Multilevel Monte Carlo For Exponential Lévy Models

Authors: Rawane MBODJ, Nicolas NAJBURG, Jonathan OURAGA, Yanyang ZHU

Supervisor: Guillaume COQUERET

Assessor: Bertrand MAILLET

Presented: March 26th, 2019

Abstract : The main objective of our work is to replicate a very recent study conducted by Michael B. Giles and Yuan Xia (2017) which analyse the efficiency of a Multilevel Monte Carlo method on three types of exotic options, namely Asian, Lookback and Barrier options and three Lévy models, namely Variance Gamma (VG), Normal Inverse Gaussian (NIG) and negative α-stable processes. In the following work you will have a brief summary of the paper of Giles that served as main support for this project, we then suggest a critical analysis of the paper before proposing an extension by using the multilevel method with Exponential Lévy processes to analyse real life pricing problem.

Keywords : Multilevel Monte Carlo · Variance Gamma · Normal Inverse Gaussian · Asian options · Lookback options · Barrier options

early makers | em lyon business school

Table of Contents

I. Introduction

Empirical studies have shown that the most convenient way to value an option whose payoff depends on its historical trajectory is to estimate this trajectory. This estimation is done through simulation of multiple sample paths with some process of random walk that has specific number of steps and whose payoffs are then averaged. A good approximation of the final value of the option is the averaged value obtained discounted at a risk-free rate. That is exactly what the so-called Monte Carlo simulation method does. In fact, this method is one of the most famous and intuitive ways to obtain an estimation of a path-dependent option written on an underlying asset that follows a stochastic process. Therefore, we can assume that the accuracy of the Monte Carlo simulation result is function of the number of sample paths generated. Indeed, and as demonstrated by the Law of Large Number, the higher the number of sample paths and the number of steps embedded in each path, the more the averaged value of these samples' payoffs is close to reality and therefore the better the approximation of the option's value will be. In other words, the greater the number of sample paths N , the smaller the weak error $\mathbb{E}\left[\hat{P}\right] - \mathbb{E}[P]$ (where P is the payoff corresponding to one path and $\hat{P}$ is its approximation such that $\hat{P} \equiv f(\widehat{S_T})$). However, generating sample paths comes with computational cost and the fact that the standard Monte Carlo simulation method allocates evenly on each path the cost of producing samples, makes this method quite expensive computationally spoken. For instance, let's consider the following stochastic differential equation:

$$dS_t = a(S_t, t)dt + b(S_t, t)dW_t \ ,$$

where a and b are constant

Its Mean Square Error (MSE) is

$$N^{-1}\mathbb{V}\left[\hat{P}\right] + \left(\mathbb{E}\left[\hat{P}\right] - \mathbb{E}[P]\right)^2 \approx aN^{-1} + bh^2$$

If we want this MSE to be ε^2 , we need to set the number of paths N such that $N = \Theta(\varepsilon^{-2})$, and the timestep h such that $h = \boldsymbol{\Theta(\varepsilon)}$ which is also the value of the Root Square Mean Error (RSME). By doing so we eventually ends up with a total computational cost of $\Theta(\varepsilon^{-3})$ which represents a heavy computational burden that needs to be reduced. A recent study conducted by Kebaier (2005) Giles MCML Paths simul, which is closely related to the general approach of quasi-control variates, proved that the computational cost of $\Theta(\varepsilon^{-3})$ can be significantly

reduced to $\Theta(\varepsilon^{-2.5})$. To do so the author used the appropriate combination of results obtained using two levels of timestep, h and $\boldsymbol{\Theta}\left(h^{1/2}\right)$. The Two-level Monte Carlo can be used to give an approximation of

$$\mathbb{E}\left[\hat{P}_1\right] = \mathbb{E}\left[\hat{P}_0\right] + \mathbb{E}\left[\hat{P}_1 - \hat{P}_0\right]$$

which can be rewritten using the following estimators

$$\mathbb{E}\left[\hat{P}_1\right] = N_0^{-1} \sum_{n=1}^{N_0} \hat{P}_0^{(O,n)} + N_1^{-1} \sum_{n=1}^{N_1} (\hat{P}_1^{(1,n)} - \hat{P}_0^{(1,n)})$$

We can clearly notice that if $\hat{P}_1 - \hat{P}_0$ is small, the variance of the process decreases as well and we do not need anymore many trajectories to obtain $\mathbb{E}\left[\hat{P}_1 - \hat{P}_0\right]$, therefore the computational cost is reduced significantly.

The Two-level Monte Carlo opens the door to a new model that uses multiple levels $l = 0,1,\dots,L$ in combination with a geometric sequence of different timesteps $h_l = M^{-l}T$ such that $M \geq 2$. This model is the Multilevel Monte Carlo simulation model. The main idea behind that method is that it retains the accuracy/bias associated with the smallest timestep, but it uses calculations with larger timesteps to reduce the variance in a way that minimises the overall computational complexity Giles (2008). Indeed, by implementing this method, it is possible to reduce the computational cost to an incredible $\Theta(\varepsilon^{-2}(\log\varepsilon)^2)$. In fact, let's consider a sequence $\hat{P}_0, \hat{P}_1, \dots, \hat{P}_L$

Where

$$\mathbb{E}\left[\hat{P}_L\right] = \mathbb{E}\left[\hat{P}_0\right] + \sum_{l=1}^{L} \mathbb{E}\left[\hat{P}_l - \hat{P}_{l-1}\right] = N_0^{-1} \sum_{n=1}^{N_0} \hat{P}_0^{(O,n)} + \sum_{l=1}^{L} \{ N_l^{-1} \sum_{n=1}^{N_l} (\hat{P}_l^{(l,n)} - \hat{P}_{l-1}^{(l,n)}) \}$$

and in which there is an independent estimation for each level of correction. If we define C_0 and C_l, to be respectively the cost and variance of $\hat{P}_0$ and $\hat{P}_l - \hat{P}_{l-1}$, and V_0, V_l be respectively the variance of $\hat{P}_0$ and $\hat{P}_l - \hat{P}_{l-1}$, then total cost of the Multilevel Monte Carlo simulation should be

$$\sum_{l=0}^{L} N_l C_l$$

and its total variance should be

$$\sum_{l=0}^{L} N_l^{-1} V_l.$$

By setting the total variance equal to ε^2 gives a total cost of

$$\sum_{l=0}^{L} N_l C_l = \varepsilon^{-2} \left(\sum_{l=0}^{L} \sqrt{V_l C_l}\right)^2$$

which contrasts with a standard cost of approximately

$$\varepsilon^{-2} V_0 C_L .$$

To make the Root Mean Square Error, less than ε, one needs to choose N_l and $\sqrt{V_l C_l}$ so that total variance is less than $\frac{1}{2}\varepsilon^2$ and choose L so that $(\mathbb{E}[\hat{P}_L] - \mathbb{E}[P])^2 < \frac{1}{2}\varepsilon^2$. In order to properly implement a Multilevel Monte Carlo simulation, one should use the following algorithm

- Step 1: We start with $L = 0$
- Step 2: We estimate V_L using an initial set of 10^4 samples
- Step 3: We define the optimal N_l, $l = 0, \dots, L$ using the optimal N_l equation which is

$$N_l = \left[2\varepsilon^4 \sqrt{V_l h_l} \left(\sum_{l=0}^{L} \sqrt{\frac{V_l}{h_l}}\right)\right]$$

- Step 4: We evaluate extra samples at each level as needed for new N_l.
- Step 5: If $L \geq 2$, we should test for convergence using either of the following constraints:
  - $\max\{M^{-1}|\hat{Y}_{L-1}|, |\hat{Y}_L|\} < \frac{1}{\sqrt{2}}(M-1)\varepsilon$ or
  - $|\hat{Y}_L - M^{-1}\hat{Y}_{L-1}| < \frac{1}{\sqrt{2}}(M^2-1)\varepsilon$
- Step 6: If $L < 2$ or it does not converge, we should set $L := L + 1$ and go to step 2.

Further details about the Multilevel Monte Carlo method are provided is the summary section below.

As the standard Monte Carlo simulation has proven to be highly appropriate for path dependent options such as Lookback, Barrier and Asian options we can expect even better

results by implementing the Multilevel Monte Carlo Simulation on these types of option. However, what type of stochastic process would be appropriate for this kind of method ?

Well, the exponential Levy process which assumes that stock prices depend on Levy processes, regroups a wide range of models. While some of these models are built on pure jump processes that can reproduce the stylized features of asset returns such as heavy tails and asymmetric distribution of increments, others do not. Moreover, some allow for pure jump processes with finite activity while others use jump processes with infinite activity. However, it has been empirically proven that pure jump processes of finite activity without a diffusion component fail to generate realistic paths while on the other side those with infinite activity are more precise. Therefore, it appears essential to work with infinite-activity pure jump exponential Lévy models such as models driven by Variance Gamma (VG), Normal Inverse Gaussian (NIG) and α-stable processes which are models that have the particular feature of allowing direct simulation of increments.

The main objective of our work is to replicate a very recent study conducted by Michael B. Giles and Yuan Xia (2017) which focused on three types of exotic options, namely Asian, Lookback and Barrier options and three Lévy models, namely Variance Gamma (VG), Normal Inverse Gaussian (NIG) and α-stable processes. Although many studies have suggested advanced Multilevel Monte Carlo techniques such as the Wiener-Hoof Monte-Carlo method introduced by Kuznetsov (2011) or even the Multilevel Monte Carlo method adapted to Lévy-driven SDEs with payoffs which are Lipschitz with respect to the supremum norm Dereich (2011) this paper focused on a method that approximates discretely monitored maximums based on an uniform timestep discretisation of the Lévy process.

The outline of the work is as follow: First we provide a brief summary of the paper that served as main support for this project. Secondly a critical analysis of the paper in question will be made before proposing an extension to the ideas developed in that paper. Finally, the R codes used will be revealed in the Appendix.

II. Summary of the Paper

This section deals with the summary of the paper entitled "Multilevel Monte Carlo For Exponential Lévy Models" by Michael B. Giles and Yuan Xia (2017). This paper was the main support of our work. The main objective of this paper was to apply the multilevel Monte Carlo method on a various type of exotic options using exponential Lévy models with uniform timestep discretisation. The authors first started by providing a review of the Multilevel Monte Carlo method, before introducing the three Lévy processes, namely Normal Inverse Gaussian, Variance Gamma, and α-stable process, which were considered to conduct the research. Later, the convergence rate of the discretely monitored running maximum of a large class of Lévy processes have a power law behaviour for small jumps and have exponential tails, will be bound as a preparation measure for the multilevel variance of Lookback and Barrier analysis. Finally, the variance of the multilevel estimators are bound and numerical results regarding the application of the multilevel Monte Carlo on Asia, Lookback and Barrier options using the three different exponential Lévy models, are presented.

Multilevel Monte Carlo (MLMC) method

Let's denote by P the payoff of a path-dependent option that follows an exponential Lévy process on the time interval [0,T] and by $\hat{P}$ its approximation using a discretisation with M^l uniform timesteps of size $h_l = M^{-l} T$ on level l, the standard Monte Carlo estimate for the expectation of the approximation aforementioned can be expressed as follow:

$$\mathbb{E}[\hat{P}_l] = \mathbb{E}[\hat{P}_0] + \sum_{l=1}^{L} \mathbb{E}[\hat{P}_l - \hat{P}_{l-1}]$$

The standard Monte Carlo estimate for $\mathbb{E}[\hat{P}_l - \hat{P}_{l-1}]$ using N_l independent paths can be expressed as follow:

$$\hat{Y}_l = N_l^{-1} \sum_{i=1}^{N_l} (\hat{P}_l^{(i)} - \hat{P}_{l-1}^{(i)})$$

One key advantage of the MLMC is that it takes advantage of the fact that the variance

$V_l := \mathbb{V}[\hat{P}_l - \hat{P}_{l-1}]$ decreases with l and selects N_l to minimise the computational cost to achieve an appropriate Root Mean Square Error (RMSE). This can be explained by the following theorem.

