OPTION PRICING, HEDGING AND SIMULATION WITH GPU UNDER MULTIDIMENSIONAL LÉVY PROCESSES

CHEN DACHENG

(B.Sci.(Hons), NUS)

A THESIS SUBMITTED

FOR THE DEGREE OF MASTER OF SCIENCE

DEPARTMENT OF MATHEMATICS

NATIONAL UNIVERSITY OF SINGPAORE

2012

DECLARATION

I hereby declare that the thesis is my original work and it has been written by me in its entirety. I have duly acknowledged all the sources of information which have been used in the thesis.

This thesis has also not been submitted for any degree in any university previously.

Chen Dacheng

01 August 2012

Acknowledgements

I would like to acknowledge the help from Prof. Peter Tankov and the support from Prof. Rongfeng SUN who is the supervisor of this thesis.

Abstract

In this project, we review some basic concepts and methods of the transformation method for the calculation of derivatives' prices. We modify some of the methods to solve some pricing problems under multivariate Lévy model. Then we proceed to review some mean variance strategies to hedge our risk under multivariate Lévy model. To verify the result of the transformation method, we also conducted Monte Carlo simulation for the multivariate Lévy process upon which our model is built.

Recently, there have been great developments on the massive parallel computing with computer graphic card. We apply this new technology to our project to do Monte Carlo simulation. We also give a side by side comparison of the result between this GPU(Graphic Processing Unit) parallel computing and the C++ implementation of the same algorithm calculated sequentially by CPU.

Contents

Chapter 1

Introduction

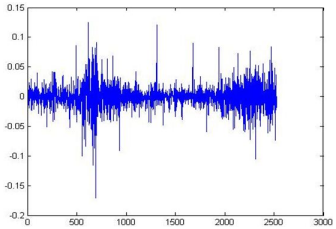
It has been almost 40 years since the first appearance of the Black-Scholes's paper "The Pricing of Options and Corporate Liabilities". During this 40 years' time, many similar models based on Brownian motion have been developed, perfected and widely used in the financial industry. Despite its popularity among academics and practitioners, many facts in the market showed that this model is flawed.

The actual security prices have jumps whereas Brownian motions do not. The distribution of the log-return[1]shows that the empirical data has heavy tails [2] whereas it is difficult to represent the heavy tails in the diffusion models(See figure 1.1). Some may argue that nowadays people do not usually use this model to price a certain option but to use this model to give implied volatility.[3] Rebonato[21] described this as "wrong number which, plugged into the wrong formula, gives the right answer." But in comparison with the empirical facts, there are some defects. In Figure 1.2, the z-axis represents the implied volatility. The surface showed its relationship with moneyness and time to maturity. At a given time to maturity, we can have a curve on the plane Moneyness-Implied volatility. From the surface we can see, as time to maturity becomes bigger and bigger, the plane

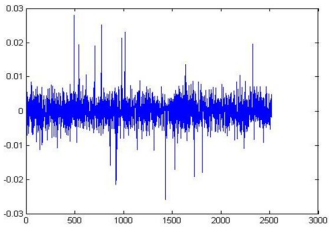
[1]The log-return is defined as: $r_{\log} = \ln(\frac{S_f}{S_i})$ where S_i and S_f are initial and ending prices of the equity respectively

[2]It is sometimes called leptokurtosis in academic literature, it is the fact that: $Kurtosis - 3 := \frac{\mathbb{E}(x-\mu)^4}{\sigma^4} - 3 > 0$

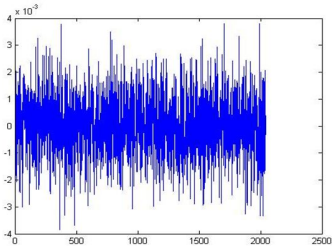
[3]the Black-Scholes value of an option is a strictly increasing function of volatility, with inversion [7] we can find the implied volatility by a particular market price.



(a) Microsoft Corporation



(b) Black-Scholes model



(c) Kou's model

Figure 1.1: Time series of log return and its simulations with same annualized return and volatility

is becoming more and more flat and the curve just mentioned becomes more and more flat too. The convexity of the curve comes from the fear of jumps but this surface is telling us that the convexity is becoming smaller and smaller as time to maturity grows. This decrease in convexity contradicts the omnipresent nature of the skews.

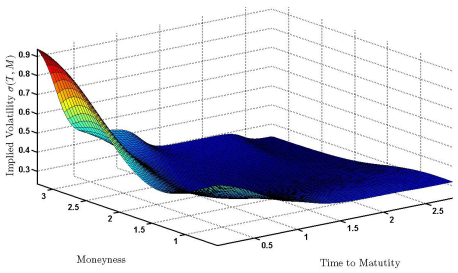


Figure 1.2: Implied volatility of DAX index option

In light of these problems, a new framework is needed. Here comes the jump

models based on Lévy process. Actually, Lévy models appeared in finance fairly early. As early as early 60s, α-stable Lévy process was proposed to model cotton prices. During the last several decades, many researchers have contributed to its development. There were jump diffusion models like Merton's model with Gaussian jumps, Kou's model with double exponential jumps; there were Brownian subordination models like Variance Gamma and Normal Inverse Gaussian models. Under many circumstances there are no closed form equations for option prices in these jumping models. Even if some models happen to have some nice properties like the exponential distribution's memoryless properties for Kou's double exponential model, the resulting solution is too complicated to read. During the 90s, some researchers introduced the Fourier transform method and not long after, Madan[4] applied Fast Fourier Transform (FFT) to it. With these developments, the theoretical models become more useful in practice.

Recently, in the global financial market, especially in the big mutual funds, hybrid products have become more and more popular. These hybrid products essentially combine various different simple products to satisfy the return expectations and risk constraints of customers. To manage the risk of these products we need to study multivariate Lévy processes. Though much research has been done on the pricing of contingent claims based on single asset, it is much more difficult to price derivatives on multiple assets[1]. There are several approaches to tackle this problem: the simpler one is the Monte Carlo method whose basic idea can be found in [5]; then there is resolution of Partial Integro-Differential Equation (PIDE) by Reich[22] and its numerical implementation in 2 dimensions by Winter[23]. But this method is very hard to reach higher dimensions, because the number of the mesh grid points will simply grow exponentially as dimension grows.

Apart from pricing, the study of hedging is also very important. Good hedging strategy will protect the writer (issuer) of a financial contract from the risk of the market. Delta hedging strategy is the strategy most applied in the financial

[1]It is called the "Curse of Dimensionality" in some literature.

market. Delta(Δ) is the Greek letter used to denote the sensitivity[1]. But under jumping processes, the situation will not be the same. A position cannot be perfectly hedged. The hedging problem thus becomes a optimization problem. Here we focused on the locally risk minimization strategy. Though this one is popular, this strategy is by no means the only one. In Chapter 4 we are going to discuss this problem.

Though theoretical development is important, sometimes developments in other areas can open other doors to the very same problem. Recently, there have been some great developments in the massive parallel computation with graphic card. This new technology is leading the scientific computation to a whole new era. Currently, we can find its application in bioinformatics, geographical data processing, physics, seismic simulation, etc. These problem has at least one point in common, they need to process huge quantity of data. The computation with graphic card gives a very good solution to this data parallel problem. Monte Carlo simulation bears similar traits to those problems described. In this project we will apply this technology to do Monte Carlo Simulation and compare it with a C++ implementation.

In this project, Chapter 2 will introduce the basic concepts and definitions of Lévy jumping processes which serve as foundations for later chapters. Chapter 3 will discuss the pricing problem under multivariate Lévy model. In this chapter, we borrow the idea of Hurd[10] to calculate the transform of basket option and extend the formula to variable weight rather than fixed equal weight for each asset. Furthermore, we also correct the published result by Luciano[16] before we finally apply it to the calculation at the end of this chapter. Chapter 4 will discuss mainly local risk minimization hedging strategy under multivariate Lévy model and Chapter 5 will firstly introduce simulation method. Then, this method will be implemented with C++, Matlab and GPU parallel computing method respectively to see their comparisons. Computer programs and some heavy calculations can be found in the Appendix at the back.

[1] So sometimes it is also called sensitivity variable.

Chapter 2

Lévy Process and Non-Arbitrage Pricing

In this chapter we review some basic concepts and definitions of Lévy processes which lay the foundation for later chapters. In the first part, we will see how Lévy processes are defined and some of its properties. The second part will show some concrete and commonly used examples of the process. The third part will define the exponential Lévy model which will be the model we use later on. The last part will show some foundations about Non-Arbitrage pricing.

2.1 Basic Definitions

Definition 2.1.1 (Lévy Process) *A $\mathbb{R}^d$ valued cadlag [1] stochastic process $(X_t)_{t\geq 0}$ on $(\Omega, \mathcal{F}, \mathbb{P})$ is a Lévy process if $X_0 = 0$, $(X_t)_{t\geq 0}$ has independent and stationary increments*

Remark 2.1.2 *Independent increments means given a sequence of time $t_0, .., t_n$, the random variables $X_{t_0}, X_{t_1} - X_{t_0}, ..., X_{t_n} - X_{t_{n-1}}$ are independent; stationary increments means the law of $X_{t+h} - X_t$ depends only on h.*

[1] It is the abbreviation of French "continue à droite, limite à gauche", meaning a function is right continuous and has left limit: $\forall t \in \mathbb{R}, f(t) = \lim_{x \uparrow t^+} f(x)$ and left limit $f(t^-) = \lim_{x \uparrow t} f(x)$ exists.

Remark 2.1.3 *Here we assume also the Lévy process is also "cadlag". This property is important for the models that we are going to use. Right continuous at t means the value is not predictable until time t and if it is left continuous, people can just have the value at t by taking a limit to it. But in real price time series, jumps are nonpredictable, so this choice is consistent with the model. On the contrary, the trading strategy should be something predictable, so under this case, we use "caglad"*[1].

Proposition 2.1.4 (Characteristic function of a Lévy process) *Given a Lévy process $(X_t)_{t\geq 0}$ on $\mathbb{R}^d$, there is a continuous function $\psi : \mathbb{R}^d \to \mathbb{R}$, such that:*

$$\mathbf{E}[e^{iz\cdot X_t}] = e^{t\psi(z)}, \quad z \in \mathbb{R}^d, \tag{2.1}$$

where the function ψ is called characteristic exponent.

Definition 2.1.5 (Lévy measure) *Given a Lévy process $(X_t)_{t\geq 0}$, the Lévy measure ν on $\mathbb{R}^d$ can be viewed as:*

$$\nu(M) = \mathbf{E}[\sharp t \in [0,1] : \Delta X_t \neq 0, \Delta X_t \in M], \quad M \in \mathcal{B}(\mathbb{R}^d). \tag{2.2}$$

Literally speaking, $\nu(M)$ is the expected number of jumps that are in M, per unit time.

Proposition 2.1.6 (Lévy Itô decomposition) *$(X_t)_{t\geq 0}$ is a Lévy process on $\mathbb{R}^d$ with Lévy measure ν which satisfies:*

$$\int_{|x|\leq 1} |x|^2 \nu(dx) < \infty; \quad \int_{|x|\geq 1} \nu(dx) < \infty.$$

There exists a vector μ and a d-dimensional Brownian motion $(B_t)_{t\geq 0}$ with covariance matrix Σ such that:

$$X_t = \mu t + B_t + X_t^l + lim_{\epsilon\downarrow 0}\tilde{X}_t^\epsilon, \tag{2.3}$$

[1] It is the abbreviation of French "continue à gauche et limite à droite", meaning a function is left continuous and has right limit: $\forall t \in \mathbb{R}, f(t) = \lim_{x\uparrow t^-} f(x)$ and right limit $f(t^+) = \lim_{x\downarrow t^+} f(x)$ exists.

where

$$X_t^l = \int_{|x|\geq 1, s\in[0,t]} x J_X(ds \times dx),$$

$$\tilde{X}_t^\epsilon = \int_{\epsilon\leq|x|<1, s\in[0,t]} x J_X(ds \times dx) - \nu(dx)ds.$$

J_X is the jump measure of X, it is a Poisson measure on $[0,\infty[\times\mathbb{R}^d$ with intensity measure $\nu(dx)dt$.

With this decomposition, every Lévy process can be characterized by a triplet (μ, Σ, ν), which is commonly known as the characteristic triplet.

Theorem 2.1.7 (Lévy-Khinchin representation and characteristic exponent) *Given a Lévy process $(X_t)_{t\geq 0}$ defined on $\mathbb{R}^d$ with characteristic triplet (μ, Σ, ν), we have*

$$\mathbf{E}[e^{iz\cdot X_t}] = e^{t\psi(z)}, \quad z \in \mathbf{R}^d \tag{2.4}$$

with

$$\psi(z) = -\frac{1}{2} z\cdot\Sigma z + i\mu\cdot z + \int_{\mathbb{R}^d} (e^{iz\cdot x} - 1 - iz\cdot x 1_{|x|\leq 1})\nu(dx). \tag{2.5}$$

The function here is called the characteristic exponent. We will denote the characteristic function by Φ. The characteristic function of process X_t at time t is just the expectation in the formula (2.4).

2.2 Some important Lévy processes

2.2.1 Jump Diffusion Models

A jump diffusion type Lévy process has the following form:

$$X_t = \mu t + \sigma W_t + \sum_{i=1}^{N_t} Y_i, \tag{2.6}$$

where $(N_t)_{t\geq 0}$ is the number of jumps, counted by a Poisson process, Y_i (i.i.d. variables) are jump sizes specified by the distribution ν_0.

