta, epsilon, omega);

        return(val);
}

alpha <- 1;
beta <- 0;
mu <- 0;
delta <- 1;
omega <- 1.38;
epsilon <- 0.0002;
expScale <- 1;

MHn <- seq(100, 5000, by=25);
N = 1000;
draws = c();

dTarget <- function(x){return(dexp(x, 30))}
dProp <- function(x){return(dexp(x, 25))}
rProp <- function(n){return(rexp(n, 25))}

nu <- function(x){
        val <- nu_nig(x, alpha, beta, mu, delta);
        return(val);
}

rCand <- function(n){
        val <- rCGF2(n, alpha, beta, mu, delta, epsilon, omega, expScale);
        return(val);
}

pCand <- function(x){
        val <- pCGF2(x, alpha, beta, delta, epsilon, omega, expScale);
        return(val);
```

```
}

x_0 <- rep(rCand(1), N);

#dTarget <- function(x) return(nu(x)*(abs(x)>=epsilon));
#dProp <- function(x) return(pcauchy(x, scale=1/7));
#rCand <- function(n) return(rcauchy(n, scale=1/7));

vars <- c();
varvars <- c();


for(i in MHn){
        show(i);
        #mh = MetropolisHastingsVI(N, dTarget, dProp, rProp, x_0, i);
        show(system.time(mh <- MetropolisHastingsVI(N, nu, pCand, rCand, x_0, i,
              debug=FALSE), gcFirst=TRUE));
        vars <- append(vars, var(mh$values));
        varvars <- append(varvars, var(vars));
}

#write.table(matrix(c(MHn, vars),nc=2), "Metropolis-Hastings-variance-exp30-vs-exp25
    .data", row.names=FALSE, col.names=FALSE);
write.table(matrix(c(MHn, vars, varvars),nc=3), "Metropolis-Hastings-variance-nu-vs-
    proposal-3.data", row.names=FALSE, col.names=FALSE);
```

B.2 Source code: copula simulations

B.2.1 Copula-functions.R

```
###This file contains all copula-specific methods, including      ###
###direct implementations, inverse functions to estimate empirical###
###copulas, and any other functions necessary to associate        ###
###dependence in our marginals.                                   ###

library(fBasics);
options(digits = 22);

#Independence copula
copula.P <- function(u,v){
        return(u*v);
}

#Complete copsitive dependence, only the first one can be a vector
copula.M <- function(u,v){
        return(min(u,v));
}

#Complete negative dependence
copula.W <- function(u,v){
        return(max(u+v-1,0));
}

#Clayton copula
copula.clayton <- function(u,v, theta){

        v <- u^(-theta)+v^(-theta)-1;
        values <- apply(array(v),1,function(x) max(x,0));

        return(values^(-1/theta));

        #return(max((u^(-theta)+v^(-theta)-1),0)^(-1/theta));
}

clayton.gen <- function(u, d){
        #u is the first value,
        #y is the uniform value.

        y <- runif(length(u));

        v <- (u^d/(y^(-d/(d+1)) + u^d -1))^(1/d)

        return(v);
}

frank.gen <- function(u1, d){
        n <- length(u1);

        V1 <- runif(n);

        X <- log(V1)/log(  ( exp(-d*u1)-1 )/( exp(-d)-1 )  )

        V2 <- runif(n)

        u2 <- (-1/d)*log(  1 + exp( log(V2)/X )*( exp(-d) -1 )  )

        return(u2);
}

gumbel.gen <- function(u1, d){
        n <- length(u1);
```

```
        V1 <- runif(n);

        X <- (-log(V1))/(-log(u1))^d

        V2 <- runif(n)

        u2 <- exp(-(-log(V2)/X)^(1/d));

        return(u2);
}

depNIGLevyCopula <- function(jSizes, p){
        show("--Copula Parameters---");
        show(c("Alpha",p$alpha));
        show(c("Beta",p$beta));
        show(c("Delta",p$delta));
        show(c("Mu",p$mu));
        show(c("AlphaDep",p$alphaDep));
        show(c("BetaDep",p$betaDep));
        show(c("DeltaDep",p$deltaDep));
        show(c("MuDep",p$muDep));
        show(c("stepDep", p$step));
        show(c("Copula method", p$copulaMtd));
        show(c("Theta", p$theta))

        lowEpsilon <- -1*p$epsilon;

        nuNIG <- function(y){
                return( nu_nig(y, p$alpha, p$beta, p$mu, p$delta ));
        }

        totNegMass <- integrate(nuNIG, lower=-Inf, upper=lowEpsilon, rel.tol
              =2.220446e-12)$value;


        pNu <- function(x){

                if(x <= 0){
                        limLow <- x;