Theorem: *If there exist independent $\hat{Y}_l$ based on number N_l of Monte Carlo samples, each with complexity C_l and positive constants $\alpha, \beta, c_1, c_2, c_3$ such that $\alpha \geq \frac{1}{2}, \min(1, \beta)$ and*

$$\left|\mathbb{E}\left[\hat{P}_l - P\right]\right| \leq c_1 h_l^{\alpha}$$

$$\mathbb{E}\left[\hat{Y}_l\right] = \begin{cases} \mathbb{E}\left[\hat{P}_0\right], & l = 0 \\ \mathbb{E}\left[\hat{P}_l - P\right], & l > 0 \end{cases}$$

$$\mathbb{V}\left[\hat{Y}_l\right] \leq c_2 N_l^{-1} h_l^{\beta}$$

$$C_l \leq c_3 N_l h_l^{-1}$$

There is a positive constant c_4 such that for any $\varepsilon < e^{-1}$, there are values L and N_l for which the multilevel estimator $\hat{Y} = \sum_{l=0}^{L} \hat{Y}_l$, has a mean square error with bound:

$$MSE = \mathbb{E}\left[\left(\hat{Y} - \mathbb{E}[P]\right)^2\right] < \varepsilon^2$$

and with a computational complexity C with bound

$$C \leq \begin{cases} c_4 \varepsilon^{-2}, & \beta > 1, \\ c_4 \varepsilon^{-2} (log\varepsilon)^2, & \beta = 1 \\ c_4 \varepsilon^{-2-(1-\beta)/\alpha}, & 0 < \beta < 1 \end{cases}$$

Lévy Models

The following three models were used:

Variance Gamma (VG) model

The main advantages of a VG process X_t with parameters (σ, θ, κ) include the fact that it allows for the fitting of the skewness and kurtosis of the stock returns and the fact that it is easily simulated as it can be represented as $X_t = \theta G_t + \sigma B_{G_t}$ in which B_t is a Brownian process and G_t is a Gamma process with parameters $(\frac{1}{\kappa}, \frac{1}{\kappa})$. After conducting a calibration Schoutens (2003) obtained the different values:

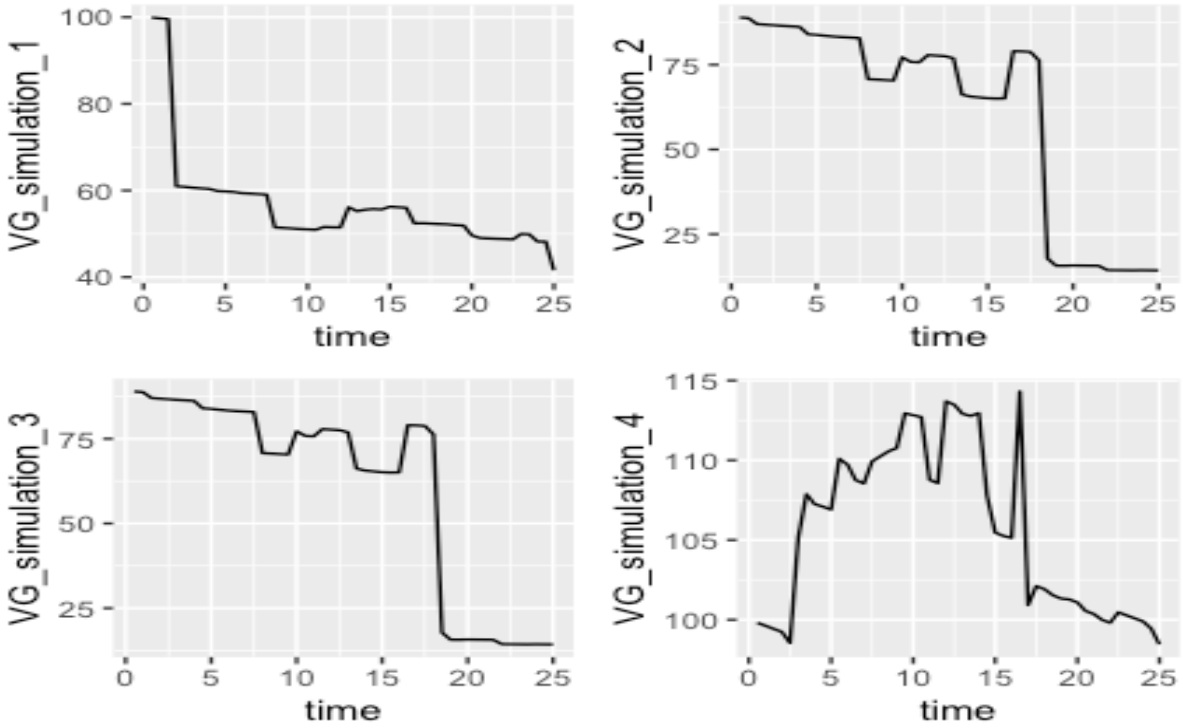


Figure 1 : Stock price simulation under Variance Gamma process

$\sigma = 0.1213, \theta = -0.1436, \kappa = 0.1686$

Normal Inverse Gaussian (NIG)

A NIG process X_t with parameters (σ, θ, κ) can be represented as follow:

$$X_t = \theta I_t + \sigma B_{I_t},$$

where the subordinator I_t represents an Inverse Gaussian process with parameters $(\frac{1}{\kappa}, 1)$.

The calibration conducted by Schoutens (2003) shows that values of the NIG parameters should be :

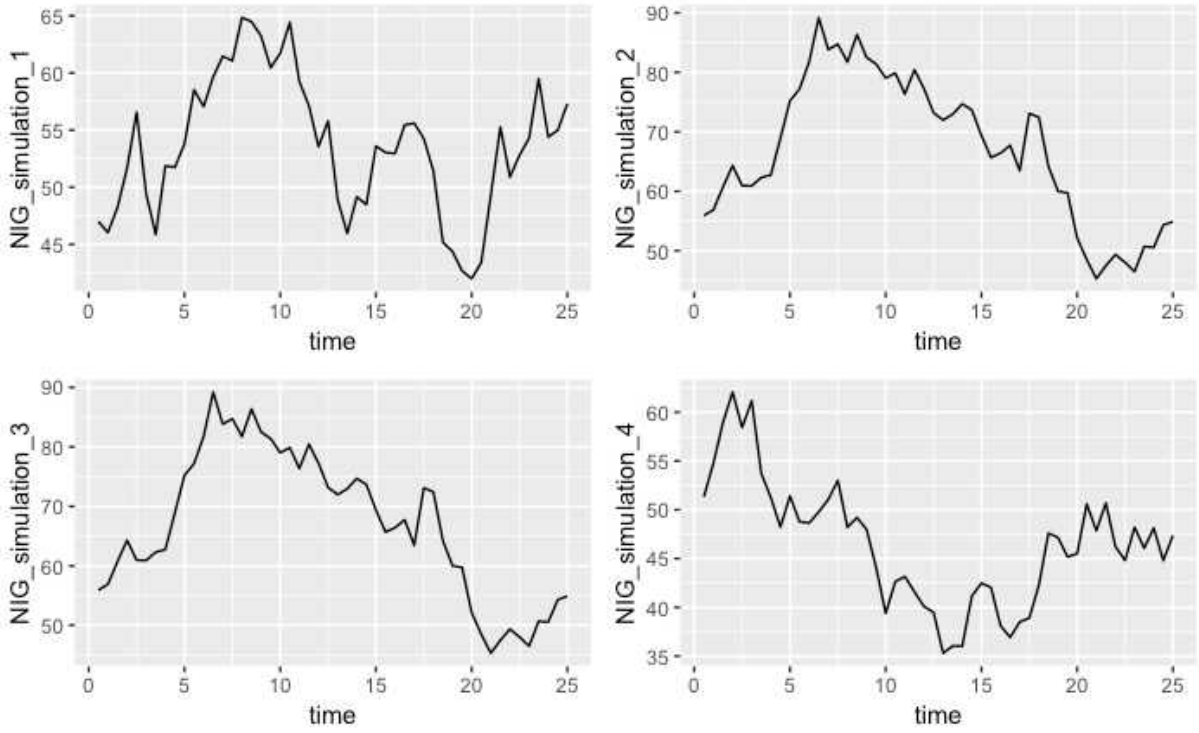


Figure 2 : Stock simulation under Normal Inverse aussian process

$\sigma = 0.1836$, $\theta = -0.1313$, $\kappa = 1.2819$ with a risk-free rate of $r = 0.05$

Spectrally negative α- stable process

A Spectrally negative α- stable process is a Lévy process of the form:

$$v(x) = \frac{B}{|x|^{\alpha+1}} 1_{\{x<0\}}$$

With $0 < \alpha < 2$ and a non-negative B . The process itself has a characteristic function as described below:

$$\mathbb{E}\left[\exp(iuX_t)\right] = \exp\{-t\, B^{\alpha}|u|^{\alpha}(1 + isgn(u)\tan\tfrac{\pi\alpha}{2})\}, \text{ if } \alpha \neq 1$$

$$\mathbb{E}\left[\exp(iuX_t)\right] = \exp\{-t\, B|u|(1 + i\tfrac{2}{\pi} sgn(u)\log|u|)\}, \text{ if } \alpha = 1$$

Where $sgn(u) = \frac{|u|}{u}$ if $u \neq 0$ and $sgn(0) = 0$ as demonstrated by Kyprianou(2006).

The following parameters values were used: $\alpha = 1.5597$ and $B = 0.1486$

Key Numerical Results

The variance D_n of the multilevel correction can be represented as follow:

$$D_n = \sup X_t - \max X_{i/n} \text{ with } i = O, 1, \dots, n \text{ and } O \leq t \leq 1$$

To obtain the order of weak convergence for lookback-type payoffs, we need $\mathbb{E}[D_n]$. Chen (2011) showed that $\mathbb{E}[D_n]$ can take the following values:

$$\text{If } \sigma > 0, \mathbb{E}[D_n] = \boldsymbol{\Theta}\left(\frac{1}{\sqrt{n}}\right);$$

If $\sigma = 0$ and X_t is of finite variation, i.e. $\int_{|x|<1} |x|\, v(dx) < \infty$

$$\mathbb{E}[D_n] = \boldsymbol{\Theta}\left(\frac{\log n}{\mathrm{n}}\right);$$

This correspond to the VG process.

If $\sigma = 0$ and X_t is an infinite variation, then

$$\mathbb{E}[D_n] = \boldsymbol{\Theta}\left(n^{-\frac{1}{\beta}+\delta}\right);$$

This correspond to the NIG process.

Where $\boldsymbol{\beta} = \inf\{\alpha > 0: \int_{|x|<1} |x|^{\alpha}\, v(dx) < \infty\}$ is the Blumenthal-Getoor index of X_t and

$\delta > 0$ is an arbitrarily small strictly positive constant.

As the paper focuses on infinite activity jump processes, the authors looked at the L^p convergence rate of D_n for pure jump processes.

Financial Products

<u>Asian options</u>

An estimation of average value of S is defined using the trapezoidal approximation as follow:

$$\bar{\bar{S}} := S_0 T^{-1} \sum_{j=0}^{n-1} \frac{1}{2} h(\exp(X_{jh}) + \exp(X_{(j+1)h})),$$

And the approximated payoff is: $\hat{P} = P(\bar{\bar{S}})$

Based on the Lipschitz property, the weak convergence for the numerical approximation can be expressed as follow:

$$\left|\mathbb{E}\left[\hat{P}_l - P\right]\right| \leq S_K \mathbb{E}\left[\left|\bar{\bar{S}} - \bar{S}\right|\right] \leq L_K \left(\mathbb{E}\left[\left(\bar{\bar{S}} - \bar{S}\right)^2\right]\right)^{1/2}$$

The convergence of the MLMC variance can be expressed as:

$$V_l \leq 2L_K^2 \mathbb{E}\left[\left(\bar{\bar{S}}_l - \bar{S}\right)^2\right] + 2L_K^2 \mathbb{E}\left[\left(\bar{\bar{S}}_{l-1} - \bar{S}\right)^2\right]$$

Lookback options

The authors worked with a lookback put option whose bounded payoff is expressed as:

$$P = \exp(-rT)(K - S_0 \exp(m))^+ \text{ with } m = sup_{0<t<T} X_t.$$

Because of the Lipschitz property, the weak convergence of the processes is the same as the one previously discussed. However, for the Lipschitz lookback put payoff the multilevel variance convergence rate is different according to the underlying Lévy process faced. Indeed,

If X_t is a Variance Gamma process, then

$$V_l = \boldsymbol{\Theta}(h_l);$$

If X_t is a Normal Inverse Gaussian process, then

$$V_l = \boldsymbol{\Theta}(h_l|log h_l|);$$

If X_t is a spectrally negative α-stable process with $\alpha > 1$, then

$$V_l = \boldsymbol{\Theta}\left(h_l^{\frac{2}{\alpha}-\delta}\right),$$

for any small $\delta > 0$.