Example 2.2.1 (Kou Model) *The Kou Model combined two exponential distributions for upward and downward jumps. The jump size distribution is as follows:*

$$\nu_0(dx) = [p\lambda_+ e^{-\lambda_+ x} 1_{x>0} + (1-p)\lambda_- e^{-\lambda_- |x|} 1_{x<0}]dx. \tag{2.7}$$

The Lévy measure is $\nu = \lambda\nu_0$, where λ is the jump intensity. The characteristic exponent is:

$$\psi(u) = -\frac{\sigma^2 u^2}{2} + i\mu u + iu\lambda \frac{p}{\lambda_+ - iu} - \frac{1-p}{\lambda_- + iu} \tag{2.8}$$

Another famous example of Jump Diffusion Models is Merton model. For this model the jump distribution is just normal distribution $N(\mu, \sigma^2)$. The characteristic exponent is

$$\psi(u) = \frac{\sigma^2 u^2}{2} + i\gamma u + \lambda(e^{-\delta^2 u^2/2 + i\mu u} - 1). \tag{2.9}$$

Kou's model is good in its direct representation of upward and downward jumps by a double exponential distribution. By doing this, it is possible to adjust the probability and intensity of upward and downward jumps. It is better than Merton's model in that it needs not to be symmetrical. We can have either

upward or downward jump more probable or more intense.

2.2.2 Subordination Models

Subordination simply speaking is to use one process to "time change" another process. The former is called subordinator whose trajectory should be almost surely increasing[1]. Evidently, time goes forward. The later is the process time changed. In the financial modeling, Brownian motion is usually used as the process time changed and several Lévy processes are used as subordinator. The logic behind is that instead of having the information arrives at constant rate, we can have the information arrives faster or slower when time 'accelerated' or 'decelerated'. As described above, the general form of these processes is $X_t = \mu S_t + \sigma B(S_t)$, where S_t is a subordinator. It is put at the time variable place of a Brownian motion with drift μ.

Theorem 2.2.2 *We fix a probability space $(\Omega, \mathcal{F}, \mathbb{P})$. Let $(S_t)_{t\geq 0}$ be a subordinator with Laplace exponent[2] $l(u)$ and triple $(\mu_s, 0, \nu_s)$. Let $(X_t)_{t\geq 0}$ be a Lévy process on $\mathbb{R}^d$ with triplet (μ, Σ, ν) and characteristic exponent $\psi(u)$. Then for each $\omega \in \Omega$, $Y(t,\omega) = X(S(t,\omega),\omega)$ is a Lévy process with characteristic function:*

$$\mathbf{E}[e^{iuY_t}] = e^{tl(\psi(u))}. \tag{2.10}$$

The triplet of Y is given as (μ_Y, Σ_Y, ν_Y), where

$$\Sigma_Y = \mu_s \Sigma, \tag{2.11}$$

$$\nu_y(D) = \mu_s \nu(D) + \int_0^\infty p_t^X(D)\nu_s(dt), \quad \forall D \in \mathcal{B}(\mathbb{R}^d), \tag{2.12}$$

[1] This can be guaranteed by either $X_t \geq 0$ almost surely or more practically it has only positive jumps of finite variation and positive dirft, and does not have diffusion component which can be seen as $\mu \geq 0$, $\Sigma = 0$, $\int_0^\infty (x \wedge 1)\nu(dx) < \infty$ and $\nu((\infty, 0]) = 0$.

[2] Defined similarly to characteristic exponent. Instead of doing Fourier transform, we do Laplace transform here. One can also obtain Laplace transform from characteristic function by substituting $-iu$.

$$\mu_Y = \mu_s \mu + \int_0^\infty \nu_s(dt) \int_{|x|\leq 1} x p_s^X(dx), \tag{2.13}$$

where p_t^X is the probability distribution of X_t.

Example 2.2.3 (Variance Gamma) *Variance Gamma process is a subordinated Lévy process obtained by time changing a Brownian motion with a gamma process. A gamma process $\Gamma(m,n)$ has a Lévy triplet defined as such: $\mu_s = \frac{m}{n} - \frac{me^{-n}}{m}$,$\Sigma = 0$, $\nu(dx) = m\frac{e^{-nx}}{x}dx, x > 0$.*
To construct a variance gamma process we take $(X_t)_{t>0}$ as a drifted Brownian motion defined as $X_t = \mu t + \sigma B_t$, $(S_t)_{t>0}$ as a gamma process $\Gamma(1/a, 1/a)$, $a > 0$. Then the variance gamma process is defined as

$$Y_{VG} = \mu S_t + \sigma B(S_t). \tag{2.14}$$

The characteristic function of the process Y_{VG} is:

$$\Phi_V G(u) = (1 - iu\mu a + \frac{1}{2}\sigma^2 a u^2)^{-\frac{t}{a}}. \tag{2.15}$$

Example 2.2.4 (Normal Inverse Gaussian) *The Normal Inverse Gaussian (NIG) process is defined similarly as the Variance Gamma process. Its subordinator is changed to an Inverse Gaussian process which has a Lévy measure $\nu_{IG}(x) = \frac{a}{\sqrt{ax^3}} exp(-\frac{b^2 x}{2})dx, \quad x > 0$. The characteristic function of a Normal Inverse Gaussian process with parameters $a > 0, -a < b < a, c > 0$ is:*

$$\Phi_{NIG}(z) = e^{t(-c(\sqrt{a^2-(b+iu)^2}-\sqrt{a^2-b^2}))}. \tag{2.16}$$

In this expression there is no specific parameter comes form the drifted Brownian motion. Because they are together incorporated in the parameters in the above expression. For an expression with explicit parameter from the drifted Brownian motion refer to [5] page 117. The Normal Inverse Gaussian process was first proposed in 1995 by Barndorff-Neilsen. This process gains its popularity because it fits the log *returns on German stock market data very well.*

2.3 Exponential Lévy model

Exponential Lévy model can be considered as a generalization of the Black-Scholes model. It can be achieved by simply replacing the Brownian motion with a Lévy process $(X_t)_{t\geq 0}$

$$S_t = S_0 e^{rt+X_t}. \tag{2.17}$$

According to [5], there are several advantages of using the exponential Lévy model. The closed-form characteristic function of certain Lévy processes makes the Fourier transform method possible; the Markov property of the price makes it possible to express the derivative price as a solution of Partial Integro-Differential Equations; the flexibility of being able to choose the Lévy measure makes the calibration and the implied volatility calculation possible. Sometimes the exponential Lévy model is written as exp-Lévy model.

2.4 Non-Arbitrage Pricing

Theorem 2.4.1 (Fundamental theorem of asset pricing) *The market model is defined by $(\Omega, \mathcal{F}, \mathbb{P})$. The asset prices $S_t, t \in [0,T]$ is arbitrage-free if and only if there exists a probability measure $\mathbb{Q}$ equivalent to $\mathbb{P}$ such that the discounted asset*[1] *$S'_t, t \in [0,T]$ is martingale with respect to $\mathbb{Q}$.*

Remark 2.4.2 *We sometimes use the term risk neutral measure, but this does not mean the investors are risk neutral. Rather it means that the contingent claim is priced in an arbitrage-free way. With this theorem, we translate the real world arbitrage-free situation into the matters of looking for an equivalent martingale measure that satisfies certain maximization conditions. In the Black-Scholes model, the equivalent martingale measure(EMM) is unique and is found by doing Girsanov transform which is essentially equating the drift of the process*

[1] $S'_t = e^{-rt} S_t$

to the risk neutral return like the LIBOR[1]*. However, in the jumping models, the EMM is not unique anymore and there can be infinitely many of them, so looking for an appropriate EMM is a non-trivial task.*
It was shown that under some optimisation criterion, the Esscher transform of the historic measure is optimal.

2.4.1 Esscher Transform

Esscher transform has existed for a very long time, but previously used in actuarial science. It can be used for pricing derivative contracts if the logarithms of the prices of the underlier follows Lévy process [9]. Since we are modeling the risk neutral dynamics with exponential Lévy processes. Esscher transform can be applied here.

Definition 2.4.3 (Esscher Transform) *Let X be a Lévy process with characteristic triplet (μ, σ^2, ν). Define a probability space $(\Omega, \mathcal{F}, \mathbb{P})$. We assume that the Lévy measure satisfies $\int_{|x|\geq 1} e^{\theta x}\nu(dx) < \infty$, where θ is a real number. The Esscher transform is to find an equivalent probability $\mathbb{Q}$ under which X is a Lévy process which has a characteristic triplet $(\mu_e, 0, \nu_e)$, where $\mu_e = \mu + \int_{-1}^{1} x(e^{\theta x} - 1)\nu(dx)$, $\nu_e(dx) = e^{\theta x}\nu(dx)$. The Radon-Nikodym derivative that corresponds to this measure change is*

$$\frac{d\mathbb{Q}}{d\mathbb{P}}|\mathcal{F}_t = \frac{e^{\theta X_t}}{\mathbf{E}[e^{\theta X_t}]} = exp(\theta X_t - h(\theta)t), \tag{2.18}$$

where $h(\theta) = -\psi^{\mathbb{P}}(-i\theta)$.

Remark 2.4.4 *The discounted price process of the stock is $e^{-rt}S_t$. It must be a martingale under $\mathbb{Q}$. Therefore for $t > 0$, we have*

$$S_0 = \mathbf{E}^{\mathbb{Q}}[e^{-rt}S_t] = S_0\mathbf{E}^{\mathbb{P}}[e^{(1+\theta)X_t - h(\theta)t - rt}]. \tag{2.19}$$

[1]London Interbank Offered Rate. This rate is not controlled by any government, only decided by the market.

From the above equation, we have the following relationship:

$$-t\psi^{\mathbb{P}}(-i(1+\theta)) - h(\theta,t) - rt = 0. \tag{2.20}$$

The Same procedure applies to the riskless bond dynamic $B_t = B_0 e^{rt}$. We have another relation:

$$B_0 = \mathbf{E}^{\mathbb{Q}}[B_t e^{-rt}] = \mathbf{E}^{\mathbb{Q}}[B_0 e^{rt} e^{-rt}] = B_0 \mathbf{E}^{\mathbb{P}}[e^{\theta X_t - h(\theta)t}]. \tag{2.21}$$

So

$$-t\psi^{\mathbb{P}}(-i\theta) - h(\theta,t) = 0. \tag{2.22}$$

We can solve for $h(\theta) = -\psi^{\mathbb{P}}(-i\theta)$ and substitute into the equation (2.20) we have the following relationship:

$$-r - \psi^{\mathbb{P}}(-i(1+\theta)) + \psi^{\mathbb{P}}(-i\theta) = 0. \tag{2.23}$$

If the solution for the equation (2.23) exists, the Esscher transform exists. The characteristic exponent of X under measure $\mathbb{Q}$ is given by:

$$\psi^{\mathbb{Q}}(t) = \psi^{\mathbb{P}}(t - i\theta) - \psi^{\mathbb{P}}(-i\theta). \tag{2.24}$$

Example 2.4.5 *Let us apply the above result to a Brownian Motion $X_t = \mu t + \sigma B_t$. As we know that the characteristic exponent of the Brownian Motion is $\psi^{\mathbb{P}}(t) = \frac{\sigma^2 t^2}{2} - i\mu t$. We substitute this equation into (2.23) and solve for $\theta = -\frac{\mu+\sigma^2/2-r}{\sigma^2}$. Consequently the characteristic exponent of the process under risk neutral measure $\mathbb{Q}$ is just $\psi^{\mathbb{Q}}(t) = \frac{\omega^2 t^2}{2} - it(r - \frac{\sigma^2}{2})$, which is a result can be obtained by Girsanov transform. This is not a surprise, because the risk neutral measure is unique under diffusion models. So the Esscher transform and the Girsanov transform should give the same result.*

Remark 2.4.6 *A more general result is $\psi^{\mathbb{Q}}(-i) = -r$. Similar results can also be obtained by analogously applying the Itô Formula for semi-martingales to the Exponential Lévy process and set the drift term to zero.*

2.4.2 Non-arbitrage condition in the multidimensional setting

In Remark 2.4.6 we have seen a non-arbitrage condition of one dimension case. This result has existed for many years, whereas the extension to the multidimensional setting is just recent.