                        if(x > lowEpsilon) x <- lowEpsilon;

                        if(x == lowEpsilon){
                                v <- totNegMass;
                        }else{
                                v <- integrate(nuNIG, lower=-Inf, upper=limLow, rel.
                                    tol=2.220446e-12)$value;
                        }

                }else{
                        #limLow <- -1*p$epsilon;
                        limUp <- x;

                        if(x < p$epsilon) x <- p$epsilon;

                        v <- totNegMass;

                        if(x > p$epsilon){
                                v <- v + integrate(nuNIG, lower=p$epsilon, upper=Inf
                                    , rel.tol=2.220446e-12)$value;
                                v <- v - integrate(nuNIG, lower=limUp, upper=Inf,
                                    rel.tol=2.220446e-12)$value;

                        }

                }

                v <- v/p$lambda;

                return(v);
        }

        hypArgs <- c(-0.5, p$alpha, p$beta, p$delta*p$step, p$mu*p$step);
        hypArgsDep <- c(-0.5, p$alphaDep, p$betaDep, p$deltaDep*p$step, p$muDep*p$
              step);

        theoreticalMin <- qghyp(0, hypArgs);
        theoreticalMax <- qghyp(1, hypArgs);
        theoreticalMinDep <- qghyp(0, hypArgsDep);
        theoreticalMaxDep <- qghyp(1, hypArgsDep);

        qNu <- function(prob, alpha, beta, delta, mu){
                maxNeg <- pNu(-1*p$epsilon);

                if(prob <= maxNeg){
                        limLow <- theoreticalMin;
                        limUp <- -1*p$epsilon;
                }else{
                        limLow <- p$epsilon
                        limUp <- theoreticalMax;
                }

                #show(c("prob:", prob));

                if(prob > pNu(10^3)) prob <- round(pNu(10^3), digits=6);
                if(prob < 10^-10) prob <- 10^-10;
```

```
                x <- uniroot(function(y) return(pNu(y) - prob), c(limLow,limUp), tol
                    =2.220446e-8)$root;

                return(x);
        }

        Fu <- apply(as.array(jSizes), 1, pNu);

        Fv <- do.call(p$copulaMtd, list(u=Fu, d=p$theta));

        jSizes2 <- apply(as.array(Fv), 1, qNu);

        #Sanity check, not very mathematical, but necessary:
        idxInsanePos <- (jSizes2 > 3*jSizes & jSizes >0);
        idxInsaneNeg <- (jSizes2 < 3*jSizes & jSizes <0)

        jSizes2[idxInsanePos] <- 3*jSizes[idxInsanePos];
        jSizes2[idxInsaneNeg] <- 3*jSizes[idxInsaneNeg];
        #End of sanity check.

        jDepSizes <- list(X=jSizes, Y=jSizes2, Fu=Fu, Fv=Fv);

        return(jDepSizes);
}

copula.empir <- function(X, Y){
        sortedX <- sort(X);
        sortedY <- sort(Y);

        nX <- length(X);
        nY <- length(Y);

        if(nX != nY){
                show("Error! Length of X and Y not the same!");
        }

        #Samples:
        s <- matrix(c(X,Y), nc=2);

        #But it actualy needs to be square:
        C <- matrix(NA, ncol=nY, nrow=nX);

        for(i in 1:length(X)){
                for(j in 1:length(Y)){

                        C[i,j] <- length(s[(s[,1] <= sortedX[i])*(s[,2] <= sortedY[j
                            ])])/nX;
                }
        }

        v <- list(X=X, Y=Y, C=C, nX=nX, nY=nY);

        return(v);
}
```

B.2.2 Empirical-copula.R

```
#Emirical-Copula.R

source("NIG-Levy2.R");
source("Copula-functions.R");
library("fCopulae");