Barrier options

The paper considered a bounded up-and-out barrier option whose payoff can be expressed as follow:

$$P = \exp(-rT) f(S_T) \mathbf{1}_{\{\sup_{0<t<T} S_T < B\}} = \exp(-rT) f(S_T) \mathbf{1}_{\{m < \log(\frac{B}{S_0})\}}$$

where $m = sup_{0<t<T} X_t$ and $|f(x)| \leq F$ is bounded. Using the estimation of $\hat{m}_l$, the numerical approximation of the payoff is:

$$\widehat{P_l} = \exp(-rT) f(S_T) \mathbf{1}_{\{\widehat{m_l} < \log(\frac{B}{S_0})\}}$$

With $\widehat{m_l} = max_{0 \le i \le M^l} X_{ihl}$

III. Numerical Results

For each Lévy process and each option type, the paper presents numerical results on the Multilevel Monte Carlo method efficiency with a set of four plots. The idea of this part is not to analyse each one of these as it is already done in the paper, but more to compare the graphs for simple Geometric Brownian Motion and Variance Gamma process in the case of Asian option. The R code used for these figures can be found at the end of this report. Let's start by recalling for the three option types, the form of the numerical payoff.

Numerical Payoff

Asian option

For the purpose of this research an arithmetic Asian call option was considered. Its discounted payoff is as follow:

$P = \exp(-rT)\max(0, \bar{S} - K)$ with $T = 1, r = 0.05, S_0 = 100, K = 100$ and

$$\bar{S} = S_0 T^{-1} \int_0^T \exp(X_t) dt$$

Note that for a general Lévy process, as it is not easy to directly sample, the integral process, the trapezoidal approximation was used:

$\bar{\bar{S}} := S_0 T^{-1} \sum_{j=0}^{n-1} \frac{1}{2} h(\exp(X_{jh}) + \exp(X_{(j+1)h}))$,

where $n = T/h$ is the number of steps and the payoff approximation is:

$$\hat{P} = \exp(-rT) \max(0, \bar{\bar{S}} - K)$$

Lookback option

The Lookback option studied in the paper is a put option on the floating underlying

$P = \exp(-rT)(K - S_0 \exp(m))^+$ with $m = sup_{0<t<T} X_t$

The important values were the following: $T = 1$, $r = 0.05$, $S_0 = 100, K = 110.$

For simplicity, the discretely monitored maximum was used as an approximation,

$\hat{P}_l = \exp(-rT)(K - S_0 \exp(\hat{m}_l))^+$, with $\hat{m}_l = \max_{0<t<T} X_{jh_l}$

Barrier option

The barrier option considered was an up-and-out call with the following payoff:

$$P = \exp(-rT) f(S_T) \mathbf{1}_{\{\sup_{0<t<T} S_T < B\}} = \exp(-rT) f(S_T) \mathbf{1}_{\{m < \log(\frac{B}{S_0})\}}$$

where $m = sup_{0<t<T} X_t$ with $T = 1, r = 0.05, S_0 = 100, K = 100, B = 115.$

The discretely monitored approximation gives the following:

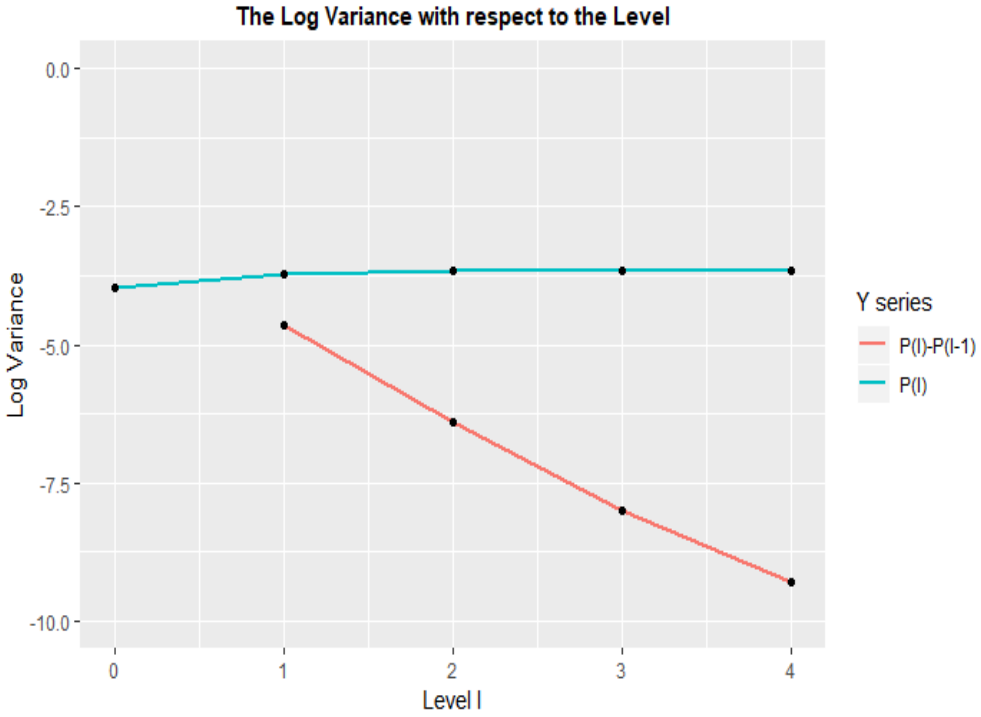
$\widehat{P_l} = \exp(-rT) f(S_T) \mathbf{1}_{\{\widehat{m_l} < \log(\frac{B}{S_0})\}}$ with $\widehat{m_l} = max_{0 \le j \le n_l} X_{ihl}$

Plots

Variance Behaviour

The first plot represents the logarithm in base M, M being the basis number of points for level 1, of the variance with respect to the level L. This choice is motivated by the idea that a slope of – 1 corresponds to a variance proportional to M^{-l}, which turns out to be h_l. Two curves are represented in this graph: $\widehat{P_l}$ - $\widehat{P_{l-1}}$ for the Multilevel Monte Carlo and $\widehat{P_l}$ for a simple Monte Carlo.

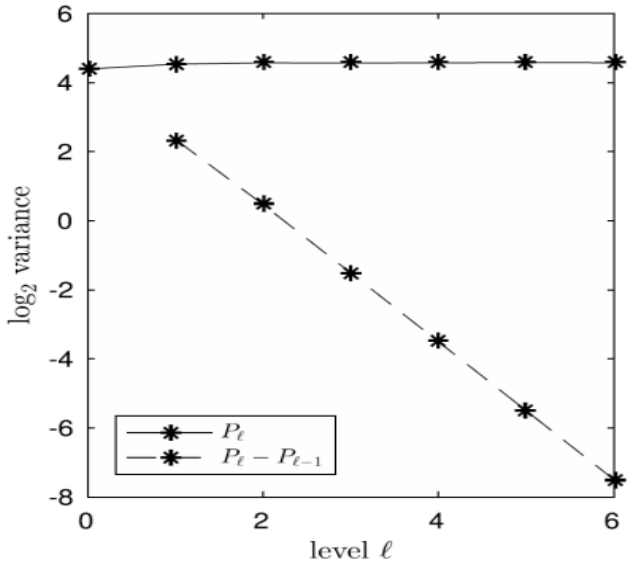
For a Geometric Brownian Motion applied on an Asian Option, the graph is the following:



We can clearly notice here the decreasing shape of the variance for the Multilevel Monte Carlo while the Standard Monte Carlo seems to be independent of the level. In fact, the slope of $\widehat{P_l}$ - $\widehat{P_{l-1}}$ is approximately -1, indicating that $V_l = V[\widehat{P_l} - \widehat{P_{l-1}}] = \Theta(h)$.

For the level 4, V_l is more than 1,000 times smaller than the variance $V[\widehat{P_l}]$ of the standard Monte Carlo method with the same timestep.

Let's now compare this graph with the one obtained in the paper with a Variance Gamma Process on an Asian Option.

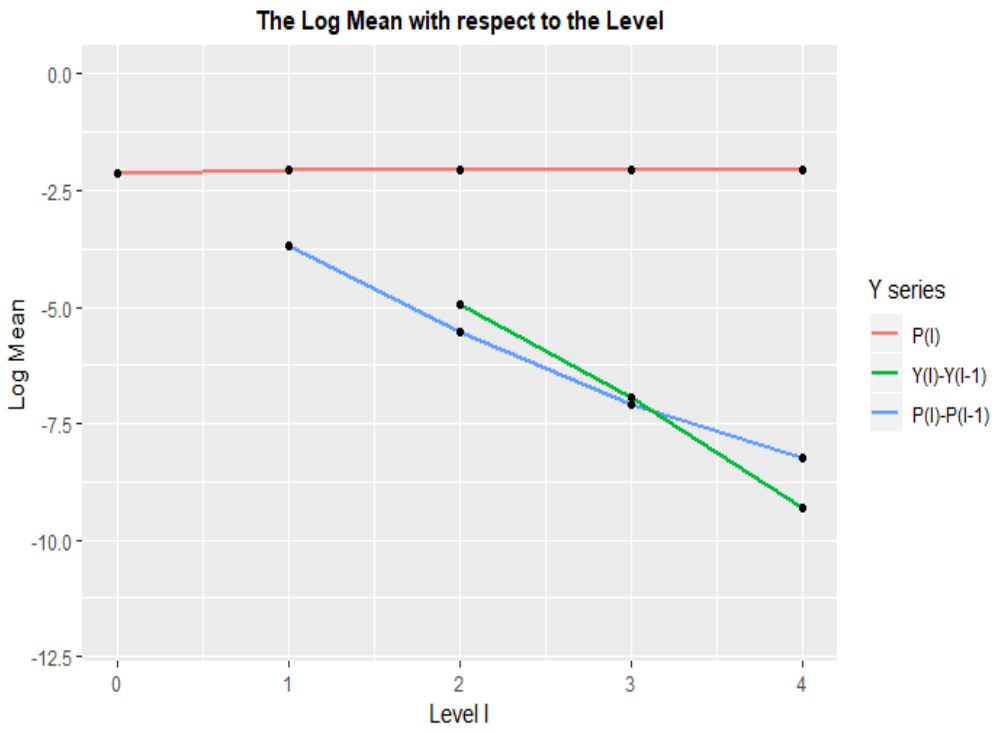


In comparison with the previous graph, the values were moved upward, with for instance a variance of the simple Monte Carlo around 4 whereas it was around -2.5 for the Geometric Brownian Motion. The slope is more about -2 here indicating a second order weak convergence. With Standard Monte Carlo, the variance is independent from the level just like in the previous graph.

Mean Behaviour

The second plot represents once again the logarithm of the mean with respect of the level. The curves are the same as in the Variance graph, but we have added the correction given by $Y(l) - Y(l-1)$, which is the adjustment of the option payoff given by the new level.