Theorem 2.4.7 (Non-arbitrage in multidimensional exp-Lévy model) *Let $(X, \mathbb{P})$ be a Lévy process defined on $\mathbb{R}^d$ with triplet (μ, Σ, ν). The following statements are equivalent:*
1.There exists a probability measure $\mathbb{Q}$ equivalent to $\mathbb{P}$. $(X, \mathbb{Q})$ is a Lévy process and (X^i) is a $\mathbb{Q}$-martingale for all $1 \leq i \leq d$.
2.Denote Y to be a linear combination of X^i. Y has triplet (μ, σ^2, ν). All such Y satisfy one of the following four conditions:
▷ *2.1. $Y \equiv 0$ or $(Y, \mathbb{P})$ is not almost surly monotone,*
▷ *2.2. $\sigma > 0$,*
▷ *2.3. $\sigma = 0$ and $\int_{|x|\leq 1} |x|\nu(dx) = \infty$,*
▷ *2.4. $\sigma = 0, \int_{|x|\leq 1} |x|\nu(dx) < \infty$ and $-b$ is in the relative interior of the smallest convex cone containing the support of ν, where $b = \mu - \int_{|x|\leq 1} x\nu(dx)$ is the drift of Y.*

Chapter 3

Transformation Method for Option Pricing

In this part we are going to see how Fourier transform is used to calculate the option price. The first part will recall how the pricing formula comes from. After the first part we will see that to calculate the option price, we need two things: one is Fourier transform of payoff function which is normally calculated in closed form, the other is the characteristic exponent of the underlying processes. We have seen the one dimensional case in the previous chapter. Here we focus on the multidimensional case. In [10], Hurd and Wei proposed a method which was used to transform the payoff function of spread option. We borrow his idea to calculate the Fourier transform of basket options and we furthermore give each dimension a variable weight instead of a fixed equal weight in their original work. These payoff transforms is put in section 3.2. As for the characteristic exponent, we use the research by Luciano[16]. In his published work, he tried to use the theoretical result in [1]. This result can be viewed as a multivariate version of Theorem 2.2.2 in the previous chapter. But the published characteristic exponent of Normal Inverse Gaussian process by Luciano wrongly used the theorem. We corrected the problem in this project and used this corrected version to do the calculation. The corrected characteristic exponent and the calculation results are put together in section 3.4. The transformation results are compared with Monte Carlo simulation which is served as benchmark. We also gave a detailed

description to the implementation for multidimensional Fast Fourier Transform in section 3.3.

3.1 Formulation with Partial Integro-Differential Equation (PIDE)

Recall that in the Black-Scholes model, we have the following PDE:

$$\frac{\partial V}{\partial t}(t, S_t) + rS_t\frac{\partial V}{\partial S_t} + \frac{1}{2}\sigma_t^2 S_t^2 \frac{\partial^2 V}{\partial S_t^2}(t, S_t) - rV(t, S_t) = 0. \tag{3.1}$$

Analogously, in the jump models, there is a similar formulation in terms of Partial Integral Differential Equation [11].
Consider $V(t, S_t)$ to be the price at time t of an option, written on a vector of d underlyings $\mathbf{S}_t$. Let $\phi(\mathbf{S}_T)$ be the T-maturity payoff. In an arbitrage-free and frictionless market, the value of the option is the discounted expectation under a risk-neutral measure $\mathbb{Q}$, namely:

$$V(t, \mathbf{S}_t) = \mathbb{E}_t^{\mathbb{Q}}[e^{-r(T-t)}\phi(\mathbf{S}_T)]. \tag{3.2}$$

Now taking $\mathbf{S}_t = \mathbf{S}_0 e^{\mathbf{X}_t}$ where $\mathbf{X}_t$ is a Lévy process under risk neutral measure with characteristic triplet (μ, Σ, ν). The discount-adjusted and transformed price process:

$$v(t, \mathbf{X}_t) := e^{r(T-t)}V(t, S_t). \tag{3.3}$$

We thus have the following formulation:

$$\begin{cases} (\partial_t + \mathcal{L})v = 0 \\ v(T, x) = \phi(\mathbf{S}_0 e^x), \end{cases} \tag{3.4}$$

where $\mathcal{L}$ is the infinitesimal generator of the multi-dimensional Lévy process X

and acts on twice differentiable functions $v(x)$[1] as follows:

$$\mathcal{L}v(x) = (\mu \cdot \partial_x + \frac{1}{2}\partial_x \cdot \Sigma \partial_x)v(x) + \int_{\mathbb{R}^n/\{0\}} (v(x+y) - v(x) - y \cdot \partial_x v(x) 1_{|y|<1})\mu(dy). \tag{3.5}$$

We take the Fourier transform on both sides of the above formula, to find that:

$$\mathcal{F}[\mathcal{L}v](t,\omega) = \{i\mu \cdot \omega - \frac{1}{2}\omega \cdot \Sigma\omega + \int_{\mathbb{R}^n} (e^{i\omega y} - 1 - iy\omega 1_{|y|<1})\mu(dy)\}\mathcal{F}[v](t,\omega). \tag{3.6}$$

Recall the Lévy-Kintchine representation in (2.5), we can see that the right side of above formula is just $\Psi(\omega)\mathcal{F}[v](t,\omega)$. Consequently, (3.4) is transformed to:

$$\begin{cases} \partial_t \mathcal{F}[v](t,\omega) + \Psi(\omega)\mathcal{F}[v](t,\omega) = 0 \\ \mathcal{F}[v](T,\omega) = \mathcal{F}[\phi(S_0)e^x](\omega). \end{cases} \tag{3.7}$$

This is an ordinary differential equation and its solution is:

$$\mathcal{F}[v](t_1,\omega) = \mathcal{F}(T,\omega)e^{\psi(\omega)(T-t)}, \tag{3.8}$$

and the final result is obtained by taking an inverse transform:

$$v(t,x) = \mathcal{F}^{-1}\{\mathcal{F}[v](T,\omega)e^{\psi(\omega)(T-t)}\}(x). \tag{3.9}$$

3.2 Practical calculation of several derivative contracts

Recently, Hurd and Wei[10] proposed a method to calculate the spread option price with multivariate exponential Lévy model. The essential point of their method is to scale the payoff function with respect to strike K. Take for example the spread option's payoff function $(S_T^1 - S_T^2 - K)^+$ [2]. If we scale it with respect

[1] We assume $v(x)$ to be twice differentiable

[2] $(S_T^1 - S_T^2 - K)^+$ means $\max\{(S_T^1 - S_T^2 - K), 0\}$

to strike K, it will give $K(\frac{S_T^1}{K} - \frac{S_T^2}{K} - 1)^+$. We take one more step to transform this function into exponential form $K(e^{x_1} - e^{x_2} - 1)^+$, where $x^{1,2} = \log(\frac{S_T^{1,2}}{K})$. The calculation will be carried out with this transformed payoff function. After we have calculated the result, we have to multiply back the K to obtain the actual price. With the equation (3.9) above, we need to firstly calculate the interior Fourier transform of payoff function. This step is to get a closed form expression for the Fourier transform of payoff function. Then we combine the Fourier transform just calculated and the characteristic function which is also in closed form for certain models into one single closed form expression. The final step is to take an inverse Fourier transform on this expression. This step is carried out by FFT. From here we can see why people want to have a closed form characteristic function of a model. Because with closed form expression of characteristic function, the calculation will actually be reduced to only one numerical integration which can be done by FFT.

In the following we will extend Hurd and Wei's method on the calculation of Spread option price to two other options: rainbow option and basket option. Both of them are showed in two dimensional case. It will be simple to extend to higher dimension.

3.2.1 Rainbow Option

Rainbow option is actually a family of options: there are "Call on max" $(\max(S^1, ..., S^n) - K)^+$; "Call on min" $(\min(S^1, ..., S^n) - K)^+$; "Put on max" $(K - \max(S^1, ..., S^n))^+$ and "Put on min" $(K - \min(S^1, ..., S^n))^+$. Since the calculation are generally similar, here only the call on min is studied. As mentioned above, we firstly transform the payoff function into this form: $\phi(x) = (\min(e^{x^1}, e^{x^2}) - 1)^+$. Then, we calculate the Fourier transform of this payoff function $\hat{\phi}(u)$:

$$\hat{\phi}(u) = \int_0^\infty \int_{x_1}^\infty (e^{x^1} - 1) e^{-i(u_1 x^1 + u_2 x^2)} dx^2 dx^1 + \int_0^\infty \int_{x_2}^\infty (e^{x^2} - 1) e^{-i(u_1 x^1 + u_2 x^2)} dx^1 dx^2. \tag{3.10}$$

Since x^1, x^2 are symmetrical, we only need to evaluate one of those two double integrals on the righthand of the equation (3.10). We choose the second integration. The evaluation does not involve any contour integration, and the result is $-\frac{1}{(i(u_1+u_2)-1)(u_1+u_2)u_1}$. By symmetry, the first integral is just $-\frac{1}{(i(u_1+u_2)-1)(u_1+u_2)u_2}$. Summing up, we have

$$\hat{\phi}(u) = -\frac{1}{(i(u_1+u_2)-1)u_1u_2}. \tag{3.11}$$

Having the Fourier transform of payoff function in closed form, with the help of equation (3.9), we have

Proposition 3.2.1 *The value of rainbow option call on min is written as the following double integral:*

$$V_{call\ on\ min}(\mathbf{X}_0) = \frac{1}{(2\pi)^2} e^{-rT} \int\int_{R^2+i\epsilon} e^{iu\cdot\mathbf{X}_0} \Phi(u;T)\hat{\phi}(u)d^2u, \tag{3.12}$$

where $\Phi(u;T)$ *is the characteristic function of* $\mathbf{X}_T$, $\mathbf{X}_0 = (x_0^1, x_0^2)'$, $x_0^{1,2} = \log\frac{S_0^{1,2}}{K}$. $\epsilon = (\epsilon_1, \epsilon_2)$, $\epsilon_{1,2} < 0$.

Remark 3.2.2 *Those two* ϵ*s in the above double integral are to make the double integral finite in the actual numerical calculation these parameters are used to make imaginary part of the result go to 0. This integration on complex plane approach was proposed by [15]. Refer to his work for more details. As for this case, we can see that if we substitute* $i\epsilon$ *in the place of* u *of* $e^{iu\cdot\mathbf{X}_0}$, *we have* $e^{-\epsilon\cdot\mathbf{X}_0}$. *This means if* $\mathbf{X}_0$ *is negative, the value of this exponential term will be smaller than 1. This coincides with the real situation when the call is out of money*[1], *the price of this call is generally very small.*

3.2.2 Basket Option

The basket option's payoff defined on d assets $\mathbf{S}_t = (S_t^1, ..., S_t^d)$ is usually defined as $(\omega_1 S_t^1 +, ..., +\omega_d S_t^d - K)^+$, where ω_i are weights of each underlying asset. Here we study the two assets case: $(\omega_1 S_t^1 + \omega_2 S_t^2 - K)^+$. First we consider the relation

[1] $S_0 < K$, in this situation, the $\log\frac{S_0}{K} < 0$.

$(\omega_1 S_t^1 + \omega_2 S_t^2 - K)^+ - (K - \omega_1 S_t^1 - \omega_2 S_t^2) = \omega_1 S_t^1 + \omega_2 S_t^2 - K$. Following the same procedure of scaling as before we have the relationship

$$(\omega_1 e^{x^1} + \omega_2 e^{x^2} - 1)^+ = \omega_1 e^{x^1} + \omega_2 e^{x^2} - 1 + (1 - \omega_1 e^{x^1} - \omega_2 e^{x^2})^+, \tag{3.13}$$

where $x^{1,2} = \log \frac{S_t^{1,2}}{K}$. We will explain the reason for applying this relationship later after the calculation.

We take the expectation under risk neutral measure $\mathbb{Q}$. The left hand side of equation (3.13) gives us the price of the basket option:

$$\begin{aligned} V_{basket}(\mathbf{X}_0) &= e^{-rT}\mathbf{E}^{\mathbb{Q}}[(\omega_1 e^{x^1} + \omega_2 e^{x^2} - 1)^+] \\ &= e^{-rT}\mathbf{E}^{\mathbb{Q}}[\omega_1 e^{x^1} + \omega_2 e^{x^2} - 1] + e^{-rT}\mathbf{E}^{\mathbb{Q}}[(1 - \omega_1 e^{x^1} - \omega_2 e^{x^2})^+] \\ &= \frac{\omega_1 S_0^1}{K} + \frac{\omega_2 S_0^2}{K} - e^{-rT} + e^{-rT}\mathbf{E}^{\mathbb{Q}}[(1 - \omega_1 e^{x^1} - \omega_2 e^{x^2})^+] \\ &= \frac{\omega_1 S_0^1}{K} + \frac{\omega_2 S_0^2}{K} - e^{-rT} + \frac{e^{-rT}}{(2\pi)^2} \int\int_{\mathbb{R}^2 + i\epsilon} e^{iu\cdot\mathbf{X}_0} \Phi(u;T)\hat{\phi}(u) d^2u. \end{aligned} \tag{3.14}$$

As we can see in (3.14), we need to evaluate $\hat{\phi}$ to get an explicit expression. Before we start let's recall the definition of Beta function:

$$B(p+1, q+1) = \int_0^1 x^p (1-x)^q dx, \tag{3.15}$$

$$B(x, y) = \frac{\Gamma(x)\Gamma(y)}{\Gamma(x+y)}. \tag{3.16}$$

We have the following calculation:

$$\begin{aligned}\hat{\phi}(u) &= \int_{-\infty}^{-\log\omega_1} e^{-iu_1x^1}\int_{-\infty}^{\log(1-\omega_1e^{x^1})} e^{-iu_2x^2}(1-\omega_1e^{x^1}-\omega_2e^{x^2})dx^2dx^1 \\ &= \frac{1-iu_2+\omega_2iu_2}{(-iu_2)(1-iu_2)}\int_{-\infty}^{-\log\omega_1} e^{-iu_1x^1}(1-\omega_1e^{x^1})^{1-iu_2}dx^1 \\ &= \omega_1^{iu_1}\frac{1-iu_2+\omega_2iu_2}{(-iu_2)(1-iu_2)}\int_0^1 t^{-1-iu_1}(1-t)^{1-iu_2}dt,\ \ \omega_1e^{x^1}=t\end{aligned} \tag{3.17}$$