#Common variables
T <- 1;
n <- T*10000;
step <- T/(n-1);
t <- seq(0,T,step);

#TODO: Approximation arguments by list:
approxArgs <- list(alpha = 1,
                   beta = 0,
                   delta = 1,
                   mu = 0,
                   T = T,
                   t = t,
                   n = n,
                   theta = 16,
                   rho = 0.8,
                   dep = TRUE,
                   alphaDep = 1,
                   betaDep = 0,
                   deltaDep = 1,
                   muDep = 0,
                   step = step,
                   MHn = 500,
                   maxErr = 0.01,
                   omegaOverride = FALSE,
                   verbose=TRUE,
                   ARmethod="MetropolisHastingsVI",
                   rProposal="rQuadFn2",
                   dProposal="pQuadFn",
                   copulaMtd="clayton.gen");

rhos = c(0, 0.5, 1);
```

```
thetas = c(2, 8, 16, 100);

appr <- rep(NA, length(rhos));
empirC <- rep(NA, length(rhos)*length(thetas));

for(j in 1:length(thetas)){
        for(i in 1:length(rhos)){
                approxArgs$rho <- rhos[i];
                approxArgs$theta <- thetas[j];

                appr <- approxNuNiG(approxArgs);

                Xincr <- appr$X[2:length(appr$X)]-appr$X[1:(length(appr$X)-1)];
                Yincr <- appr$Y[2:length(appr$Y)]-appr$Y[1:(length(appr$Y)-1)];

                pX <- pnig(Xincr, alpha=approxArgs$alpha, beta=approxArgs$beta,
                      delta=approxArgs$delta*approxArgs$step, mu=approxArgs$mu*
                      approxArgs$step);
                pY <- pnig(Yincr, alpha=approxArgs$alphaDep, beta=approxArgs$betaDep
                      , delta=approxArgs$deltaDep*approxArgs$step, mu=approxArgs$
                      muDep*approxArgs$step);

                empirC <- pempiricalCopula(pX, pY);
                data <- matrix(c(Xincr, Yincr, pX, pY), nc=4);

                fileC <- paste(c("Empirical-copula-empirC-theta",approxArgs$theta,"-
                      rho", approxArgs$rho,"-x2.data"), collapse="");

                file <- paste(c("Empirical-copula-theta",approxArgs$theta,"-rho",
                      approxArgs$rho,"x2.data"), collapse="");

                write.table(data, file, row.names=FALSE, col.names=FALSE)
                write.table(empirC$z, fileC, row.names=FALSE, col.names=FALSE)


                show(c("Biggest increment:", max(Xincr)));
        }
}
```

B.3 Source code: option pricing

B.3.1 Option-pricing-basket.R

```
rm(XY);

XY <- read.table("clusterSampler-XY-copula-gumbel.gen-theta20-rho0-T1-123.data");


XY <- read.table("clusterSampler-XY-gumbel.gen-theta10-rho0-T1-ALL.data");
XY <- read.table("clusterSampler-XY-gumbel.gen-theta20-rho0-T1-ALL.data");

XY <- read.table("XY20.data");

Xall <- XY[,1]
Yall <- XY[,2]


XallNIG <- read.table("clusterSampler-X-realNIG.1.data");
YallNIG <- read.table("clusterSampler-Y-realNIG.1.data");


n <- length(X);

X <- c(Xall[1:2000], Xall[2501:5000])
Y <- c(Yall[1:2000], Yall[2501:5000])

#X <- Xall;
#Y <- Yall;

expX <- exp(Xall);
expY <- exp(Yall);

expXNIG <- exp(XallNIG);
expYNIG <- exp(YallNIG);

expX <- exp(Xall);
expY <- exp(Yall);


val <- c();
valVar <- c();

K <- 10;

basketApprox <- apply(as.matrix(exp(Xall) + exp(Yall) -K), 1, function(x) max(x,0));
sum(basketApprox)/length(basketApprox)*exp(-1.05*1/36)

sum(basketNIG)/length(basketNIG)*exp(-1.05*1/36)
basketNIG <- apply(t(as.matrix(exp(XallNIG) + exp(YallNIG) -K)), 1, function(x) max(
      x,0));
```