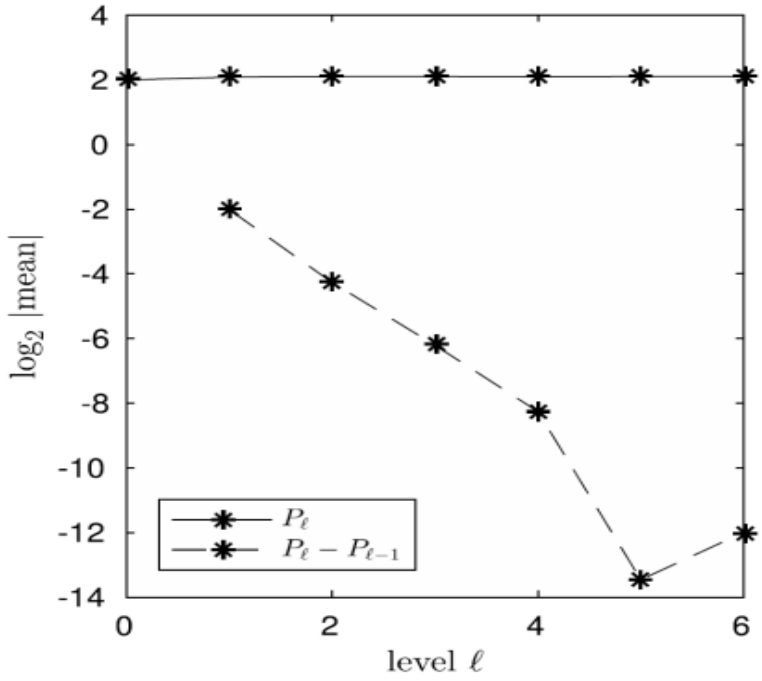
For the Geometric Brownian Motion process, the plot is based on $4 * 10^6$ paths.



The slope is approximately -1 again implying an $\Theta(h)$ convergence of $\mathrm{E}[\widehat{P_l} - \widehat{P_{l-1}}]$.

Another interesting thing to notice is that at level 3, the relative error $\mathrm{E}[\mathrm{P} - \widehat{P_l}]/E[P]$ is less than 10^{-3}.

The same graph for the Variance Gamma process (without the correction curve) has basically the same general shape:

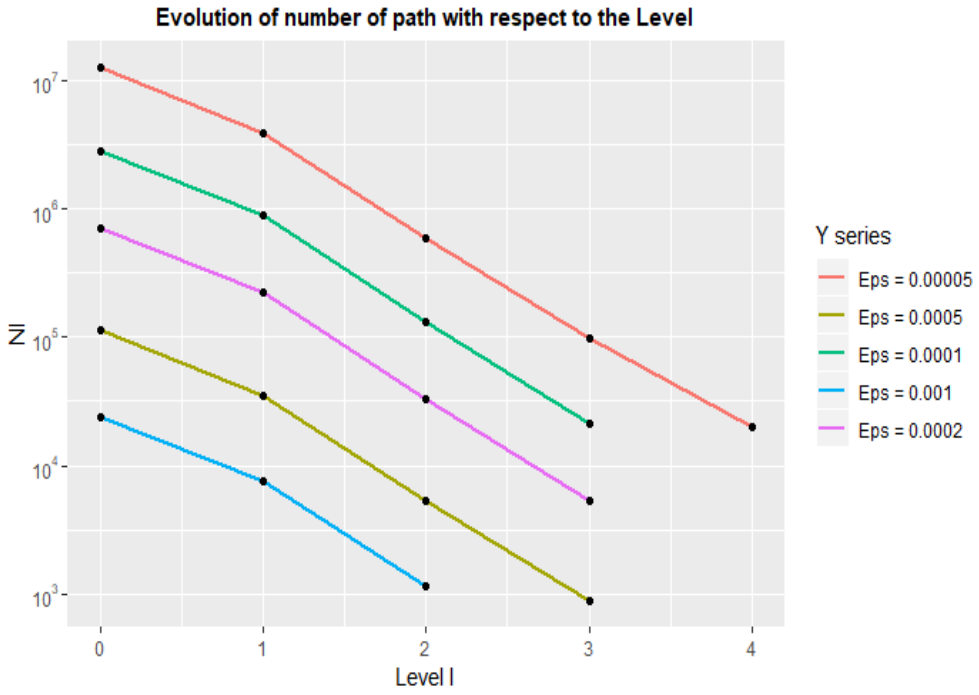


Once again, the plot was moved upward, and the slope of the Multilevel Monte Carlo has a decreasing slope of -2 approximately which is higher than the -1 obtained for the Geometric Brownian Motion. The standard Monte Carlo mean is independent from the level and we can see a strange behaviour at level 5 for the $\widehat{P_l}$ - $\widehat{P_{l-1}}$ curve.

Number of paths N_l

The main idea of the Multilevel Monte Carlo is the trade off between the geometric increase of the number of points generated for each simulation with respect to the level and the decrease of optimal number of paths, level after level in order to reduce the computational cost.

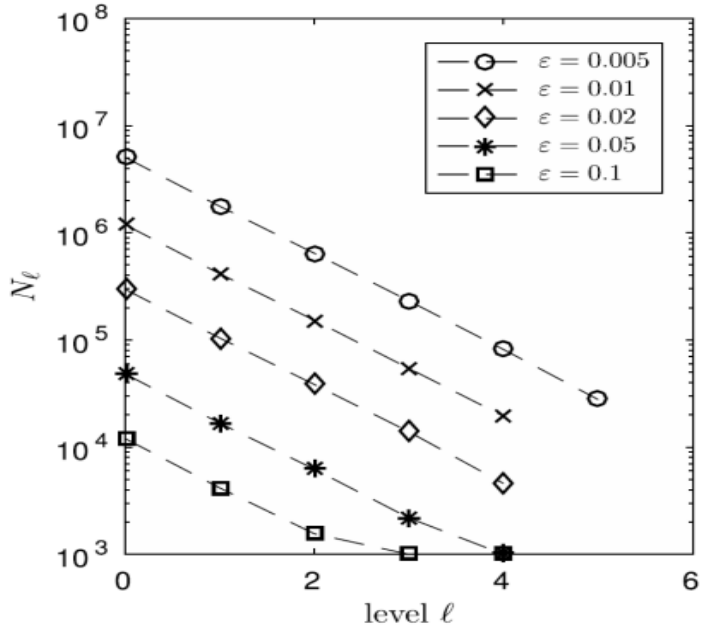
In this plot, each line corresponds to one Multilevel calculation and shows the value for N_l, the number of paths, with respect to the level $l = 0 \dots 4$. Each line is associated to a single value of the Root Mean Square Error.



It also shows that the value for L, the maximum level of refinement (here set to 4) increases as the value of ε, the accuracy required for the Root Mean Square Error, decreases.

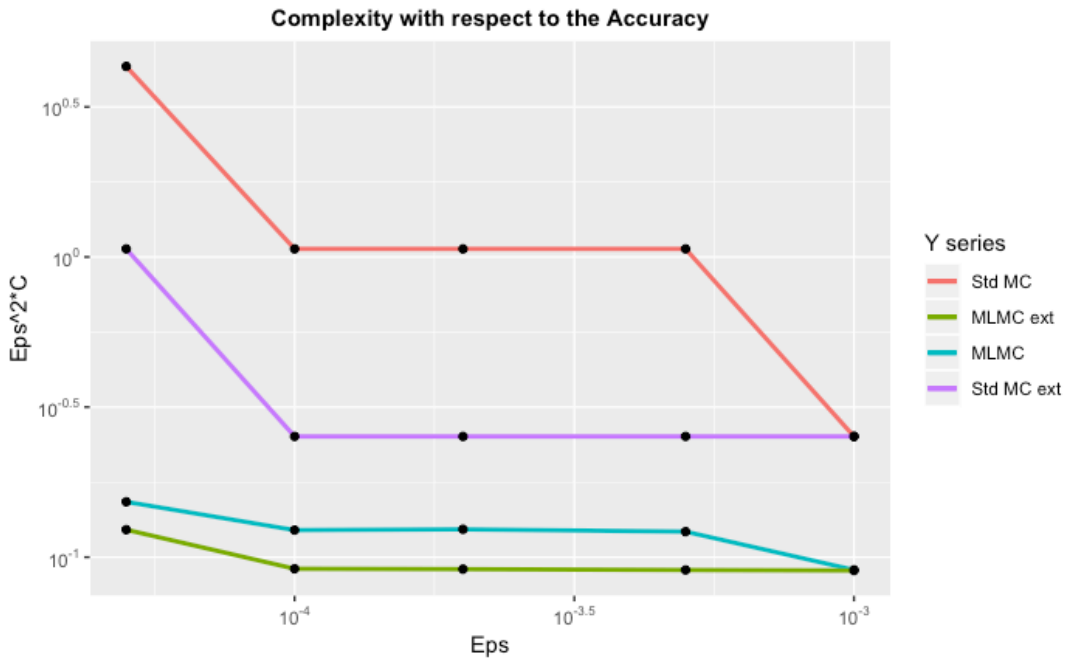
The lower the ε, the higher the number of simulations. This is totally logic that if a better precision is required, the number of simulations to achieve this accuracy should be higher.

In the case of a Variance Gamma process, the values are a little bit lower but we maintain the same general shape.



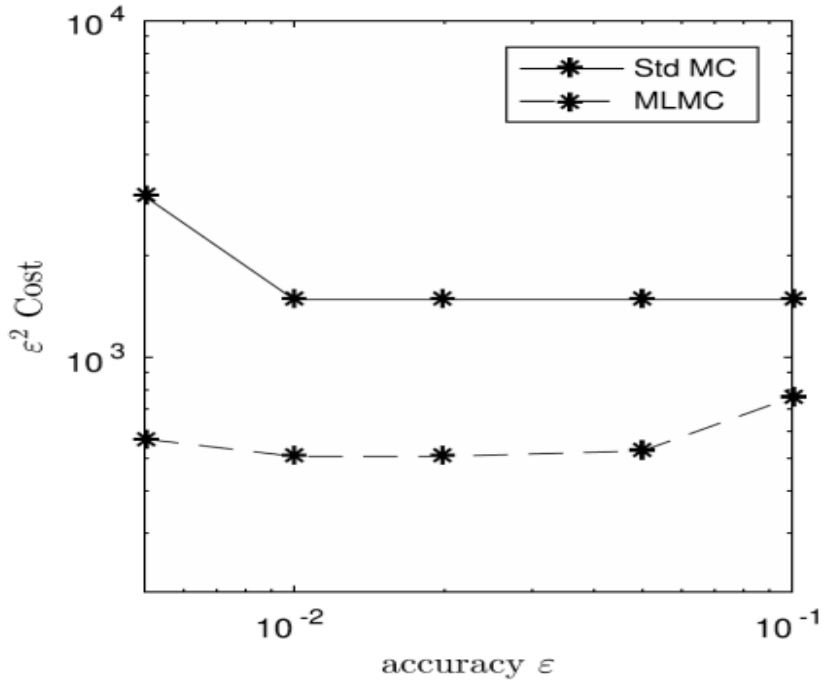
Computational Cost

This last graph tries to point out the improvement made by the Multilevel Monte Carlo in terms of computational complexity in comparison to the standard Monte Carlo.



The plot shows the variation of the computational complexity C (amount of resources required for running the algorithm) with the desired accuracy ε. The plot is of $\varepsilon^2 C$ versus ε because we expect to see that $\varepsilon^2 C$ is only very weakly dependent on ε for the Multilevel method. For the standard Monte Carlo method, theory predicts that $\varepsilon^2 C$ should be proportional to the number of timesteps on the finest level, which is roughly proportional to ε^{-1}. This is shown with the staircase effect in the figure.