Applying the equation (3.15) and (3.16) to the last line of (3.17), the following result is obtained:

$$\hat{\phi}(u) = (1-(\omega_2-1)iu_2)\omega_1^{iu_1}\frac{\Gamma(-iu_2)\Gamma(-iu_1)}{\Gamma(2-iu_1-iu_2)}. \tag{3.18}$$

Now it is easy to understand that the reason we use the relationship (3.13) is to construct the form of Beta integral. Summing up we have a similar proposition for basket option as follows:

Proposition 3.2.3 *The valued of basket option Call is written in the following form:*

$$V_{basket} = \frac{\omega_1S_0^1}{K}+\frac{\omega_2S_0^2}{K}-e^{-rT}+\frac{e^{-rT}}{(2\pi)^2}\int\int_{\mathbb{R}^2+i\epsilon} e^{iu\cdot\mathbf{x}_0}\Phi(u;T)\hat{\phi}(u)d^2u, \tag{3.19}$$

where $\hat{\phi}(u) = (1-(\omega_2-1)iu_2)\omega_1^{iu_1}\frac{\Gamma(-iu_2)\Gamma(-iu_1)}{\Gamma(2-iu_1-iu_2)}$, $\Phi(u;T)$ *is the characteristic function of* $\mathbf{X}_T$, $\mathbf{X}_0=(x_0^1,x_0^2)'$, $x_0^{1,2}=\log\frac{S_0^{1,2}}{K}$, $\epsilon=(\epsilon_1,\epsilon_2),\epsilon_{1,2}>0$.[1]

Remark 3.2.4 *This calculation of basket option price can be extended to higher*

[1]Because this time, the integration is actually calculating a Put which is the inverse of the setting in the Rainbow option calculated above.

dimensions and the Fourier transform of ϕ is as follows:

$$\hat{\phi}(u) = (1 - iu_n + i\omega_n u_n)\frac{\prod_{k=1}^{n}\Gamma(-iu_k)}{\Gamma(2 - i\sum_{k=1}^{n} u_k)}\prod_{k=1}^{n-1}\omega_k^{iu_k}. \tag{3.20}$$

The exact calculation which is a bit long can be found in Appendix A.

3.3 Fast Fourier Transform

In the last section we have explained the way to calculate the option price with Fourier transform. This Part is basically an explanation of the numerical implementation of the FFT.

3.3.1 Definition of FFT

Definition 3.3.1 *$X(k) = fft(x(j))$ and $x(j) = ifft(X(k))$ implement the transform and inverse transform pair given for vectors of length N by:*

$$X(k) = \sum_{j=1}^{N} x(j)\omega_N^{(j-1)(k-1)}, \tag{3.21}$$

$$x(j) = \frac{1}{N}\sum_{k=1}^{N} X(k)\omega_N^{-(j-1)(k-1)}, \tag{3.22}$$

where $\omega_N = e^{-\frac{2\pi i}{N}}$.

The $fft2(ifft2)$ is just applying the $fft(ifft)$ on each column of the input N by N matrix, the result is still an N by N matrix.

3.3.2 Discretization

Since the expressions of the rainbow option price and basket option price in the previous chapters are similar, the FFTs are carried out in the similar way. So

we only present a two dimensional example for the spread option. The double integral in Proposition 3.2.1 can be estimated by a double sum over the lattice:

$$\{l(k) = (l(k_1), l(k_2)) = (-N\eta/2 + k_1\eta, -N\eta/2 + k_2\eta) | k_1, k_2 = 0, 1..., N-1\}. \tag{3.23}$$

Then we construct a reciprocal lattice with spacing $\eta^* = 2\pi/N\eta$ and choose $X_0 = \log S_0$ on this lattice. With this we have transformed the space of log returns to the space of initial log prices.
The new lattice is thus:

$$\{l'(l) = (l'(l_1), l'(l_2)) = (-N\eta^*/2 + n_1\eta^*/2, -N\eta^*/2 + n_2\eta^*) | n_1, n_2 = 0, 1, ..., N-1\}. \tag{3.24}$$

So the approximation of $P(\mathbf{S}_0, T)$ is written as:

$$\frac{\eta^2 e^{-rT}}{(2\pi)^2} \sum_{k_1,k_2=0}^{N-1} e^{i(\mu(k)+i\epsilon)x'} \Phi(\mu(k) + i\epsilon; T)\hat{P}(\mu(k) + i\epsilon). \tag{3.25}$$

The above form can be expressed with $ifft2$

$$(-1)^{l_1+l_2} e^{-rT} (\frac{\eta N}{2\pi})^2 e^{-\epsilon x(l)'} [ifft2(H)](l)), \tag{3.26}$$

where

$$H(k) = (-1)^{l_1+l_2} \Phi(\mu(k) + i\epsilon; T)\hat{P}(\mu(k) + i\epsilon). \tag{3.27}$$

3.4 Application

3.4.1 A Multivariate Subordinator Model

Much study has been done on subordination models, but many models are based on one asset. Currently, structured products are becoming more and more popular, multivariate models are in need. Recently, Luciano[16] proposed a model which is very interesting to study. This is a model based on multivariate Gaus-

sian subordination. Unlike many similar models appeared before, this model has a multivariate subordinator, instead of a univariate subordinator. It is better to have a multivariate subordinator, because even if some major events in the market will affect many assets at a time, it is still not reasonable to just bundle all of them together. With multivariate subordinator, more independence for each asset will be retained.

3.4.2 Construction of the Model

The model consists of a multivariate subordinator which time changes a Brownian motion on $\mathbb{R}^d$ with independent components. The subordinator is defined as follows:

$$\mathbf{S}_t = (S_t^1, ..., S_t^d)^\intercal = (X_t^1 + \alpha_1 Z_t, ..., X_t^d + \alpha_d Z_t)^\intercal. \tag{3.28}$$

As we can see, each component of the subordinator consists of two parts: one is individual part X_t^i, the other is common part Z_t^i. With this construction, it is easy to represent the market shock by Z_t^i and assets' own characteristics by X_t^i. The time changed Brownian motion is defined as follows:

$$\mathbf{B}_t = (\mu_1 t + \sigma_1 B_t^1, ..., \mu_d t + \sigma_d B_t^d)^\intercal, \;\; \mu_i \in \mathbb{R}, \; \sigma_i \in \mathbb{R}_+. \tag{3.29}$$

The time changed Lévy process is just:

$$\mathbf{Y}_t = (\mu_1 S_t^1 + \sigma_1 B^1(S_t^1), ..., \mu_d S_t^d + \sigma_d B^d(S_t^d))^\intercal. \tag{3.30}$$

To calculate the characteristic function of the subordinated process (Y), we need theorem 2.2.2. According to the theorem, we need two things, one is the Laplace exponent of the subordinator at time one, the other is the characteristic exponent of the subordinated process at time one. To obtain the Laplace exponent we can first get the characteristic exponent. The characteristic exponent of the subordinator $\mathbf{S}$ satisfies:

$$\psi_{\mathbf{S}}(\mathbf{w}) = \sum_{i=1}^{d} \phi_{X_i}(\omega_i) + \phi_Z(\sum_{i=1}^{d} \alpha_i \omega_i), \tag{3.31}$$

So the characteristic function of the subordinator at time one is as follows:

$$\Phi_{\mathbf{S}}(\mathbf{u}) = \prod_{i=1}^{d} \phi_{X_i}(u_i)\phi_Z(\sum_{i=1}^{d} \alpha_i u_i). \tag{3.32}$$

Remark 3.4.1 *In the original paper of Luciano[16], he did not transform the characteristic exponent into Laplace exponent. Instead, he directly applied the characteristic exponent in the calculation of the characteristic function of the subordinated process at time one. As a consequence of this misapplication of the theorem, the resulting formula was originally written as follows:*

$$\begin{aligned}\phi_{\mathbf{Y}}(\mathbf{u}) = \exp\Bigg[-\sum_{k=1}^{d}(1-a\gamma_k)\left(\sqrt{-2i\left(i\beta_k\delta_k^2u_k - \frac{1}{2}\delta_k^2u_k^2\right) + \frac{b^2}{\gamma_k^2}} - \frac{b}{\gamma_k}\right) \\ -a\left(\sqrt{-2i\sum_{n=1}^{d}\gamma_n\left(i\beta_n\delta_n^2u_n - \frac{1}{2}\delta_n^2u_n^2\right)} - \frac{b}{\gamma_n}\right)\Bigg].\end{aligned} \tag{3.33}$$

In the following example this mistake is corrected.

3.4.3 Example with Inverse Gaussian and Gamma Subordinator

In Chapter 2 we introduced inverse Gaussian process and Gamma process. They were used as one dimensional subordinator. Here we extend the univariate case to multivariate case, with multivariate subordination.

Example 3.4.2 (Multivariate Gamma subordinator) *The Laplace exponent of gamma subordinator is* $-a\log(1-\frac{u}{b})$. *We have* Z_t *in (3.28) follows* $\Gamma(a,b)$ *at time one and* X_t^i *in (3.28) follows* $\Gamma(\frac{b}{\alpha_i}-a, \frac{b}{\alpha_i})$ *at time one. Let* $0 < \alpha_i < \frac{b}{\alpha_i}$.

Following the above construction, the characteristic function of $\mathbf{Y}$ *is given by:*

$$\Phi_{\mathbf{Y}}(\mathbf{u}) = \prod_{k=1}^{d} \left(1 - \frac{\alpha_k(i\mu_k u_k - \frac{1}{2}\sigma_k^2 u_k^2)}{b}\right)^{-t\left(\frac{b}{\alpha_k} - a\right)} \left(1 - \frac{\sum_{k=1}^{d} \alpha_d(i\mu_k u_k - \frac{1}{2}\sigma_k^2 u_k^2)}{b}\right)^{-ta}. \tag{3.34}$$

Example 3.4.3 (Multivariate Inverse Gaussian subordinator) *The Laplace exponent of Inverse Gaussian subordinator is* $-2a(\sqrt{b^2 - 2t} - b)$*. In this example the* $X_t^i + \alpha_i Z_t$ *in (3.28) is replaced by* $X_t^i + \gamma_i^2 Z_t$*, with* X_t^i *follows* $IG(1 - a\gamma_i, \frac{b}{\gamma_i})$ *at time one and* $\gamma_i^2 Z_t$ *follows* $IG(a\gamma_i, \frac{b}{\gamma_i})$ *at time one. The independent Brownian motions in (3.29) are replaced by* $\beta_i\delta^2 t + \delta_i B_t^i$*. Let* $b > 0$*,* $1 < a < \frac{1}{\gamma_i}$*,* $\alpha_i > 0$*,* $-\alpha_i < \beta < \alpha_i$*,* $\delta > 0$*,* $b_i = \frac{b}{\gamma_i} = \delta_i\sqrt{\alpha_i^2 - \beta_i^2}$*. Thus the characteristic function of the process is as follows:*

$$\phi_{\mathbf{Y}}(\mathbf{u}) = \exp\left[-\sum_{k=1}^{d}(1 - a\gamma_k)\left(\sqrt{-2\left(i\beta_k\delta_k^2 u_k - \frac{1}{2}\delta_k^2 u_k^2\right) + \frac{b^2}{\gamma_k^2}} - \frac{b}{\gamma_k}\right)\right. \tag{3.35}$$
$$\left. -a\left(\sqrt{-2\sum_{n=1}^{d}\gamma_n^2\left(i\beta_n\delta_n^2 u_n - \frac{1}{2}\delta_n^2 u_n^2\right) + b^2} - b\right)\right].$$

Remark 3.4.4 *The model of the Normal Inverse Gaussian case is of infinite variation, which verifies the condition 2.3 stated in Theorem 2.4.7. At least, in this case, the finite dimensional linear combination of several Normal Inverse Gaussian process with each satisfies that condition will suffice to satisfy the conditions in Theorem 2.4.7. So the model is valid. As for the Variance Gamma case, it is of finite variation. So one needs to verify the condition 2.4 of the same theorem.*

3.4.4 Numerical Results and Benchmark Comparison

In this part we calculate a European basket call, the payoff function is defined as

$$(\omega_1 S_t^1 + .. + \omega_d S_t^d - K)^+. \tag{3.36}$$

Price Pairs	FT	MC	std err
$S_0^1 = 100$, $S_0^2 = 100$, $K = 80$, $\omega_1 = 0.5$, $\omega_2 = 0.5$	19.57	19.16	0.051
$S_0^1 = 120$, $S_0^2 = 70$, $K = 80$, $\omega_1 = 0.5$, $\omega_2 = 0.5$	14.57	14.99	0.033
$S_0^1 = 120$, $S_0^2 = 70$, $K = 80$, $\omega_1 = 0.4$, $\omega_2 = 0.6$	9.57	9.99	0.035
$S_0^1 = 120$, $S_0^2 = 70$, $K = 80$, $\omega_1 = 0.6$, $\omega_2 = 0.4$	24.57	24.78	0.022

Table 3.1: Computation comparison between transform method and Monte Carlo simulation

For the simplicity of the comparison, we choose to implement a two dimensional case. The price will be calculated by transform method as described in the previous sections of this chapter and Monte Carlo simulation which will serve as benchmark. The detailed simulation method can be found in the first section of Chapter 5.

For the simplicity of the computation and comparison, here we take just 10000 paths for all the Monte Carlo Simulations. We also take the N in definition 3.3.1 to be 2^{10}. In the following table we compare several prices and weights pairs.