```
for(i in seq(10,length(expX), 10)){
        n <- i;
        basket <- apply(t(as.matrix(expX[1:n] + expY[1:n] -K)), 1, function(x) max(x
            ,0));
        val <- append(val, sum(basket)/n);
        valVar <- append(valVar,var(val));
}

        basket <- apply(t(as.matrix(expX[1:n] + expY[1:n] -K)), 1, function(x) max(x
            ,0));

par(mfrow=c(2,1))

plot(1:length(val), val, 'l', xlab="Path simulations", ylab="Price")
plot(1:length(valVar), valVar, 'l', xlab="Path simulations", ylab="Variance")
```

B.3.2 clusterSampler.R

```
#The cluster sampler!!
source("NIG-Levy2.R");
library("snow");

#Common variables
T <- 1;
n <- T*10000;
step <- T/(n-1);
t <- seq(0,T,step);

#TODO: Approximation arguments by list:
approxArgs <- list(alpha = 1,
                   beta = 0,
                   delta = 1,
                   mu = 0,
                   T = T,
                   t = t,
                   n = n,
                   theta = 20,
                   rho = 0,
                   dep = TRUE,
                   alphaDep = 1,
                   betaDep = -0.1,
                   deltaDep = 1,
                   muDep = 0,
                   step = step,
                   MHn = 1000,
                   maxErr = 0.01,
                   omegaOverride = FALSE,
                   verbose=FALSE,
                   ARmethod="MetropolisHastingsVI",
                   rProposal="rQuadFn2",
                   dProposal="pQuadFn",
                   #copulaMtd="clayton.gen");
                   copulaMtd="gumbel.gen");
                   #copulaMtd="frank.gen");

nigArgs <- list(alpha = 1,
                beta = 0,
                delta = 1*step,
                mu = 0*step,
                n = n,
                verbose=FALSE);

nigArgs2 <- list(alpha = 1,
                   beta = -0.1,
                   delta = 1*step,
                   mu = 0*step,
                   n = n,
                   verbose=FALSE);

numNodes <- 15;
numSamples <- 450;

runCluster <- function(r){
        #Start a cluster with 'numNodes' nodes:
        c <- makeCluster(numNodes, type="SOCK");

        #Load all libraries at all nodes:
        clusterEvalQ(c, library("fBasics"));
        clusterEvalQ(c, library("fCopulae"));
        clusterEvalQ(c, library("HyperbolicDist"));
        clusterEvalQ(c, source("NIG-Levy2.R"));

        #Run the batches distributed over the nodes:

        #Since each cluster gives one realisation of a path, run a multiple of
            numNodes.

        max <- (numSamples/numNodes);
        dataX <- array(dim=c(numSamples, n));
        dataY <- array(dim=c(numSamples, n));

        show("Running cluster sampling with arguments:");
        show(c("T",T));
        show(c("Rho", approxArgs$rho));
```

```
        show(c("Theta", approxArgs$theta));

        begin <- timestamp()
        for(i in 1:max){
                show(paste(c("Batch no. ",i,"/", max), collapse=""));

                results <- clusterCall(c, approxNuNiG, approxArgs);

                #results <- clusterCall(c, nigLevy, nigArgs);
                #results2 <- clusterCall(c, nigLevy, nigArgs2);

                for(j in 1:length(results)){
                        idx <- j + (i-1)*numNodes;
                        dataX[idx,] <- as.array(results[[j]]$X);
                        dataY[idx,] <- as.array(results[[j]]$Y);

                        #dataX[idx,] <- as.array(results[[j]]$X);
                        #dataY[idx,] <- as.array(results2[[j]]$X);
                }
        }

        fileX <- paste(c("clusterSampler-X-copula-",approxArgs$copulaMtd,"-theta",
             approxArgs$theta,"-rho", approxArgs$rho,"-T",T,".",r,".data"), collapse
             ="");
        fileY <- paste(c("clusterSampler-Y-copula-",approxArgs$copulaMtd,"-theta",
             approxArgs$theta,"-rho", approxArgs$rho,"-T",T,".",r,".data"), collapse
             ="");

        write.table(t(dataX[,10000]), fileX, col.names=FALSE, row.names=FALSE);
        write.table(t(dataY[,10000]), fileY, col.names=FALSE, row.names=FALSE);