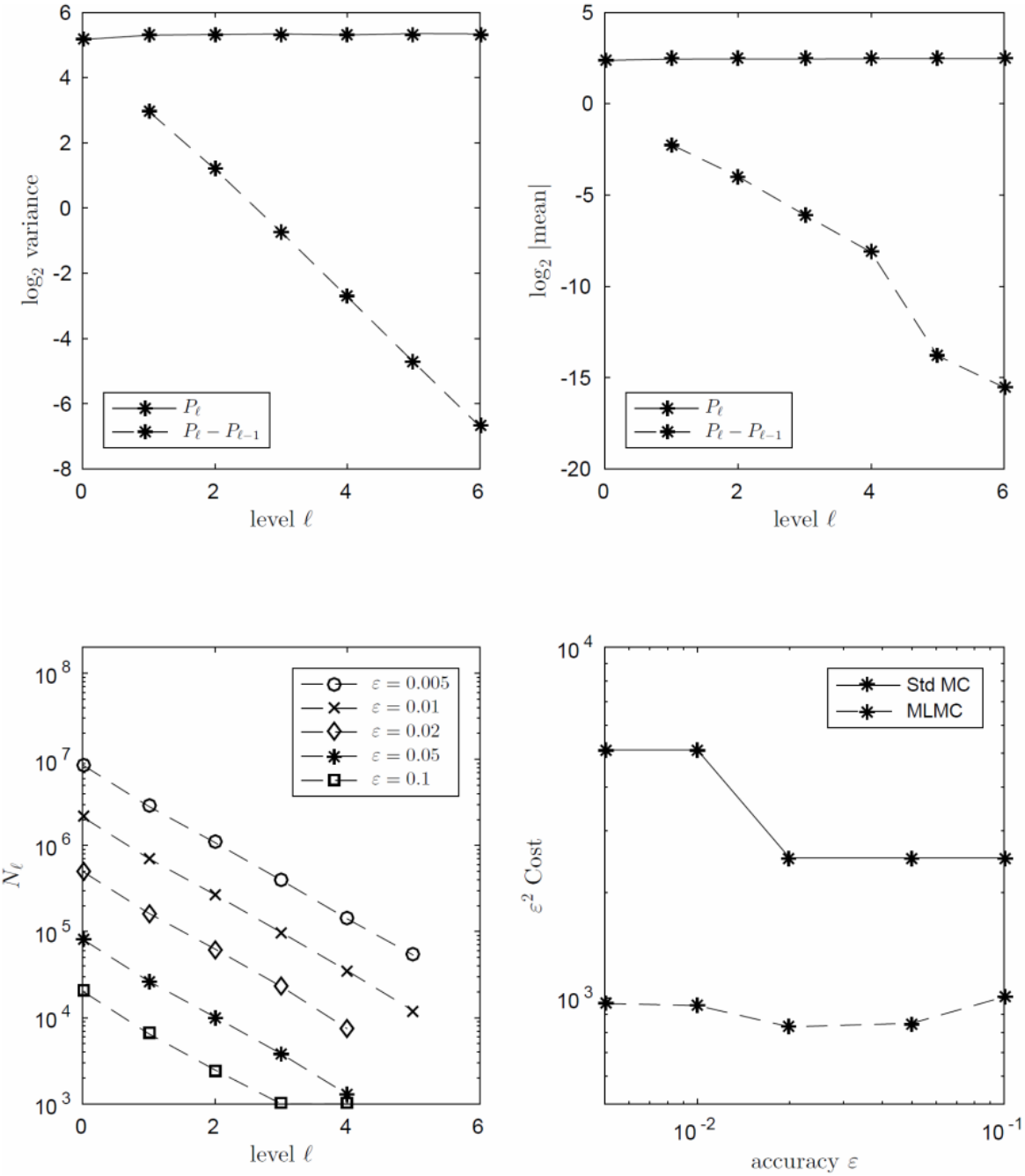
In the case of Variance Gamma process, the authors obtained the following plot:



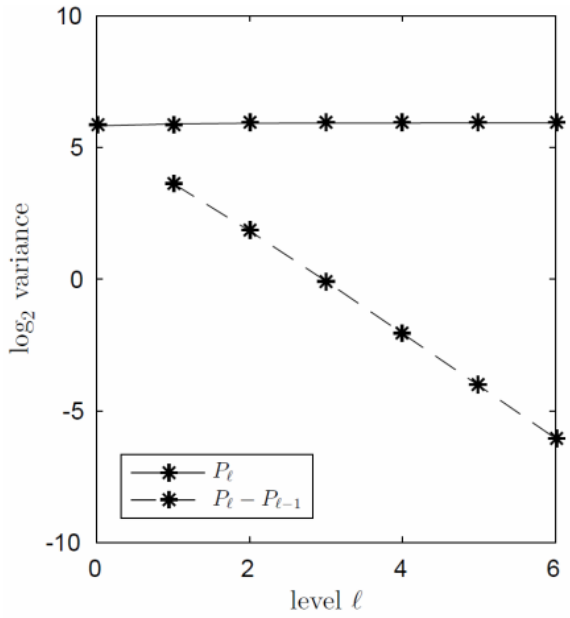
For the comparison, we need to compare it to the red and grey curves of the previous graph. We see a higher computational cost for both Multilevel Monte Carlo and Standard Monte Carlo in the case of Variance Gamma process which seems logic. The two graphs also point out that the complexity is less for the Multilevel Monte Carlo, which is the overall aim of this method. For the Variance Gamma, the convergence rate for the variance is superior to one, so the Multilevel Monte Carlo gives a complexity of $\Theta(\varepsilon^2)$ which is consistent with the little variation obtained in the Multilevel Monte Carlo graph.

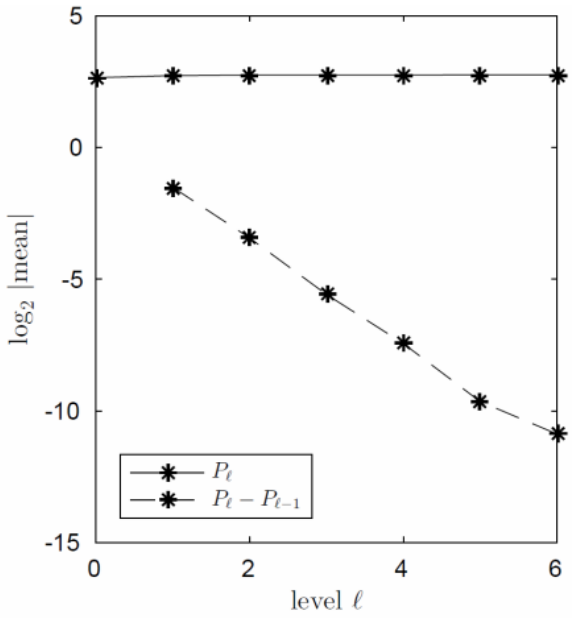
For an Asian Option, the Multilevel Monte Carlo is only 3-8 times more efficient that Standard Monte Carlo. In fact, the high convergence rate means that only 4 levels of refinements are often required, meaning only with M = 2, 16 differences in cost between the Multilevel and the Standard Monte Carlo.

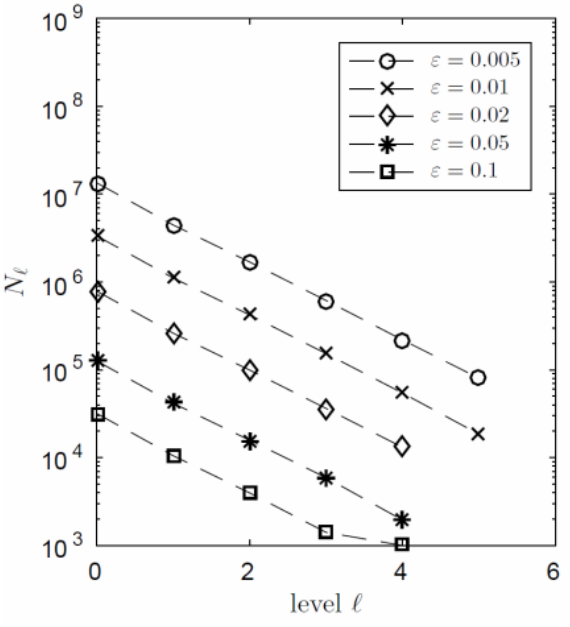
As an example, here are the plots obtained in the paper for a Normal Inverse Gaussian for the same Asian Option.

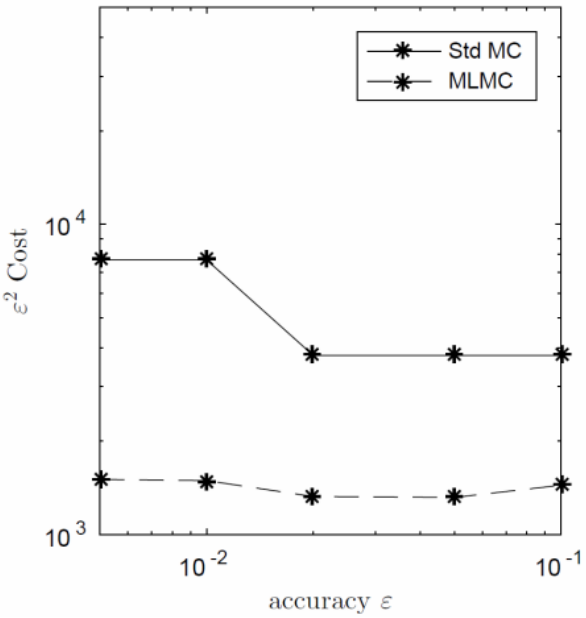


For the spectrally negative α-stable process with the Asian Option, the plots are the following:









IV. Critical analysis

As our replication focused entirely on the VG and NIG models let's discuss further these two processes.

Lévy process

Exponential Levy models generalize the classical Black and Scholes setup by allowing the stock prices to jump while preserving the independence and stationarity of returns. There are ample reasons for introducing jumps in financial modeling.

Firstly, asset prices do jump, and some risks simply cannot be handled within continuous-path models.

Secondly, the well documented phenomenon of implied volatility smile in option markets shows that the risk-neutral returns are non-gaussian and leptokurtic. While the smile itself can be explained within a model with continuous paths, the fact that it becomes much more pronounced for short maturities is a clear indication of the presence of jumps. In continuous-path models, the law of returns for shorter maturities becomes closer to the Gaussian law, but in the reality, in models with jumps returns become less Gaussian as the horizon becomes shorter.

Finally, jump processes correspond to genuinely incomplete markets, whereas all continuous-path models are either complete or 'completable' with a small number of additional assets. This fundamental incompleteness makes it possible to carry out a rigorous analysis of the hedging error and find ways to improve the hedging performance using additional instruments such as liquid European options.

A great advantage of exponential Lévy models is their mathematical tractability, which makes it possible to perform many computations explicitly and to present deep results of modern mathematical finance in a simple manner. However, some fundamental aspects such as asymptotic behavior of implied volatility or the computation of hedge ratios have only recently been given a rigorous treatment.

Lévy processes are stochastic processes with stationary and independent increments. The only Lévy process with continuous trajectories is the Brownian motion with drift; all other

representatives of this class admit discontinuities in finite or (countably) infinite number. A general Lévy process can be represented as :

$$X_t = \gamma t + B_t + N_t + \lim_{\varepsilon \to 0} M_t^{\varepsilon}$$

Examples of exponential Lévy models

Parametric exponential Lévy models fall into two categories.

In the first category, called jump-diffusion models, the "normal" evolution of prices is given by a diffusion process, punctuated by jumps at random intervals. Here the jumps represent rare events — crashes and large drawdowns. Such an evolution can be represented by a Levy process with a nonzero Gaussian component and a jump part with finitely many jumps:

$$X_t = \gamma t + \sigma w_t + \sum_{i=1}^{N_t} Y_t,$$

where (Y_i) are independent identically distributed, and N is a Poisson process.

- Merton model, first model of this type, suggested jumps in the log price X are assumed to have a Gaussian distribution: $Y_i \sim N(\mu, \delta^2)$.
- Kou model, jump sizes are distributed according to an asymmetric Laplace law with a density of the form: $v_0(dx) = [p\lambda_+ e^{-\lambda_+ x} I_{x>0} + (1-p)\lambda_- e^{-\lambda_- x} I_{x<0}]dx$, with $\lambda_+ > 0$, $\lambda_- > 0$ governing the decay of the tails for the distribution of positive and negative jump sizes and p ∈ [0, 1] representing the probability of an upward jump. The probability distribution of returns in this model has semi-heavy (exponential) tails.

The second category consists of models with an infinite number of jumps in every interval, called infinite activity or infinite intensity models. In these models, one does not need to introduce a Brownian component since the dynamics of jumps is already rich enough to generate nontrivial small-time behavior.

- Variance Gamma Process, which was introduced to the finance community as a model for asset returns (log-price increments) and options pricing. Variance gamma process is pure jump, and thus there is no continuous (Brownian motion and deterministic) component.