As we can see, the transformation method confirms relatively well to the Monte Carlo simulation. As simulation paths number becomes bigger, the standard error can be further reduced.

Here we want to further compare the Monte Carlo simulation and the transform method. In practice there are indeed advantages for the transform method. If all parameters are well chosen, the method can in one run generate a matrix which can be reused for many strikes. From this point of view, it is fast and efficient. Compared with naive Monte Carlo simulation this method is far better. But this method also has several 'down points'. It is much more complicated to understand than the simple Monte Carlo simulation. It is also not as 'robust' as Monte Carlo simulation. As we have seen in both equation 3.12 and 3.19, we have a small ϵ under the integral sign. This small variable turns out to play a big role in the precision of the result. We need to change the value of this ϵ to make

the imaginary part of the transformed result goes to zero. Only at this moment can we really get a good result which conforms well the Monte Carlo simulation. If no weight changes undergo among different dimensions during subsequent calculation it is fine. If their are changes, we need to readjust these ϵs. One or two dimensions can still be fine. If the dimension goes up, there will be a very big trouble. This is not the end, as this is Fourier transform that we are using, it has one common problem, the solution oscillate at certain parts. This behavior will sometimes give us aberrant results. Another relatively big problem is the change of dimension, if we really want to go to higher dimensions we need to apply multiple times of Fourier transform on a high dimensional matrix which is technically difficult to be converted to parallel algorithms. If we do not convert it into parallel algorithms, the high speed will not be guaranteed. Even if a dimension adjustable program without parallel algorithms will still introduce a lot of complexity on the programming itself.

From our discussion above, this should be a dilemma: each one has its problems and merits. This is where new technology comes into scene to change the balance. By using massively parallel computation, we can greatly improve the speed of Monte Carlo simulation. Coupled with its simplicity, it is a good choice for production code in real life calculation. Still, we have to say transform method is unbeatable under one dimensional situation.

Chapter 4

Hedging Methods

In this chapter we are going to look at the local risk minimization strategy. After this, we are going to review some other strategies based on the multivariate Lévy process.

Hedging is also a very important aspect of financial mathematics. A good strategy will protect the writer[1] of a contract from risk of loss. Hedging is usually achieved by constructing a portfolio which will replicate the payoff of the contract issued by the writer.

Let us take basket option which we have discussed previously as an example. We only see the European type basket option and don't take into consideration of any default risk. After the option is sold, there are two scenarios that might happen at the expiration date T. The basket $\sum_i \omega_i S_i$ is lower than or equal to the strike price K. In this situation, the buyer will not exercise the option and the deal is closed at this moment. On the contrary, the basket can also be more expensive than the strike. In this situation, the writer will owe the buyer an amount equals to $\sum_i \omega_i S_i - K$. To mitigate this risk, the writer will choose to hedge the option by using the money comes from the sales of this option to buy underlying stocks in the basket and risk free bond. The objective is to have this portfolio's value greater or equal to the basket minus the strike, thus the writer

[1]The writer of a contract is the issuer of a contract. In real life situation, many of them are investment banks. That is the reason that they are called "sell side". They sell those contracts to investors, speculators or institutional treasurers, etc.

will always be safe whatever the outcome is. Under Black-Scholes' model, a delta hedging strategy was introduced. The delta is calculated as

$$\Delta_t = \frac{\partial V}{\partial S}(S,t), \tag{4.1}$$

where V is the portfolio as we had seen in equation (4.1). This delta as indicated by the equation is the change of portfolio price(in this case the option price) with respect to the price change of the underlying asset. It can also be considered as a hedge ratio, which indicates how much underlying asset should hold given an amount of contracts sold. In Black-Scholes' Model, if the time step is small enough (continuous re-balancing and dynamic hedging), with this strategy we can perfectly replicate the portfolio. In this sense, this theory is very good.
However as we have discussed at the very beginning, a model without jump is far from realistic. We have also discussed about pricing under a jumping model in the previous sections. It is reasonable to discuss about how to hedge in the jumping models[1].Under these models the exact replication is not possible anymore. Hedging therefore becomes an approximation of terminal pay-off with an admissible portfolio with respect to different criterions. Though there are various hedging strategies available, we are going to emphasis on only one of them - locally risk minimization strategy and demonstrate how this strategy can be applied in our context.

To give a more complete discussion on hedging strategies, in the second part of this section we are going to see two other different hedging strategies which are based on different theoretical settings from this one.

4.1 Locally Risk-minimizing Hedging Strategy

Before we start, in this part we need some background of pseudo differential operators. For a complete treatment of this subject refer to [12]. This is a three volumes book, but here we only use some applications for parabolic equation. A quick checking for theories that related to this part can be found in the Chapter

[1]Here we are talking about exponential Lévy models.

2 of the second volume.

We approach the problem as a writer of the contract. So as a writer, we will short (sell) a contract and buy (long) underlying securities upon which the contract is built. We denote the contract by $C_t(\mathbf{S}_t)$ where $\mathbf{S}_t = (S_t^1, ..., S_t^d)^T$. We denote the weight of each security by ω_t where $\omega = (\omega_t^1, ..., \omega_t^d)^T$. At each instance, we have the following form for the portfolio:

$$W_t := \omega_t \cdot \mathbf{S}_t - C_t(\mathbf{S}_t) + w_t, \tag{4.2}$$

where W_t denotes wealth and the w_t denotes the residue of wealth.
At each time change Δt, the portfolio becomes :

$$W_{t+\Delta t} := \omega_t \cdot \mathbf{S}_{t+\Delta t} - C_{t+\Delta}(\mathbf{S}_{t+\Delta t}) + w_{t+\Delta t}, \tag{4.3}$$

where $w_{t+\Delta t} = w_t e^{r\Delta t}$.

The objective is to minimize the local variance of this portfolio:

$$\mathbf{E}_t[(W_{t+\Delta t} - \mathbf{E}_t[W_{t+\Delta t}])^2]. \tag{4.4}$$

Remark 4.1.1 *The probability of this expectation is the historical probability* $\mathbb{P}$[1]*, same for all the expectation afterwards. The dynamics are underlying price processes.*

We denote $x_j = \ln S_t^j$, $c(x,t) = C(e^{x_1}, ..., e^{x_n}; t)$. We substitute the equation (4.2) and (4.3) to (4.4). After rearrangement, we have the (4.4) equals the fol-

[1]this probability is the same as the $\mathbb{P}$ in Chapter 2 where the risk neutral pricing was discussed. It is actually the probability defined by the market model.

lowing form

$$\begin{aligned}\mathbf{E}_t[(C_{t+\Delta t}(\mathbf{S}_{t+\Delta t}) - \mathbf{E}_t[C_{t+\Delta t}(\mathbf{S}_{t+\Delta t})])^2] \\ - 2\sum_{j=1}^{n} \omega_t^j \mathbf{E}_t[(C_{t+\Delta t}(\mathbf{S}_t) - \mathbf{E}_t[C_{t+\Delta t}(\mathbf{S}_t)])(S^j_{t+\Delta t} - \mathbf{E}_t[S^j_{t+\Delta t}])] + \\ \sum_{i,j=1}^{n} \omega_t^i \omega_t^j \mathbf{E}_t[(S^j_{t+\Delta t} - \mathbf{E}_t[S^j_{t+\Delta t}])(S^i_{t+\Delta t} - \mathbf{E}_t[S^i_{t+\Delta t}])]. \quad (4.5)\end{aligned}$$

We minimize this expression with respect to ω_t to get for $j = 1, ..., n$,

$$\begin{aligned}\sum_{i=1}^{n} \omega_t^i (\mathbf{E}_t[S^j_{t+\Delta t} S^i_{t+\Delta t}] - \mathbf{E}_t[S^i_{t+\Delta t}]\mathbf{E}_t[S^i_{t+\Delta t}]) \\ = \mathbf{E}_t[C_{t+\Delta t}(\mathbf{S}_{t+\Delta t}) S^j_{t+\Delta t}] - \mathbf{E}_t[C_{t+\Delta t}(\mathbf{S}_{t+\Delta t})]\mathbf{E}_t[S^j_{t+\Delta t}]. \quad (4.6)\end{aligned}$$

We have the following two relations[1]:

$$\mathbf{E}_t[S^i_{t+\Delta t} S^j_{t+\Delta t}] = e^{\Delta t \psi(-i(e_i+e_k))} S^i_t S^j_t \quad (4.7)$$

and

$$\mathbf{E}_t[S^i_{t+\Delta t}]\mathbf{E}_t[S^j_{t+\Delta t}] = e^{-\Delta t(\psi(-ie_j)+\psi(-ie_i))} S^i_t S^j_t, \quad (4.8)$$

where e_i and e_j are standard bases of $\mathbb{R}^n$.
As Δt goes to 0, we take the first order approximation for the exponential form. The LHS of the (4.6) becomes

$$\sum_{i=1}^{n} \omega_t^i S_t^j S_t^i [-\psi(-i(e_j + e_i)) + \psi(-ie_j) + \psi(-ie_i)]\Delta t. \quad (4.9)$$

We denote L the infinitesimal generator for the process. We have the following relation from the Pseudo-Differential Operator theory:

$$\partial_t + L = \partial_t - \psi(D_x) \quad (4.10)$$

[1]The ψ is the characteristic exponent of the underlying price dynamics

$$\psi(D)e^{x_j} = e^{x_j}\psi(D - ie_j), \tag{4.11}$$

where D is a differential operator.[1] Then we can rewrite the RHS of equation (4.6) in the following form:

$$\begin{aligned}\mathbf{E}[C_{t+\Delta t}(\mathbf{S}_{t+\Delta t})S^j_{t+\Delta t}] - C_t(\mathbf{S}_t)S^j_t \\ - (\mathbf{E}[C_{t+\Delta t}(\mathbf{S}_{t+\Delta t})]S^j_t - C_t(\mathbf{S}_t)S^j_t) \\ + (\mathbf{E}[C_{t+\Delta t}(\mathbf{S}_{t+\Delta t})]S^j_t - \mathbf{E}[C_{t+\Delta t}(\mathbf{S}_{t+\Delta t})]\mathbf{E}[S^j_{t+\Delta t}]).\end{aligned} \tag{4.12}$$

There are three lines in the equation (4.12). The first line can be expressed as:

$$\mathbf{E}[C_{t+\Delta t}(\mathbf{S}_{t+\Delta t})S^j_{t+\Delta t}] - C_t(\mathbf{S}_t)S^j_t = S^j_t(\partial_t - \psi(D_x - ie_j))c(x,t)\Delta t + o(\Delta t). \tag{4.13}$$

The second line can be expressed as:

$$\mathbf{E}[C_{t+\Delta t}(\mathbf{S}_{t+\Delta t})]S^j_t - C_t(\mathbf{S}_t)S^j_t = S^j_t(\partial_t - \psi(D_x))c(x,t)\Delta t + o(\Delta t). \tag{4.14}$$

Then the third line as:

$$\mathbf{E}[C_{t+\Delta t}(\mathbf{S}_{t+\Delta t})]S^j_t - \mathbf{E}[C_{t+\Delta t}(\mathbf{S}_{t+\Delta t})]\mathbf{E}[S^j_{t+\Delta t}] = S^j_t\psi(-ie_j)c(x,t)\Delta t + o(\Delta t). \tag{4.15}$$

Putting together with the LHS of the equation we have the following form:
for $j = 1, ..., n$

$$\begin{aligned}\sum_i^n \omega^i_t S^i_t[-\psi(-i(e_j + e_i)) + \psi(-ie_j) + \psi(-ie_i)] = \\ [-\psi(D_x - ie_j) + \psi(D_x) + \psi(-ie_j)]c(x,t).\end{aligned} \tag{4.16}$$

We set matrix $\{V(e_i, e_j)\}^n_{i,j=1} := \{-\psi(-i(e_i + e_j)) + \psi(-ie_i) + \psi(-ie_j)\}^n_{i,j=1}$ vector $\{m(e_j)\}_{j=1,...,n} := \{-\psi(D - ie_j) + \psi(D) + \psi(-ie_j)\}_{j=1,...,n}$ and vector $\Omega_t :=$

[1] for this part, we assume these two relations (4.10)(4.11) to be true. For demonstration and theory refer to [12], volume II Chapter 2 part 2.7 especially the example in this part.

$\{\omega_t^j S_t^j\}_{j=1,\ldots,n}$ So the whole equation is as follows:

$$V\Omega_t^T = m^T c(x,t). \tag{4.17}$$

If V is invertible, we have the result of the hedging ratio:

$$\Omega_t^T = V^{-1} m^T c(x,t). \tag{4.18}$$

It can be verified that if the underlying process follows a multivariate Brownian Motion, the equation (4.18) will give back the delta hedging strategy.
This equation does not depend on payoff function and can be changed to other processes given the characteristic exponent is explicit. Further more, if the matrix V is invertible, an explicit hedging ratio can be easily calculated.
We should have posted here an example to demonstrate the calculated result for our Normal Inverse Gaussian model or Variance Gamma model, for their characteristic exponents are explicit. But as it turned our that the explicit function is rather complicated thus making the formula (4.18) very hard to calculate by hand. There are two methods can be applied: if one wants to have an explicit formula for demonstration purpose, one can use the symbolic toolbox of Matlab to do the calculation. But this method might be feasible only for lower dimension. The more practical method is to directly substitute parameters to explicitly calculate the matrix V and the vector m. This is the way this method can be used in a computer program.