        #Comment in for real NIG-Levy.
        #fileX <- paste(c("clusterSampler-X-realNIG.",r,".data"), collapse="");
        #fileY <- paste(c("clusterSampler-Y-realNIG.",r,".data"), collapse="");

        #write.table(t(dataX[,10000]), fileX, col.names=FALSE, row.names=FALSE);
        #write.table(t(dataY[,10000]), fileY, col.names=FALSE, row.names=FALSE);

        stopCluster(c);

        show(begin)
        timestamp()
}

several <- function(){
        ext <- c(1,2,3,4,5,6,7,8,9,10);

        for(r in ext){
                show(c("batch:",r));
                runCluster(r);
        }
}
```

List of Figures

Bibliography

[1] Milton Abramowitz and Irene A. Stegun, *Handbook of mathematical functions with formulas, graphs, and mathematical tables*, 10th printing, with corrections ed., Dover, Washington, D.C. : U.S. Dept. of Commerce U.S. G.P.O., 1972.

[2] Albrecher and Predota, *On asian option pricing for nig lévy processes*, Journal of Computational and Applied Mathematics **172** (2004), 153–168.

[3] David Applebaum, *Lévy processes and stochastic calculus*, The Press Syndicate of the University of Cambridge, The Pitt Building, Trumpington Street, Cambridge, United Kingdom, 2004.

[4] Ole E. Barndorff-Nielsen, *Normal inverse gaussian processes and the modelling of stock returns*, Research Reports (1995), no. 300.

[5] ———, *Processess of the normal inverse gaussian type*, Finance and Stochastics (1998), no. 2, 41–68.

[6] Marian Bubak, Geert D. van Albada, Peter M.A. Sloot, and Jack J. Dongarra, *Computational science - iccs 2004: 4th international conference, kraków, poland, june 6-9, 2004, proceedings, part ii (lecture notes in computer science)*, Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2004.

[7] Cherny, *No-arbitrage and completeness for the linear and exponential models based on lévy processes*, Tech. report, Moscow State University, 2001.

[8] Siddhartha Chib and Edward Greenberg, *Understanding the metropolis-hastings algorithm*, The American Statistician **49** (1995), no. 4, 327–335.

[9] Rama Cont and Peter Tankov, *Financial modelling with jump processes*, CRC Press LCC, 2000 N.W. Corporate Blvd., Boca Raton, Florida 33431, United States, 2004.

[10] Frees and Valdez, *Understanding relationships using copulas*, North American Actuarial Journal **1(2)** (1997), 1–25.

[11] Elias S.W. Shui Hans U. Gerber, *Option pricing by esscher transforms*, Transactions of Society of Actuaries **46** (1994), 99–135.

[12] Hubalek and Sgarra, *Esscher transforms and the minimal entropy martingale measure for exponential lévy models*, Quantitative finance **6** (2006), 125–145.

[13] Roger B. Nelsen, *An introduction to copulas, 2nd edition*, Springer Science+Business Media, New York, Inc., Springer Street, NY, USA, 2006.

[14] Antonis Papapantoleon, *An introduction to levy processes with applications in finance*, 2008.

[15] Raftery and Lewis, *Comment: One long run with diagnostics: Implementation strategies for markov chain monte carlo*, Statistical Science **7** (1992), no. 4, 493–497.

[16] ———, *How many iterations in the gibbs sampler?*, Bayesian Statistics **4** (1992), 763–773.

[17] S. Raible, *Lévy processes in finance: Theory, numerics, and empirical facts*, Ph.D. thesis, Albert-Ludwigs-Universität Freiburg i, Br., Germany, 2000.

[18] S. Rasmus, *Derivative prices for models using lévy processes and markov switching*, Ph.D. thesis, Lund University, 2006.

[19] C. P. Robert and G. Casella, *Monte carlo statistical methods*, Springer Verlag New York, Inc, 1999.

[20] J. Rosinkski S. Asmussen, *Approximations of small jumps of levy processes with view towards simulation*, Journal of Applied Probability **38** (2001), no. 2, 482–493.

[21] Neil Shephard, *Estimation of an asymmetric model of asset prices*, Journal of Business and Economic Statistics (1996), no. 14, 429–434.