- Normal inverse Gaussian Process, which is a continuous probability distribution that is defined as the normal variance-mean mixture where the mixing density is the inverse Gaussian distribution.

  We will discuss VG and NIG process in the following contents.
- Hyperbolic motion

  The NIG process is a special case of a set of generalized hyperbolic distributions.
- CGMY: tempered stable process, Lévy density:

$$v(x) = \frac{c_-}{|x|^{1+\alpha_-}} e^{-\lambda_- x} I_{x<0} + \frac{c_+}{x^{1+\alpha_+}} e^{-\lambda_+ x} I_{x>0}$$

  with $\alpha_+ < 2 \; and \; \alpha_- < 2$. The Variance Gamma is a special case of CGMY.

The models we choose in the paper are Variance Gamma Model and Normal Inverse Gaussian Model, now let's discuss these two processes with generating Monte Carlo algorithms.

Advantages of Variance Gamma and Normal Inverse Gaussian

Variance Gamma model

The Variance Gamma process is a pure jump, three-parameter stochastic process, which allows us to control not only the volatility but also the skewness and kurtosis of the return distribution. Specifically, the option price computed by the VG model reproduced the implied volatility smile. In addition, it gives higher put option value than the Black Scholes formula, which is known to underprice put options.

The VG process had been implemented for option pricing with Fast Fourier transform by Carr and Madan, but this approach can be used only on European style option. Additionally, an exercise based on Hirsa and Madan approach which employs a finite difference method by first solving the PDE of the process, can price the American option while forming an exercise boundary. To do so, the integral part of the equation is discretized according to the jump size.

However, the VG model have a disadvantage: the functions involved, expressed as power series, are computationally expensive. To price options using the closed form may be slower than computing prices by numerical procedures employing Fourier-inversion.

NIG model

Due to the specific characteristics, the NIG distribution is very interesting for applications within finance. With its four parameters $\alpha, \beta, \mu \; and \; \delta$ (α: tail heaviness of steepness, β: symmetry, δ : scale, μ : location), the NIG distribution allows both skewness and higher kurtosis than the Normal (Gaussian) distribution. These properties, especially allowing slower decreasing tails, makes it a suitable tool for modeling financial derivatives with most commonly the stock return as the underlying asset. That is to say, the NIG market model fits market data better, both with historical and implied parameters, compared to the Black & Scholes Model.

The employment of the NIG distribution does not only bring significant improvement with respect to computation times but also more flexibility in the modelling of the dependence structure of a credit portfolio.

V. Extensions

The Multilevel Monte Carlo method does certainly improve the computational cost compared to the standard Monte Carlo method. However, this method's final approximations appear to be not so significantly from the standard Monte Carlo approximations.

As part of the extension of the initial paper we decide to use the multilevel monte carlo method in order to price options and analyse the benefits of this model. Our primary question is to figure out, in practice if we have the value added of this method described above in a pricing point of view. How fast our model converge to the real price ?

In order to analyse that we will price an asian option following a variance gamma process. First note that in order to implement the algorithm we have to add in mean correcting pricing measure to allow us to price options under the risk neutral world.

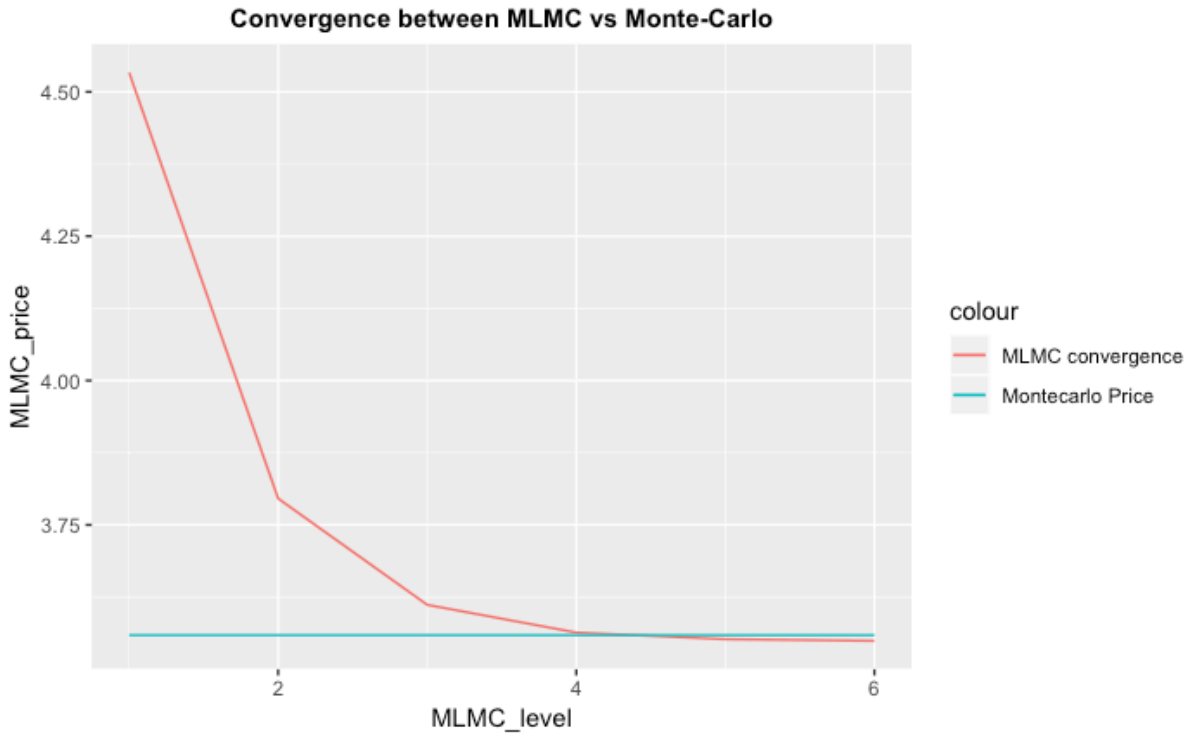


Figure 3 : Convergence between MLMC price vs standard monte carlo price. Real price is around 3.487

The graph shows the convergence of the price of an asian option with multilevel monte-carlo. The blue line corresponds to the price to match which was obtained by a standard Monte Carlo algorithm (under a variance gamma process). The real price of the option is

3.38698. As we said earlier our model converge quite swiftly the real price but the gain of accuracy is not that impressive.

However, as we will see in the following graphs to come the number of simulation and the variance of the payoff will decrease sharply at every level of computation sharply allowing us to reduce tremendously our computational cost.

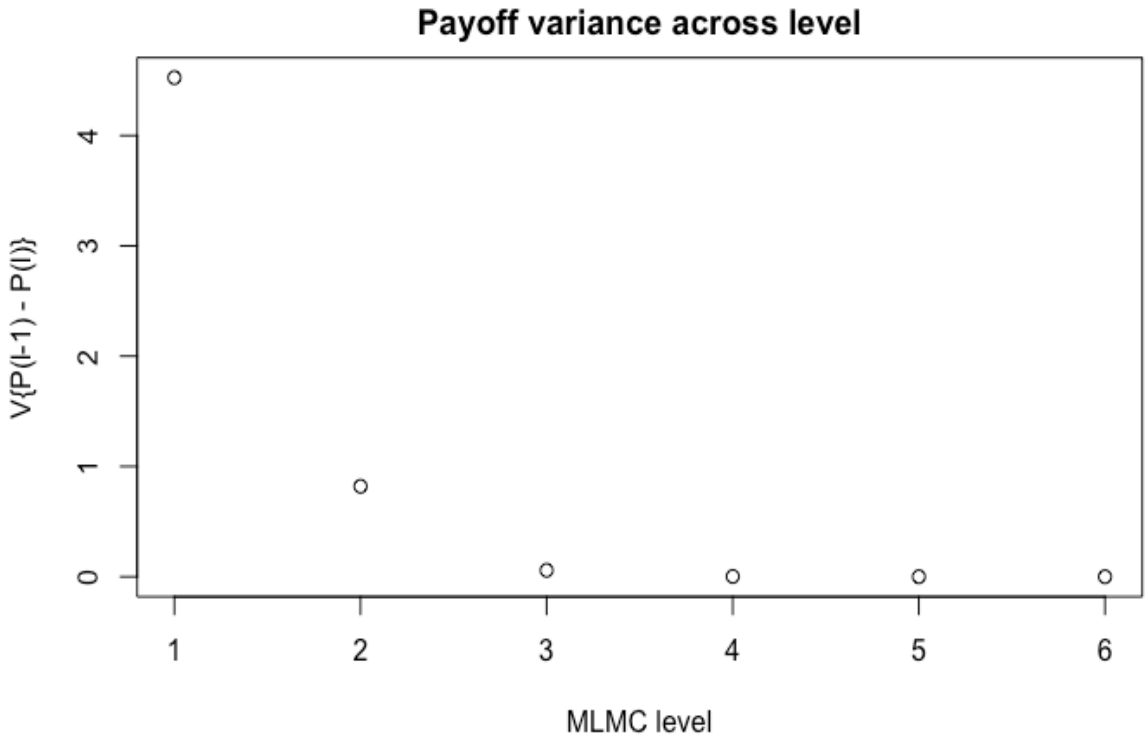


Figure 4 : Payoff variance. V[1]:"4.525" ~ V[2]:"0.820" ~ V[3] :"0.05" 9~ V[4] :"0.002" ~ V[5]:"0.001" ~ V[6] :"0.000007"

First let take a look to the variance of the payoff as we increase in the level the variance decrease intensively. The variance of a classical monte carlo is a flat curve with the multilevel monte carlo, the variance is converging swiftly to 0.

This convergence in the variance will also have an impact in the number of simulation needed to be done from one level to another as long as our constrains are not matched. Recall that the variance V_l is the variance between actual payoff and previously computed payoff such that $V_l = V[\widehat{P_l} - \widehat{P_{l-1}}]$.

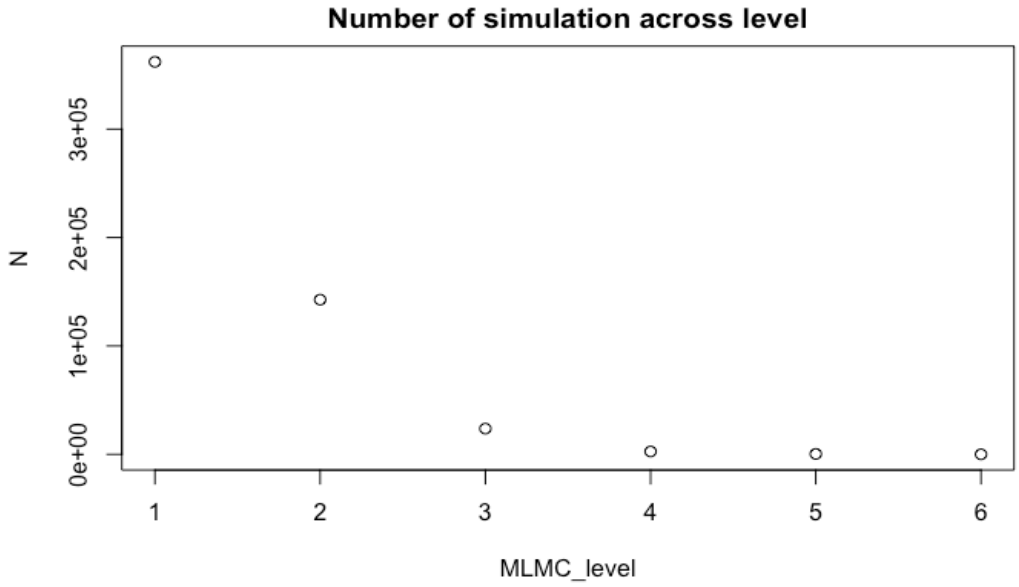


Figure 5 : Optimal number of simulation. $N[1]$:362033 ~ $N[2]$:142669 ~ $N[3]$:23728 ~ $N[4]$:2657 ~ $N[5]$:397 ~ $N[6]$:38

N is the number of simulation, with respect to the level l={0,…,6}. For each multilevel computation, the algorithm decides how much simulation is needed.