Local risk minimization is one of those quadratic hedging strategies. Its theoretical foundation was laid by Föllmer and Schweizer in [8]. In their research a minimum martingale measure was proposed and the measure can be uniquely determined. Also a general form of hedging ratio was proposed in the research and was written in the form of a Radon-Nikodym derivative was also derived. In fact the matrix form given in (4.18) can be analogous to equation (2.15) in [8]. Here we use underlying securities to hedge the option. But it is by no means the only one. In incomplete market, options are no longer redundant. It is also possible to do the hedging with other options[6] and give better result than simply

using underlying securities.

4.2 Alternative Hedging Methods

In the previous section we have presented a quadratic hedging strategy for our model. In this section we are going to review two other methods. One is based on stochastic receding horizon control theory for further details refer to [19] and [20]. This method is intuitive in its construction and easy for higher dimension implementation. The other is based on Malliavin calculus, which is a theory that allows us to take "derivative" with respect to noise in the system. Malliavin calculus appeared relatively early. But previously much study was for Brownian kind of noise. For jump type models refer to [13], [2], [3], [18].
This part is written for us to see the hedging problem from different perspectives.

4.2.1 Multi-Dimensional Option Hedging with Receding Horizon Control

This method is a dynamic hedging strategy formulated to hedge the risk of basket option with presence of transaction costs. Compared to the method presented previously this one is more close to the reality. It takes in to account the transaction cost and it is discrete.

Receding horizon control means the following: at each time step, we solve a finite horizon optimal control problem and implement the initial control action. Thus this is a sub-optimal control policy. In [20], author used semi-definite programming to solve the finite horizon optimization problem. In the example presented in [19], author used a multivariate Brownian motion as noise generator. But the setting of this model is not subject to the difference of noise if the noise's property is "good" enough.

In the text, author formulated the hedging problem into the following control problem:
$\max_{u_k^j, \tau_k^j} \delta$

subject to:
$u^j_{-1} = 0,\ u^j_N = 0,\ j = 1...l$
$W_{k+1^-} = (1 + r_f)W_{k^-} + \sum_{j=1}^{l}\{(\mu^j - r_f + \bar{w}^j_k)u^j_k - (1 + r_f)r^j_k\}$
$S^i_{k+1} = (1 + \mu^i + \bar{w}^i_k)S^i_k, i = 1...n$
$\tau^j_k = \kappa^j |u^j_k - u^j_{k-1}\frac{S^j_k}{S^j_{k-1}}|, j = 1...l$
$\mathbf{E}[W_{N^-} - \sum_{j=1}^{l}\tau^j_N] - \gamma\sqrt{Var\left(W_{N^-} - \sum_{j=1}^{l}\tau^j_N\right)} \geq \delta$
$\mathbf{E}[W_{N^-} - \sum_{j=1}^{l}\tau^j_N - (\alpha^T S_N - \bar{K})] - \gamma\sqrt{Var\left(W(N^-) - \sum_{j=1}^{l}\tau_j(N) - (\alpha^T S(N) - \bar{K})\right)} \geq \delta$

In the above control problem, S^i_k denotes the price of stock j; W_k and W_{k^-} denote the wealth immediately after and before the trade at time k respectively; τ^j_k denotes the transaction cost by stock j at time k; $r_f = r\Delta t$ is the instant interest; μ^j is the instant drift of the underlying dynamics; $\bar{w}^j_k$ denote the "noise"(price change) for stock j at time k; κ is the proportion of transaction cost.

This formulation is designed to include first two moments of the dynamics. Because this allows the formulation of receding horizon on-line optimization to be solved as a semi-definite program, thus gaining in computation power. However features further than two moments is not included.
This problem is then transformed to an on-line optimization with horizon T, and implemented in to a receding horizon algorithm. For details refer to section 3.4 and 3.7 of [19].

4.2.2 Multidimensional Option Hedging with Malliavin calculus

In the references given at the beginning of this section, we have two approaches. One focuses on jump diffusion type of process, as Bavouzet[2] has presented in their work. In their research, an integration by parts formula for a general multi-dimensional random variable that has differentiable density and absolutely continuous law was developed. Another is based on time-changed models as the ones we have presented in the beginning chapters of this project. The research

of Bayazit[3] is in this direction. In his research, under one dimensional case, various Greeks[1] are calculated for both Variance Gamma model and Normal Inverse Gaussian model. In the research of Arturo[13], sensitivities are calculated for mutidimensional time-changed model. The research of Petroni[18] is also of multidimensional, and is based on the research of Kohatsu-Higa. Sensitivities are also calculated under a multidimensional Brownian motion model for various exotics. For simplicity, here we only present some basics of Malliavin calculus and give a one dimensional example for the calculation of Δ for Variance Gamma model as presented in Bayazit's[3] research.

Malliavin calculus introduces an additional term, H which is also called Malliavin weight. With the help of this term the derivative operator for the expectation will be removed. i.e..

$$\frac{\partial}{\partial S_0}\mathbf{E}[\phi(S_T)|\mathcal{F}_0] = \mathbf{E}[\phi(S_T)H(S_T, \frac{S_T}{S_0})|\mathcal{F}_0].$$

This is like the test function in the distribution theory which will "smooth" the expectation. We present some essential definitions of this theory and one example with the calculation of Δ of a payoff function based on Variance Gamma process. Given a sequence of random variables $(U_n)_{n\in\mathbb{N}^*}$ on a probability space $(\Omega, \mathcal{F}, \mathbb{P})$. U_n has moments of any order. We assume ρ_n to be the density of U_n. We also assume that ρ_n is continuously differentiable on $\mathbb{R}$, $\forall m \in \mathbb{N}, \lim_{y\to\pm\infty} |y|^m \rho_n(y) = 0$ and $\frac{\partial_y \rho_n(y)}{\rho_n(y)}$ has at most polynomial growth. A random variable F is called a simple functional if there exists some $n \in \mathbb{N}$ and some measurable function $f : \mathbb{R}^n \to \mathbb{R}$ such that $F = f(U_1, ..., U_n)$. The space of simple functionals $f \in \mathcal{C}^m_\uparrow(\mathbb{R}^n)$[2] is denoted by $S_{(n,m)}$. A k-length simple process is a sequence of random variables $V = (V_i)_{i\leq k}$, $k \leq n$ such that $V_i = f_i(U_1, ..., U_n)$. The space of $k-length$ processes is denoted by $P^k_{(n,m)}$.

Definition 4.2.1 (Inner Product) *Let $U = (U_i)_{i\leq k}$ and $V = (V_j)_{j\leq k}$ be two*

[1] By Greeks we are talking about sensitivities of contracts with respect to different variables. In the paper, Δ, Γ which is sensitivity of Δ with respect to price change, Vega, etc are calculated.

[2] This means f is up to order m differentiable.

k-length simple processes in $P^k_{(n,1)}$ *then*

$$\langle U, V\rangle = \sum_{i=1}^{k} U_i V_i \tag{4.19}$$

is called the inner product of U *and* V*.*

Definition 4.2.2 (Malliavin Derivative) *The* $k-$*length simple process* $D^k : S_{(n,1)} \to P^k_{(n,0)}$, $k \leq n$ *is called the Malliavin derivative operator and it is defined as* $D^k F = (D_i F)_{i\leq k}$ *where* $F = f(V_1, ..., V_n) \in S_{(n,1)}$ *and* $D^k_i F = \partial_i f(V_1, ..., V_n)$, $i \leq k$*.*

Definition 4.2.3 (Skorohod Integral) $\delta^k : P^k_{(n,0)} \to S_{(n,1)}$, $k \leq n$ *is called the Skorohod integral operator and is defined for any* $k-$*length simple process* $U \in P^k_{(n,0)}$ *such that*

$$\delta^k(U) = -\sum_{i=1}^{k} [D_i U_i + \theta_i(V_i) U_i], \tag{4.20}$$

where

$$\theta_i(y) = \partial_y \ln[\rho_i(y)] = \frac{\rho_i^{'}(y)}{\rho(y)}, \quad if \rho_i(y) > 0. \tag{4.21}$$

Proposition 4.2.4 (Duality Formula) *Let* $F \in S_{(n,1)}$ *and* $U \in P^k_{(n,0)}$*, then*

$$\mathbf{E}[\langle D^k F, U\rangle] = \mathbf{E}[F\delta^k(U)]. \tag{4.22}$$

Definition 4.2.5 (Malliavin Covariance Matrix) *Let* $F = (F_1, F_2, ..., F_d)$ *be an d-dimensional vector of simple functionals such that* $F_i \in S_{(n,1)}$*. The matrix* $M^k_\sigma(F)$ *is called the Malliavin covariance matrix of* F *whose entries are given by*

$$M^k_\sigma(F)_{ij} = \langle DF_i, DF_j\rangle = \sum_{t=1}^{k} \partial_t f_i \partial_t f_j(V_1, ..., V_n), \tag{4.23}$$

where $F_i = f_i(V_1, ..., V_n)$*.*

Theorem 4.2.6 (Integration by Parts) *Let $F = (F_1, F_2, ..., F_d) \in S^d_{(n,2)}$ and $G \in S_{(n,1)}$. We assume that the $M^k_\sigma(F)$ is invertible and denote $M^k_\gamma(F) = [M^k_\sigma(F)]^{-1}$. We also assume that $\mathbf{E}[detM^k_\gamma(F)]^4 < \infty$. The for every smooth function $\phi : \mathbb{R}^d \to \mathbb{R}$*

$$\mathbf{E}[\partial_i \phi(F)G] = \mathbf{E}[\phi(F)H^k_i(F,G)], \tag{4.24}$$

where $H^k_i(F,G) = \sum_{i=1}^d GM^k_{\gamma_{ji}} L^k F - M^k_{\gamma_{ji}}(F)\langle D^k F, D^k G\rangle - G\langle D^k F, D^k M^k_{\gamma_{ji}}(F)\rangle$.

Example 4.2.7 (Variance Gamma model) *The Variance Gamma model S_T is discretized as follows:*

$$S_T = S_0 e^{rT + \sum_{i=1}^n \sigma\sqrt{\Delta X_i}B_i + \theta\sum_{i=1}^n \Delta X_i}, \tag{4.25}$$

where X_t follows gamma process $\Delta X_i = X_{t_i} - X_{t_{i-1}}$. Then the Greek Δ is as follows:

$$\Delta = e^{-rT}\frac{\partial}{\partial S_0}\mathbf{E}[\phi(S_T)] = e^{-rT}\mathbf{E}[\phi'(S_T)\frac{S_T}{S_0}] = \mathbf{E}[\phi(S_T)H^k_\Delta(S_T, \frac{S_T}{S_0})], \tag{4.26}$$

where $H^k_\Delta(F, G_{S_0}) = \frac{-1}{S_0}M^k_\gamma(F)\left(M^k_\sigma(F) - S_T^2\sum_{j=1}^k Z_j\sigma\sqrt{\Delta G_j}\right) - \frac{1}{S_0}M^k_\gamma(F)M^k_\sigma(F) + \frac{2}{S_0}M^k_\gamma(F)M^k_\sigma(F)$, $M^k_\sigma(F) = \sum_{i=1}^k \sigma^2\Delta G_i S_T^2$ and $M^k_\gamma(F) = \frac{1}{M^k_\sigma(F)}$.

As we can see from the formula, the Δ is the sensitivity with respect to S_0. This feature may be where this model is limited. It may be possible to apply the logic in the receding horizon control and update the S_0 at each step and set T.

Chapter 5

Simulation Method and GPU computing with CUDA

In this chapter we are going to firstly present the simulation method for the multivariate subordination model. Then we are going to discuss about recent development in GPU[1] computing. We are going to implement our simulation method with both Matlab and CUDA[2], and comparison of result will tell the importance of this development.

5.1 Simulation method

In the previous part of this work, a multivariate time changed model was described. Now, we are going to talk about the simulation method for this model. In Chapter 6 we gave two examples, one was a model that used Gamma process as its subordinator, another was a model that used Inverse Gaussian process as its subordinator. Here we are going to present a simulation method with the help of the second one[3].

There are two steps for the simulation: the first step is to construct the time changing process, the second step is to use the process produced in the first step to

[1]GPU stands for Graphic Processing Unit.
[2]CUDA stands for Compute Unified Device Architecture.
[3]For details see Example 3.4.2.

subordinate a multivariate independent Gaussian process. So before everything can start we have to build a Inverse Gaussian random number generator. The method is as follows:

Algorithm 5.1.1 (Generating Inverse Gaussian Random Variables) *The Inverse Gaussian density has the following form:*

$$f(x) = \sqrt{\frac{\lambda}{2\pi x^3}} e^{-\frac{\lambda(x-\mu)^2}{2\mu^2 x}} 1_{x>0}. \tag{5.1}$$

The algorithm is as follows:
1.Generate a normal random variable N;
2.Set $Y = N^2$*;*
3.Set $X_1 = \mu + \frac{\mu^2 Y}{2\lambda} - \frac{\mu}{2\lambda}\sqrt{4\mu\lambda Y + \mu^2 Y^2}$*;*
4.Generate a uniform [0,1] random variable U. If $U \leq \frac{\mu}{X_1+\mu}$ *return* X_1*, else return* $\frac{\mu^2}{X_1}$*.*
This algorithm is based on the work of Schucany[17].

With the help of the above Inverse Gaussian random variable generator, we have the following algorithm for multivariate Normal Inverse Gaussian process.

Algorithm 5.1.2 (Simulating multivariate Normal Inverse Gaussian process) *We are going to simulate the process (6.3) with the subordinator (6.1) on a time grid of* $t_1, ..., t_n$*. So we have to firstly simulate the subordinator* $\mathbf{S}_t = (S_t^1, ..., S_t^d)$*. At each time step i:*
1. Generate a Inverse Gaussian variable ΔZ *with parameter* $\mu_{Z_i} = t_i - t_{i-1}$ *and* $\lambda_{Z_i} = \frac{(t_i - t_{i-1})^2}{\xi_z^2}$ *where* $\xi_z = \frac{\mu_z^{3/2}}{\sqrt{\lambda_z}}$*;*
2. Generate d independent Inverse Gaussian variable ΔX_i^k *with each variable has parameters* $\mu_{x^k} = t_i - t_{i-1}$ *and* $\lambda_{x^k} = \frac{(t_i - t_{i-1})^2}{\xi_{x^k}}$ *where* $\xi_{x^k} = \frac{\mu_{x^k}^3/2}{\sqrt{\lambda_{x^k}}}$ *and* $k = 1, ..., d$*. So* $\Delta S_i^k = \Delta X_i^k + \alpha^k \Delta Z_i^k$*;*
3. Generate d i.i.d N(0,1) random variables $N^1, ..., N^d$*. Set* $\Delta Y_i^k = \sigma N^k \sqrt{\Delta S_i^k} + \theta \Delta S_i^k$*.*
Then we just follow this for all time steps until the end for all d assets.

With the above simulation method, we can perform Monte Carlo simulation to calculate the expectation of the payoff function at the expiry as we have seen

in the previous chapters. We have implemented a Matlab program to perform this calculation. Since we are dealing with a multidimensional problem, we have to pay much attention to the implementation of the program. "for" loops should be avoided. Matrix forms should be applied in the implementation. By doing this we are actually trade off memory for speed. The program for Inverse Gaussian generator and the simulation kernel can be found in the Appendix.

Apart from the Matlab implementation, we also implemented the program with C++ to double check the performance under industry standard.

5.2 Choosing hardware according to the nature of the problem

Before we present anything of this part. A comparison shall be produced to prove its importance. As we have seen in the previous section, we had carefully implemented the algorithm in Matlab[1]. The program was run on a laptop with Intel i7-3612 CPU[2]which is one of the higher end CPU as of the year 2012. we tried with 6 assets, and 10000 paths. The Matlab implemetation and the CUDA-thrust[3] implementation gave the same result, whereas the former needed 11.3089 seconds the latter needed just 0.282 second! Dividing 11.3089 by 0.282 gives 40.1025. This is a 40 times speed up. The program is also implemented with C++ which unfortunately needs 25.893 seconds. If we insist to compare the GPU calculation with C++ it is a 92 times speed up.

Remark 5.2.1 *There might be a little bit of surprise here about the C++ performance. It is even worst than Matlab. There should be no surprise actually. Because Matlab is a specialized software optimized for the matrix calculation. So in the Matlab program, we avoided the use of for loops. For example, we generate*

[1]The operating system is Windows 7 (64bit). The Matlab that I am using is Matlab 2012. The C++ development environment is Visual Studio 2012. The C++ standard that I am using is the latest C++11 standard. Standard Template Library was involved in the implementation.

[2]The clock rate of a single core is 2.1 GHz.

[3]This is the programming language used for the nVidia GPU.

in one operation a big matrix of random variables, this is much more efficient than looping for each iteration. The calculation is also done with matrix operation. So it is normal that our Matlab program actually runs faster than C++ implementation.

Some may argue that it is possible to build a multi-threaded program by using the multi-threading library of C++11 standard. And in my case the CPU has actually 8 cores, each one of them actually can give dozens of threads, together you can have around 60 something threads running. Naively speaking we can have a 60 times speed up.

However the truth is not that optimistic. The reason lies still in the CPU itself. Undeniably,threads on different cores are truly independent. It it also true that one can have multiple threads on one core. In fact many years ago, almost all our personal computers ran on one single core and we could still watch videos and edit documents at the same time without any problem. But the truth is the parallelism on a single cores is not truly parallel. It is actually a pseudo-parallelism, which means there is a context switching mechanism undergoing all the time. This context switching actually chops different tasks into sequential pieces and switch from one to another all the time. Thus we will have an impression that all the tasks progressing simultaneously.

Evidently, the higher the clock speed, the faster the context switching and program execution. So if we trace back the history of processor development before 2000 or 2003, it was almost a history of raising clock speed. Back to 1980s, a 80286 processor has a clock rate 16MHz, in 2006, a Intel Core 2 Duo has more than 2 GHz. But during recent years, the limit comes. Higher speed processor is becoming exceedingly difficult to build. So instead of building faster single core CPU, the industry chooses to go to multicore CPU or even build multi-CPU motherboards.

Before we proceed, we need to examine a bit more the parallelism. There are two big categories, one is data parallelism the other is task parallelism. CPU is optimized for the task parallelism. Task parallelism can be seen everywhere:

multiple windows running at the same time, multiple internet connections, etc. Since you won't have thousands of tasks, dozens of cores will beyond necessary. Data parallelism is characterised by the huge quantity of data and relatively light calculation for each data point. For example, if we want to build a neural network, the training of the network many involve big quantity of data. Whereas for each data point, a simple logistic function calculation may suffice. In our case, Monte Carlo simulation is similar to the data parallelism. For each path we run a fixed quantity of lightweight calculation and many paths are needed to give a satisfactory result. GPU is well suited to this kind of parallelism it may have thousands of cores. The latest GeForce GTX690 has 3072 cores.

Apart from the number of cores, memory bandwidth might also give a clue about which hardware is better for the given task. This criteria actually measures how fast data is transferred. The typical high end CPU(Intel i7 series) memory bandwidth is around 20GB/s. Whereas the high end GPU (GeForce GTX 690)will have memory bandwidth around 350GB/s. For data heavy calculation this is critical to have high bandwidth.

For more information refer to Kirk's[14] book and nVidia's web site for developers.

5.3 GPU and CUDA

5.3.1 GPU at a glimpse

nVidia GPU is formed by Streaming Multiprocessors(SM). For example, my machine has 96 cores. Each core is actually a streaming processor(SP). These 96 cores are organized in two groups with each group has 48 cores. Each group of these 48 cores[1] or SPs is a Streaming Multiprocessor. Each core or SP can run one or more threads at a time. Still my example, I have 2 SMs, each one can be seen as a block. I have thus 2 blocks. Each block in my case can have 1024

[1]This number depends on the hardware version. Current version 2.1 has 48 cores a SM.

threads, so together I can have 2048 threads. These two blocks together forms a grid. In the case of GeForce GTX 690, it has 3072 cores which is 64 SMs, the threads it can have is 65536. So if each CPU core can have 32 threads and on each chip we have 8 cores, we need 256 multicore CPU to produce that many threads. This is equivalent to a medium size cluster already.

5.3.2 CUDA-thrust implementation

CUDA is actually a computing scheme that combines the CPU and GPU computing power together and having each part perform what they do best. The letter 'C' in this acronym actually means unified.

The compiler actually divides the program into two parts: one part execute on the 'host' which is the machine on which the graphic card resides, the other part executes on the 'device' which is the graphic card or more precisely the SMs. When reflected in the program, we will have the key word "__host__" at the top of the part for 'host', and "__device__" at the top of the part for 'device'. In our context, since we are running Monte Carlo simulation to calculate an expectation, the kernel will do the calculation for one path on one thread. We then achieve the parallelism with the function "transform_reduce()".[1] 'Transform' here means a path of simulation, 'reduce' here is to sum the result up.

The program itself is written with thrust[2] library which is a C++ like abstraction of CUDA language. Thrust is fully compatible with CUDA and C++. This facilitates the programming process on the host and enhances the readability of the program written. But if programmers intend to do lower level control of threads, a good understanding of CUDA itself is desirable.

The randomness come from XORWOW random generator. It is included in the latest release of CUDA toolkit. This generator passed the full suite of NIST pseudorandomness test. In the program it is achieved by calling "cu-

[1] This function inherit the C++ function "transform_reduce()"
[2] Refer to the website http://thrust.github.com/.

N Assets	C++	Matlab	GPU	std err	speed up C++	speed up Matlab
2	104.065s	32.551s	0.358s	0.004	x290	x91
4	186.280s	49.701s	0.637s	0.005	x293	x78
8	340.044s	81.815s	1.193s	0.003	x285	x69

Table 5.1: Speed up comparison of different asset number

rand_uniform_double()".

Some main part of the code is listed in the Appendix.

In the following we calculate basket call options price which is the same contract we did at the end of Chapter 3. The payoff function is the same as (3.36). All the simulations are run with identical parameters: $\beta_i = -0.2$, $\delta_i = 2$, $\gamma_i = 0.2$, $S_0^i = 100$, $T = 1$ and $K = 80$. All assets are equal weighted.

The first table is a table constructed with different number of assets, each trial with 100000 paths. We can see a very high speed up for our simulations. The second table for all trials we all have 3 assets but the number of path will be different. As we can see from the table as path number increases, the speed up also increases. If you observe more carefully the data, we can also observe that every time N increases 10 times Mathlab running time increases 10 times whereas GPU running time increases by approximately ln(10) times.

There is another great effect that can be achieved by GPU implementation: we can actually simulate an index asset by asset. For this we do not offer a table for that, but it is tried that with 64 assets and 10000 paths we can simulate a result for only 0.918 second! Various indexes have around a hundred stocks on it which is well within the range of the capacity of a graphic card. What can be done is that we can calibrate each asset in the index for its parameter. This calibration can also be implemented with GPU or with the newly launched C++11 library for multithreading. If well implemented the running time should also be short.

N Paths	C++	Matlab	GPU	std err	speed up C++	speed up Matlab
100	0.181s	0.046s	0.004s	0.41	x45	x12
1000	1.448s	0.425s	0.013s	0.07	x111	x33
10000	14.317s	4.146s	0.054s	0.02	x265	x77
100000	143.432s	40.738s	0.501s	0.006	x286	x82

Table 5.2: Speed up comparison of different number of paths

Chapter 6

Conclusions

In this project, we have given a framework to the pricing method for the multivariate asset models. We applied the Fast Fourier Transformation method to the calculation of the derivative prices and compared it with Monte Carlo simulation to verify the consistence between these two methods. We have also deduced a hedging method based on locally risk minimization and assessed two other hedging theory based on other theoretical set up. We finished this project by Monte Carlo simulation. We have also implemented the simulation on C++, Matlab and with GPU respectively. The comparison of three approaches was given and the result is impressive – the running time is reduced by almost two orders.

Appendix A

The exact calculation of equation (3.20):

$$
\begin{aligned}
\hat{\phi}(u) &= \int \cdots \int_{\mathbb{R}^n} e^{-iu\cdot\mathbf{x}}\phi(\mathbf{x})d^n x \\
&= \int_{-\infty}^{-\log\omega_1} e^{-iu_1x_1} \int_{-\infty}^{log(1-\omega_1e^{x_1})} e^{-iu_2x_2} \cdots \int_{-\infty}^{log(1-\omega_1e^{x_1}-\ldots-\omega_ne^{x_n})} e^{-iu_nx_n} \\
&\quad (1-\omega_1e^{x_1}-\omega_2e^{x_2}-\ldots-\omega_ne^{x_n})dx_1dx_2\ldots dx_n \\
&= \int_{-\infty}^{-\log\omega_1} e^{-iu_1x_1} \int_{-\infty}^{log(1-\omega_1e^{x_1})} e^{-iu_2x_2} \cdots \int_{-\infty}^{log(1-\omega_1e^{x_1}-\ldots-\omega_{n-2}e^{x_{n-2}})} \\
&\quad e^{-iu_{n-1}x_{n-1}}(1-\omega_1e^{x_1}-\omega_2e^{x_2}-\ldots-\omega_{n-1}e^{x_{n-1}})^{1-iu_n}[\frac{1}{-iu_n}-\frac{1}{1-iu_n}]dx_1dx_2\ldots dx_{n-1} \\
&= \frac{1-iu_n+i\omega_nu_n}{(1-iu_n)(-iu_n)} \int_{-\infty}^{-\log\omega_1} e^{-iu_1x_1} \int_{-\infty}^{log(1-\omega_1e^{x_1})} e^{-iu_2x_2} \cdots \int_{-\infty}^{log(1-\omega_1e^{x_1}-\ldots-\omega_{n-2}e^{x_{n-2}})} \\
&\quad e^{-iu_{n-1}x_{n-1}}(1-\omega_1e^{x_1}-\omega+2e^{x_2}-\ldots-\omega_{n-1}e^{x_{n-1}})^{1-iu_n}dx_1dx_2\ldots dx_{n-1}.
\end{aligned}
$$