Variance of $\boldsymbol{l}\ (\boldsymbol{V_l})$ is evaluated on a fixed number of samples Depending on $\boldsymbol{N_l}$, it will be required to sample some more trajectories. Recall that the following formula gives us the optimal number of simulation depending on the variance V_l the root mean squared error we aim to target and the number of timesteps.

$$N_l = [2\varepsilon^{-2}\sqrt{V_l h_l}\left(\sum_{l=0}^{L}\sqrt{\frac{V_l}{h}_l}\right)$$

These 3 graphs allow us to see the gain on cost of computation, the last one is interesting and shows a relevant aspect of the multilevel method. Most of the simulations are done in the first levels with low accuracy and in the finest level we have much less simulation to perform with a high accuracy in order to have a final price for this asian option.

As a conclusion, the method is useful when one market operator evolving in a fast paced environment need to have an accurate price for a given option faster. Instead of running

2.000.000 monte carlo simulations, by using the multilevel method he could considerably reduce the time of computation and achieve a good approximation of the price of the option.

VI. Conclusion

In the above work, we have shown that a multilevel approach, using exponential levy processes, can reduce the order of complexity of a standard Monte Carlo simulations. One of the key point of our extension that could be relevant to dig in is the calibration part. A detailed analysis of the calibration of the model to market data could influence the result of the pricing part but the overall analysis of the convergence and complexity reduction will remain the same as they are fundamental to this model.

VII. References

- Kebaier, A. 2005. Statistical Romberg extrapolation: A new variance reduction method and applications to options pricing. Ann. Appl. Probab. 14(4) 2681–2705.
- Michael B. Giles.2008. Multilevel Monte Carlo Path Simulation. Operations Research Vol. 56, No. 3, May–June 2008, pp. 607–617 issn0030-364X|eissn1526-5463|08|5603|0607.
- Michael B. Giles & Yuan Xia. 2017.Multilevel Monte Carlo for exponential Lévy models. Finance Stoch(2017) 21:995–1026 DOI 10.1007/s00780-017-0341-7.
- Kuznetsov, A.: On extrema of stable processes. Ann. Probab. 39, 1027–1060 (2011).
- Dereich, S. : Multilevel Monte Carlo algorithms for Lévy-driven SDEs with Gaussian correction. Ann. Appl. Probab. 21, 283–311 (2011)
- Dereich, S., Heidenreich, F.: A multilevel Monte Carlo algorithm for Lévy-driven stochastic differential equations. Stoch. Process. Appl. 121, 1565–1587 (2011).
- Schoutens, W.: Lévy Processes in Finance: Pricing Financial Derivatives. Wiley/Blackwell, New Jersey (2003).
- Kyprianou, A.: Introductory Lectures on Fluctuations of Lévy Processes with Applications. Springer, Berlin (2006).
- Chen, A.: Sampling Error of the Supremum of a Lévy Process. Ph.D. Thesis, University of Illinois at Urbana-Champaign (2011).

VIII. Appendices

To visualise the evolution of the number of points

```{r}
# A simple visualisation of the number of points evolution
library(ggplot2)
x = 50 # Initial price
t = 1 # Time
M = 4 # Number of points
L = 3 # Level
b = 0.1 # Drift
s = 0.2 # Volatility
Date = (1:(M^3))/M^3*t # Vector of dates

#Initialize trajectories for each level
x0 = rep(NA, M^3)
x0[1] = x
x1 = rep(NA, M^3)
x1[1] = x
x2 = rep(NA, M^3)
x2[1] = x
x3 = rep(NA, M^3)
x3[1] = x
dataGraph0 = data.frame(Date, x0)
dataGraph1 = data.frame(Date, x1)
dataGraph2 = data.frame(Date, x2)
dataGraph3 = data.frame(Date, x3)
for(i in 1:M^L){
  x = x * (1 + b*t/M^L + s*sqrt(t/M^L)*rnorm(1))
```

```
  print(i*t/M^L)
  print(which(dataGraph3$Date == i*t/M^L))
  dataGraph3[which(dataGraph3$Date == i*t/M^L), 2] = x
  if(i%%M == 0){
    dataGraph2[which(dataGraph2$Date == i*t/M^L), 2] = x
  }
  if(i%%M^2 == 0){
    dataGraph1[which(dataGraph1$Date == i*t/M^L), 2] = x
  }
  if(i%%M^3 == 0){
    dataGraph0[which(dataGraph0$Date == i*t/M^L), 2] = x
  }
}

dataGraph0 = na.omit(dataGraph0)
dataGraph1 = na.omit(dataGraph1)
dataGraph2 = na.omit(dataGraph2)
dataGraph3 = na.omit(dataGraph3)
dataGraph3[1,2] = 50

ggplot(data = dataGraph0, aes(x = Date)) + geom_line(aes(y = x0)) + ggtitle("Level 0 : M^0 + 1
= 2 points") + theme(plot.title = element_text(color = "black", size = 11, face = "bold", hjust =
0.5)) + labs(x = "Time", y = "Stock Value") + ylim(35, 65)
ggplot(data = dataGraph1, aes(x = Date)) + geom_line(aes(y = x1)) + ggtitle("Level 1 : M^1 + 1
= 5 points") + theme(plot.title = element_text(color = "black", size = 11, face = "bold", hjust =
0.5)) + labs(x = "Time", y = "Stock Value") + ylim(35, 65)
ggplot(data = dataGraph2, aes(x = Date)) + geom_line(aes(y = x2)) + ggtitle("Level 2 : M^2 + 1
= 17 points") + theme(plot.title = element_text(color = "black", size = 11, face = "bold", hjust
= 0.5)) + labs(x = "Time", y = "Stock Value") + ylim(35, 65)
ggplot(data = dataGraph3, aes(x = Date)) + geom_line(aes(y = x3)) + ggtitle("Level 3 : M^3 + 1
= 65 points") + theme(plot.title = element_text(color = "black", size = 11, face = "bold", hjust
= 0.5)) + labs(x = "Time", y = "Stock Value") + ylim(35, 65)
```

```

**The functions used for our plots**

```{r}
#Level l estimator
mlmc_l = function(M, l, N){
  Time = 1
  r = 0.05
  sig = 0.2
  nf = M^l
  nc = nf/M
  hf = Time/nf
  hc = Time/nc
  Result = rep(0, 4)

  for(N1 in seq(1, N, by = 10000)){
    N2 = min(10000,N-N1+1)
    X0 = 1
    Xf = X0*rep(1, N2)
    Xc = Xf
    Af = 0.5*hf*Xf
    Ac = 0.5*hc*Xc
    if(l == 0){
      dWf = sqrt(hf)*rnorm(N2, mean = 0, sd = 1)
      Xf = Xf + r*Xf*hf + sig*Xf*dWf
      Af = Af + 0.5*hf*Xf
    } else {
      for(n in 1:nc){
        dWc = rep(0,N2)
        for(m in 1:M){
          dWf = sqrt(hf)*rnorm(N2, mean = 0, sd = 1)
```
```

```
        dWc = dWc + dWf
        Xf = Xf + r*Xf*hf + sig*Xf*dWf
        Af = Af + hf*Xf
       }
      Xc = Xc + r*Xc*hc + sig*Xc*dWc
      Ac = Ac + hc*Xc
     }
     Af = Af - 0.5*hf*Xf
     Ac = Ac - 0.5*hc*Xc
   }
   Pf = pmax(0, Af - 1)
   Pc = pmax(0, Ac - 1)
   Pf = exp(-r*Time)*Pf
   Pc = exp(-r*Time)*Pc

   if(l == 0){
    Pc = 0
   }

   Result[1] = Result[1] + sum(Pf-Pc)
   Result[2] = Result[2] + sum((Pf-Pc)^2)
   Result[3] = Result[3] + sum(Pf)
   Result[4] = Result[4] + sum(Pf^2)
  }
  return(Result)
}

#MultiLevel MonteCarlo path estimation
mlmc = function(M, eps, extrap){

  L = -1
  N = 10000
```

```
converged = 0
suml = matrix(nrow = 3, ncol = 0)

while(converged == 0){

 #Initial Variance Estimate
 L = L+1
 sums = mlmc_l(M, L, N)
 suml = cbind(suml, rep(0, nrow(suml)))
 suml[1, L+1] = N
 suml[2, L+1] = sums[1]
 suml[3, L+1] = sums[2]

 #Optimal sample size
 Vl = suml[3,]/suml[1,] - (suml[2,]/suml[1,])^2
 Nl = ceiling(2*sqrt(Vl/M^(0:L)) * sum(sqrt(Vl*M^(0:L)))/eps^2)

 #Update sample sums
 for(l in 0:L){
  dNl = Nl[l+1] - suml[1, l+1]
  if(dNl > 0){
   sums = mlmc_l(M, l , dNl)
   suml[1, l+1] = suml[1, l+1] + dNl
   suml[2, l+1] = suml[2, l+1] + sums[1]
   suml[3, l+1] = suml[3, l+1] + sums[2]
  }
 }

 #Convergence Tests
 if(extrap != 0){
  range = 0
  if(L>1 & M^L>=16){
```

```
      con = M^range*(suml[2, L+1+range]/suml[1, L+1+range] - (1/M)*suml[2, L+range]/suml[1, L+range])
      converged = ifelse(max(abs(con)) < (M^2-1)*eps/sqrt(2) | M^L > 1024, 1, 0)
     }
   } else {
     range = c(-1, 0)
     if(L>1 & M^L>=16){
      con = M^range*suml[2, L+1+range]/suml[1, L+1+range]
      converged  = ifelse((max(abs(con)) < (M-1)*eps/sqrt(2)) | (M^L > 1024), 1, 0)
     }
   }
 }
 P = sum(suml[2, 1:L+1]/suml[1, 1:L+1])
 if(extrap != 0){
  P = P + suml[2, L+1]/suml[1, L+1]/(M-1)
 }

 Result = list("P" = P, "Nl" = Nl)
 return(Result)
}
```