Let $t = \frac{\omega_{n-1}e^{x_{n-1}}}{1-\omega_1 e^{x_1}-...-\omega_{n-2}e^{x_{n-2}}}$ so $dt = \frac{\omega_{n-1}e^{x_{n-1}}}{1-\omega_1 e^{x_1}-...-\omega_{n-2}e^{x_{n-2}}}dx_{n-1}$. Then, we have

$$\begin{aligned}
\hat{\phi}(u) &= \omega_{n-1}^{iu_{n-1}} \frac{1-iu_n+i\omega_n u_n}{(1-iu_n)(-iu_n)} \int_{-\infty}^{-\log\omega_1} e^{-iu_1x_1} \int_{-\infty}^{\log(1-\omega_1 e^{x_1})} e^{-iu_2x_2} \dots \\
&\quad \int_0^1 (1-\omega-1e^{x_1}-...-\omega_{n-2}e^{x_{n-2}})^{1-iu_n-iu_{n-1}} t^{-1-iu_{n-1}}(1-t)^{1-iu_n} dx_1dx_2...dt \\
&= \omega_{n-1}^{iu_{n-1}}(1-iu_n+i\omega_n u_n)\frac{B(-iu_n,2-iu_n)}{(1-iu_n)(-iu_n)} \int_{-\infty}^{-\log\omega_1} e^{-iu_1x_1} \int_{-\infty}^{log(1-\omega_1 e^{x_1})} e^{-iu_2x_2} \\
&\quad \dots \int_{-\infty}^{log(1-\omega_1 e^{x_1}-...-\omega_{n-3}e^{x_{n-3}})} e^{-iu_{n-2}x_{n-2}}(1-\omega_1 e^{x_1}-... \\
&\quad -\omega_{n-2}e^{x_{n-2}})^{1-iu_n-iu_{n-1}} dx_1dx_2...dx_{n-2} \\
&= \omega_{n-1}^{iu_{n-1}}(1-iu_n+i\omega_n u_n)\frac{\Gamma(-iu_n)\Gamma(-iu_{n-1})}{\Gamma(2-iu_{n-1}-iu_n)} \int_{-\infty}^{-\log\omega_1} e^{-iu_1x_1} \int_{-\infty}^{\log(1-\omega_1 e^{x_1})} e^{-iu_2x_2} \dots \\
&\quad \int_{-\infty}^{log(1-\omega_1 e^{x_1}-...-\omega_{n-3}e^{x_{n-3}})} e^{-iu_{n-2}x_{n-2}}(1-\omega_1 e^{x_1}-... \\
&\quad -\omega_{n-2}e^{x_{n-2}})^{1-iu_n-iu_{n-1}} dx_1dx_2...dx_{n-2}
\end{aligned}$$

By repeatedly doing this, we arrive at the result as follows:

$$\hat{\phi}(u) = (1-iu_n+i\omega_n u_n)\frac{\prod_{k=1}^n \Gamma(-iu_k)}{\Gamma(2-i\sum_{k=1}^n u_k)} \prod_{k=1}^{n-1} \omega_k^{iu_k}. \tag{1}$$

Appendix B

Matlab code for Inverse Gaussian random number generator

```
function f = InvGauRnd(m,n,mu,lambda)
//m,n are dimension of the resulting matrix.
A = randn(m,n);
B = ones(m,n);
muMatri = mu*B;
Y = A.*A;
X = muMatri + (mu^2/2/lambda).*Y-mu/2/lambda.
*sqrt(4*mu*lambda.*Y+mu^2.*(Y.*Y));
U = rand(m,n);
bar = muMatri./(X+muMatri);
index = U<= bar;
r1 = index.*X;
r2 = (B-index).*(muMatri.*muMatri./X);
f=r1+r2;
end
```

Matlab program for the simulation kernel:

```
function f = SimuKer(S0,K,T,beta,delta,gamma,aa,b,weight)
n = round(T*252);
A = NIGMultiSimu(n,T,beta,delta,gamma,aa,b);
B = cumsum([S0';A]);
index = B<0;
re = zeros(1,2);
```

```
if index == zeros(size(B))

    re(1,1) = posi(B(n,:)*weight'-K);
    re(1,2) = 1;
end
f = re;
end
function f = posi(a)
    if a>0
        f=a;
    else
        f=0;
    end
end
```

This is the Inverse Gaussian random number generator with CUDA-thrust

```
__device__
double generatorIG(double mu,double lambda,
double nRand,double uRand)
{
double x = mu + mu*mu*nRand*nRand/2/lambda-mu/2/lambda*
sqrtf(4*mu*lambda*nRand*nRand+mu*mu*nRand*
nRand*nRand*nRand);
double bar = mu/(x+mu);
return (uRand<bar)? x:(mu*mu/x);
}
```

Note that the "device" indicates that this program is in the kernel and runs on the device.

This is the part that we calculate the weighted sum of the simulated portfolio price.

```
for(unsigned int i = 0; i < N; ++i)
{
  delZ = generatorIG(a,b,curand_normal_double(&s),
```

```
  curand_uniform_double(&s));
  for(unsigned int k = 0; k < AssetCount; ++k)
  {
  randholder = (Assets[k].gamma)*delZ+generatorIG(Assets[k]
  .mu, Assets[k].lambda,curand_normal_double(&s),curand_
  uniform_double(&s));
  sum += Weight[k]*(Assets[k].sigma*curand_normal_double(&s
  )*sqrt(randholder)+Assets[k].theta*randholder);
  }
}
```

This is the function that we used to realize the parallel sum.

```
double estim = thrust::transform_reduce(
thrust::counting_iterator<int>(0),
thrust::counting_iterator<int>(M),
estimate(assetArray,weightArray,initialArray,
1.0/daysInyearf,b*b*b/a/daysInyearf/daysInyearf,
assets.size(),Time,strike),
0.0f,
thrust::plus<double>());
```

C++ program for the Monte Carlo Simulation.

```
#include<iostream>
#include<numeric>
#include<list>
#include<vector>
#include<random>

using namespace std;

const int daysInYear = 252;
const double a = 1.2;
const double b = 1.0;
list<double> SimuRec;
```

```
//Holder of Parameters
struct ParaNIG{
   double S0, mu, lambda, sigma, theta, gamma, weight, Strike;
   ParaNIG(double _S0, double _mu, double _lambda, double
   _sigma, double _theta, double _gamma, double _weight,
   double _strike) : S0(_S0), mu(_mu), lambda(_lambda), sigma
   (_sigma), theta(_theta), gamma(_gamma), weight(_weight),
   Strike(_strike){};
};

//Generator of Inverse Gaussian random number
double InvGausGen(double mu, double lambda, double nRand,
double uRand){
        double x = mu + pow(mu,2)*pow(nRand,2)/(2*lambda)-mu/
(2*lambda)*sqrt(4*mu*lambda*pow(nRand,2)+pow(mu,2)*pow(nRand,4));
        double bar = mu/(x+mu);
        return (uRand<bar)? x:(pow(mu,2)/x);
}

//One path of simulation
double SimuOnce(int AssetNum, double T, double strike,
vector<ParaNIG>&para){
        random_device rd;
        mt19937 gen(rd());
        uniform_real_distribution<> uRand(0,1);
        normal_distribution<> nRand(0,1);

        unsigned int N = (int)daysInYear*T;
        double delZ, randholder, sum=0;

        for(int i=0;i<AssetNum;++i){
        sum=sum+para[i].weight*para[i].S0;
```

```
        }

        //Path generator generating of a path of N steps
        for(unsigned int i = 0;i < N; ++i){
                delZ = InvGausGen(a,b,nRand(gen),uRand(gen));
        for(int k = 0; k < AssetNum; ++k){
    randholder = para[k].gamma*delZ+InvGausGen(para[k].mu,
    para[k].lambda,nRand(gen),uRand(gen));
        sum    = sum + para[k].weight*(para[k].sigma*nRand(gen)
    *sqrt(randholder)+para[k].theta*randholder);
                }
        }

        return (sum-strike>0)?(sum-strike):0;

}

int main(){

        int AssetNum;
        unsigned int PathNum;
        double T, strike;

        cout<<"Please input number of assets, strike,
time period and number of paths"<<endl;
        cin>>AssetNum>>strike>>T>>PathNum;



        cout<<"Fast mode: All assets will be the same.
Complete mode: assets shall be inserted one by one."
<<endl;
        cout<<"Fast mode input 1; Complete mode input 2."
<<endl;
```

```
        int mode;
        vector<ParaNIG> para;
        double S0, beta, delta, gamma, weight;
        cin>>mode;
        //This part is for speeding up the input process
(since it is multidimensional in the fast mode we presume
all the dimensions are identical)
        switch(mode){
                //mode where all dimensions are equal
                case 1:
        cout<<"Please input in sequence S0, beta, delta,
        gamma"<<endl;
        cin>>S0>>beta>>delta>>gamma;
        for(int i=0;i<AssetNum;++i){
        para.push_back(ParaNIG(S0,1.0/daysInYear, pow(b,3)
        /(pow(gamma,3)*pow(daysInYear,2)*(1-a*gamma)), delta,
        pow(delta,2)*beta, pow(gamma,2), 1.0/AssetNum,
        strike));
        }
        break;
                //mode where you can input the parameter for
        each of the dimensions
                case 2:
        for(int i=0;i<AssetNum;++i){
        cout<<"Please input in sequence the following
variables for asset"<<i+1<<endl;
        cout<<"S0 beta delta gamma weight"<<endl;
        cin>>S0>>beta>>delta>>gamma>>weight;
        para.push_back(ParaNIG(S0, 1.0/daysInYear, pow(b,3)
        /(pow(gamma,3)*pow(daysInYear,2)*(1-a*gamma)), delta,
        pow(delta,2)*beta, pow(gamma,2),weight,strike));
        }
        break;
```

```
        default:
        cout << "Value Unknown"<<endl;
        }

        double holder=0;
        for(unsigned int i = 0; i<PathNum; ++i){
                holder = SimuOnce(AssetNum,T,strike, para);
                SimuRec.push_back(holder);
        }
        //this is from the <numeric> of the STL to get the
    sum of each term in the list
        double result = accumulate(SimuRec.begin(),
    SimuRec.end(),0.0);
        cout<<result<<endl;
        cin>>mode;

}
```

References

[1] O.E. Barndorff-Nielsen, J. Pedersen, and K.I. Sato. Multivariate subordination, self-decomposability and stability. *Advances in Applied Probability*, **33**, 2001. 15

[2] M.P. Bavouzet and M. Messaoud. Computation of greeks using malliavin's calculus in jump type market models. *Institut National de Recherche en Informatique et en Automatique Rapport de Recherche*, 2005. 35, 36

[3] D. Bayazit and C.A. Nolder. Malliavin calculus for levy markets and new sensitivities. 2009. 35, 37

[4] P. Carr and D.B. Madan. Option valuation using the fast fourier transform. *Journal of COmputational Finance*, **2**:753–778, 1998. 3

[5] R. Cont and P. Tankov. *Financial Modelling with Jump Processes*. CHAPMAN HALL/CRC, New York, 2004. 3, 10, 11

[6] R. Cont, P. Tankov, and E. Voltchkova. Hedging with options in models with jumps. *Proceedings of the Abel Symposium in honor of Kiyosi Ito*, 2005. 34

[7] B. Dupire. Pricing with a smile. *RISK*, **7**:18–20, 1994. 1

[8] H. Föllmer and M. Schweizer. Hedging of contingent claims under incomplete information. *Applied Stochastic Analysis*, **5**, 1991. 34

[9] H. Gerber and E. Shiu. Option pricing by esscher transforms. *Transactions of Society of Actuaries*, **46**, 1994. 12

[10] T.R. Hurd and Z. Wei. A fourier transform method for spread option pricing. *Working paper McMaster University*. 4, 15, 17

[11] K.R. Jackson, S. Jaimungal, and V. Surkov. Fourier space time-stepping for option pricing with lévy models. 2008. Department of Computer Science and Department of Statistics, University of Toronto. 16

[12] N. Jacob. *Pseudo Differential Operators and Markov Processes*. Imerial College Press, 2002. 30, 33

[13] R. Kawai and A. Kohatsu-Higa. Computation of greeks and multi-dimensional density estimation for asset price models with time-changed brownian motion. 2009. 35, 37

[14] D.B. Kirk and W.W. Hwu. *Programming Massively Parallel Processors: A Hands-on Approach*. Elsevier, 2010. 44

[15] A. Lewis. A simple option formula for general jump-diffusion and other exponential lévy processes. 2001. http://www.optioncity.net. 19

[16] E. Luciano and P. Semeraro. Multivariate time changes for lévy asset models: Characterization and calibration. *Journal of Computational and Applied Mathematics*, **233**:1937–1953, 2010. 4, 15, 23, 25

[17] J. Michael, W. Schucany, and R. Haas. Generating random variates using transformations with multiple roots. *The American Statistician*, **30:88-90**, 1976. 41

[18] N.C. Petroni and P. Sabino. Multidimensional quasi-monte carlo malliavin greeks. 2011. 35, 37

[19] J.A. Primbs. Dynamic hedging of basket options under proportional transaction costs using receding horizon control. *International of Journal*, 2009. 35, 36

[20] J.A. Primbs and C. Sung. Stochastic receding horizon control of contrained linear systems with state and control multiplicative noise. *IEEE Transactions on Automatic Control*, 2009. 35

[21] R. Rebonato. *Volatility and Correlation in the Pricing of Equity.* Wiley, Chichester, 1999. 1

[22] N. Reich, C. Schwab, and C. Winter. On kolmogorov equations for anisotropic multivariate lévy processes. Research Report, Seminar für Angewandte Mathematik Eidgenössische Technische Hochschule, 2008. 3

[23] C. Winter. *Wavelet Galerkin schemes for option pricing in multidimensional Lévy models.* PhD thesis, ETH Zürich, 2009. 3