The plots

```{r}
#Geometric Brownian Motion with Asian Option
library(ggplot2)
library(scales)
N = 2000000
M = 4
L = seq(0,4)
```

```
del1 = c()
del2 = c()
var1 = c()
var2 = c()

#Convergence Tests
for(l in L){
 cat('Step', l, "\n")
 sums = mlmc_l(M, l, N)
 del1 = c(del1, sums[3]/N)
 del2 = c(del2, sums[1]/N)
 var1 = c(var1, sums[4]/N - (sums[3]/N)^2)
 var2 = c(var2, sums[2]/N - (sums[1]/N)^2)
}

var2[1] = NA
del2[1] = NA
del3 = c(NA, NA, del2[3:length(del2)] - del2[2:(length(del2)-1)]/M)
dataGraph12 = data.frame(L, del1, var1, var2, del2, del3)

#Graph log variance with respect to Level
ggplot(data = dataGraph12, aes(x = L)) + geom_line(aes(y = log(var2)/log(M), colour = "black"), size = 1) + geom_point(aes(y = log(var2)/log(M))) + geom_line(aes(y = log(var1)/log(M), colour = "red"), size = 1) + geom_point(aes(y = log(var1)/log(M))) + ggtitle("The Log Variance with respect to the Level") + theme(plot.title = element_text(color = "black", size = 11, face = "bold", hjust = 0.5)) + labs(x = "Level l", y = "Log Variance") + scale_color_discrete(name = "Y series", labels = c("P(l)-P(l-1)", "P(l)")) + xlim(0,4) + ylim(-10,0)

#Graph log mean with respect to level
ggplot(data = dataGraph12, aes(x = L)) + geom_line(aes(y = log(abs(del1))/log(M), color = "black"), size = 1) + geom_point(aes(y = log(abs(del1))/log(M))) + geom_line(aes(y = log(abs(del2))/log(M), color = "red"), size = 1) + geom_point(aes(y = log(abs(del2))/log(M))) +
```

```
 geom_line(aes(y = log(abs(del3))/log(M), color = "green"), size = 1) + geom_point(y = log(abs(del3))/log(M)) + ggtitle("The Log Mean with respect to the Level") + theme(plot.title = element_text(color = "black", size = 11, face = "bold", hjust = 0.5)) + labs(x = "Level l", y = "Log Mean") +
 scale_color_discrete(name = "Y series", labels = c("P(l)", "Y(l)-Y(l-1)", "P(l)-P(l-1)")) + xlim(0,4) + ylim(-12,0)

#Complexity tests
Eps = c(0.001, 0.0005, 0.0002, 0.0001, 0.00005)
maxl = 0
mlmc_cost = matrix(nrow = length(Eps), ncol = 2)
std_cost = matrix(nrow = length(Eps), ncol = 2)
Nls = c()
for(extrap in 0:1){
 for(i in 1:length(Eps)){
  eps = Eps[i]
  res = mlmc(M, eps, extrap)
  l = length(res$Nl) - 1
  maxl = max(maxl, l)
  mlmc_cost[i, extrap+1] = (1+1/M)*sum(res$Nl*M^(0:l))
  std_cost[i, extrap+1] = sum((2*var1[(0:l)+1]/eps^2)*M^(0:l))
  Nls = c(Nls, list(res$Nl))
 }
}

List3 = list(L, Nls[[5]], Nls[[4]], Nls[[3]], Nls[[2]], Nls[[1]])
List3 = lapply(List3, `length<-`, max(lengths(List3)))
dataGraph3 = data.frame(List3)
colnames(dataGraph3) = c('L', 'd5', 'd4', 'd3', 'd2', 'd1')

#Graph of Number of Paths with respect to Level
```

```
ggplot(dataGraph3, aes(x = L)) + geom_line(aes(y = d5, color = "black"), size = 1) + geom_point(aes(y=d5)) + geom_line(aes(y = d4, color = "green"), size = 1) + geom_point(aes(y = d4)) + geom_line(aes(y = d3, color = "red"), size = 1) + geom_point(aes(y = d3)) + geom_line(aes(y = d2, color = "blue"), size = 1) + geom_point(aes(y = d2)) + geom_line(aes(y = d1, color = "pink"), size = 1) + geom_point(aes(y=d1)) + scale_y_log10(breaks = trans_breaks("log10", function(x) 10^x),labels = trans_format("log10", math_format(10^.x))) + ggtitle("Evolution of number of path with respect to the Level") + theme(plot.title = element_text(color = "black", size = 11, face = "bold", hjust = 0.5)) + labs(x = "Level l", y = "Nl") + xlim(0,4) + scale_color_discrete(name="Y series", labels = c("Eps = 0.00005", "Eps = 0.0005", "Eps = 0.0001", "Eps = 0.001", "Eps = 0.0002"))

Std1 = Eps^2*std_cost[,1]
Std2 = Eps^2*std_cost[,2]
MC1 = Eps^2*mlmc_cost[,1]
MC2 = Eps^2*mlmc_cost[,2]
dataGraph4 = data.frame(Eps, Std1, Std2, MC1, MC2)

#Graph of complexity with respect to accuracy
ggplot(dataGraph4, aes(x = Eps)) + geom_line(aes(y = Std1, color = "black"), size = 1) + geom_point(aes(y = Std1)) + geom_line(aes(y = Std2, color = "red"), size = 1) + geom_point(aes(y = Std2)) + geom_line(aes(y = MC1, color = "green"), size = 1) + geom_point(aes(y = MC1)) + geom_line(aes(y = MC2, color = "blue"), size = 1) + geom_point(aes(y = MC2)) +
 scale_x_log10(breaks = trans_breaks("log10", function(x) 10^x),labels = trans_format("log10", math_format(10^.x))) +
 scale_y_log10(breaks = trans_breaks("log10", function(x) 10^x),labels = trans_format("log10", math_format(10^.x))) +
 ggtitle("Complexity with respect to the Accuracy") + theme(plot.title = element_text(color = "black", size = 11, face = "bold", hjust = 0.5)) +
 scale_color_discrete(name="Y series", labels = c("Std MC", "MLMC ext", "MLMC", "Std MC ext"))
```

```

**Multilevel for Asian Option with Variance Gamma Process**

```{r}
#Multilevel for Asian Option with Variance Gamma Process
eps = 0.005          # Sought precision for RMSE
Y = 1000             # Initialisation of Y
Y_new = 0            # Iteration Y update
M = 2                # M parameter
t = 1                # Time to maturity
h = t/M              # First time-step
N_base = M^14        # Base number of trajectories = 16384
S = 100
K = 100
theta = -0.1436
sigma = 0.1213
k = 0.1686
r = 0.05
m = r + (1/k)*log(1 + theta*k - 0.5*sigma^2*k)
gamma = rgamma(N_base, shape = t/k, scale = k)
normal = rnorm(N_base)
sim = S*exp(m*t + theta*gamma + sigma*sqrt(gamma)*normal)
pay = pmax((sim+S)/2 - K, 0)
C = mean(pay)
V = 0
l = 1
N = N_base
price <- c()
while(max(abs(Y)/M,abs(Y_new)) > eps*(M-1)/sqrt(2) & l < 12){
  print(paste("Step n?", l, sep = ""))                    # Print stage number
```
```

```
 gamma = matrix(rgamma(M^l*N_base, shape = t/M^l/k, scale = k), nrow = M^l, ncol =
N_base)
 normal = matrix(rnorm(M^l*N_base), nrow = M^l, ncol = N_base)
 sim = m*t/M^l + theta*gamma + sigma*sqrt(gamma)*normal
 sim = S * exp(apply(sim, 2, cumsum))
 p1 <- pmax(apply(sim,2,mean)-K,0)
 if(l>1){
  p0 = pmax(apply(sim[M*(1:(M^(l-1))),],2,mean)-K,0)
 } else {
  p0 = pmax((sim[M,]+S)/2-K,0)
 }
 h[l] <- t/(M^l)
 V[l] <- var(p1-p0)
 N[l] <- ceiling(2/eps^2*sqrt(V[l]*h[l])*sum(sqrt(V/h)))
 if(N[l] >= N_base){
  gamma2 = matrix(rgamma(M^l*(N[l]-N_base), shape = t/M^l/k, scale = k), nrow = M^l, ncol
= N_base)
  normal2 = matrix(rnorm(M^l*(N[l]-N_base)), nrow = M^l, ncol = N_base)
  sim2 = m*t/M^l + theta*gamma2 + sigma*sqrt(gamma2)*normal2
  sim2 = S*exp(apply(sim2, 2, cumsum))
  sim = cbind(sim, sim2)
 }
 p1 = pmax(apply(sim,2,mean)-K,0)
 if(l>1){
  p0 = pmax(apply(sim[M*(1:(M^(l-1))),],2,mean)-K,0)
 } else {
  p0 <- pmax((sim[M,]+S)/2-K,0)
 }
 Y <- Y_new              # Update
 Y_new <- mean(p1)-mean(p0)  # Update
 C <- C + Y_new          # Update price
 price[l] <- C
```

```
 l <- l + 1               # Increment stage
}
asian_price <- C * exp(-r*t)
MLMC_price <- price * exp(-r*t)
MLMC_level <- seq(1,l-1,1)
Montecarlo_price <-4.48 ; Montecarlo_price <- matrix(Montecarlo_price, 1, length(MLMC_level))
convergence <- data.frame(Montecarlo_price, MLMC_level, MLMC_price)
plot(MLMC_level, MLMC_price)
plot(MLMC_level, V)
plot(MLMC_level, N)
```

```{r}
#Classical Monte Carlo for Variance Gamma
PayoffVG = function(numberSimu, S0, Time, deltaT, typeProduct, K, r, Barrier){
 numberPoints = Time/deltaT
 sigma = 0.1213
 theta = -0.1436
 kv = 0.1686
 m = r + (1/kv)*log(1+theta*kv-0.5*sigma^2*kv)
 Payoff = 0
 for(i in 1:numberSimu){
  Indicator = 1
  St = S0
  X0 = 0
  Xt = X0
  sumAsian = St
  for(j in 1:numberPoints){
   G = rgamma(1, shape = deltaT/kv, scale = kv)
   N = rnorm(1, mean = 0, sd = 1)
   Xt = Xt + m*deltaT + theta*G + sigma*sqrt(G)*N
   St = S0*exp(Xt)
```

```
    sumAsian = sumAsian + St
    if(St>Barrier & typeProduct == 'BarrierOption'){
      Indicator = 0
      break
    }
  }
  if(typeProduct == 'AsianOption'){
    Payoff = Payoff + max(0, sumAsian/numberPoints - K)
  }
  if(typeProduct == 'BarrierOption'){
    Payoff = Payoff + Indicator*max(0, St - K)
  }
 }
 Payoff = Payoff/numberSimu
 return(exp(-r*Time)*Payoff)
}

numberSimu = 25000
S0 = 100
Time = 1
deltaT = 0.005
typeProduct = 'AsianOption'
K = 100
r = 0.05
Barrier = 65
Result = PayoffVG(numberSimu, S0, Time, deltaT, typeProduct, K, r, Barrier)
print(Result)
```

```{r}
#Classical Monte Carlo for Normal Inverse Gaussian Process
library(SafeBayes)
```

```
PayoffNIG = function(numberSimu, S0, Time, deltaT, typeProduct, K, r, Barrier){
 numberPoints = Time/deltaT
 sigma = 0.13
 theta = -0.1313
 kv = 15.5
 A = theta/sigma^2
 B = sqrt(theta^2+(sigma^2)/kv)/sigma^2
 C = sqrt(theta^2+2*(sigma^2)/kv)/(2*pi*sigma*sqrt(kv))
 m = r - (1/kv) + pi*C*B*(1/kv)*sqrt(B^2-(A+1)^2)
 Payoff = 0
 for(i in 1:numberSimu){
  Indicator = 1
  St = S0
  X0 = 0
  Xt = X0
  sumAsian = St
  for(j in 1:numberPoints){
   I = rinvGauss(1, 1/kv, 1)
   N = rnorm(1, mean = 0, sd = 1)
   Xt = Xt + m*deltaT + theta*I + sigma*sqrt(I)*N
   St = S0*exp(Xt)
   sumAsian = sumAsian + St
   if(St>Barrier & typeProduct == 'BarrierOption'){
    Indicator = 0
    break
   }
  }
  if(typeProduct == 'AsianOption'){
   Payoff = Payoff + max(0, sumAsian/numberPoints - K)
  }
  if(typeProduct == 'BarrierOption'){
   Payoff = Payoff + Indicator*max(0, St - K)
```

```
   }
  }
  Payoff = Payoff/numberSimu
  return(exp(-r*Time)*Payoff)
}

#MonteCarlo for NIG with Barrier Option
numberSimu = 25000
S0 = 100
Time = 1
deltaT = 0.005
typeProduct = 'BarrierOption'
K = 100
r = 0.05
Barrier = 125
Result = PayoffNIG(numberSimu, S0, Time, deltaT, typeProduct, K, r, Barrier)
print(Result)
```

Multilevel for Barrier Option with Normal Inverse Gaussian

```{r}
#Multilevel for Barrier Option with Normal Inverse Gaussian
library(SafeBayes)
eps = 0.005          # Sought precision for RMSE
Y = 1000             # Initialisation of Y
Y_new = 0            # Iteration Y update
M = 2                # M parameter
t = 1                # Time to maturity
h = t/M              # First time-step
N_base = M^14        # Base number of trajectories = 16384
S = 100
K = 100
```

```
theta = -0.1313
sigma = 0.1836
k = 15.5
r = 0.05
Barrier = 115
A = theta/sigma^2
B = sqrt(theta^2+(sigma^2)/k)/sigma^2
C = sqrt(theta^2+2*(sigma^2)/k)/(2*pi*sigma*sqrt(k))
m = r - (1/k) + pi*C*B*(1/k)*sqrt(B^2-(A+1)^2)
inverse = rinvGauss(N_base, 1/k, 1)
normal = rnorm(N_base)
sim = S*exp(m*t + theta*inverse + sigma*sqrt(inverse)*normal)
condition = ifelse(sim < B, 1, 0)
pay = pmax((sim - K)*condition, 0)
C = mean(pay)
V = 0
l = 1
N = N_base

while(max(abs(Y)/M,abs(Y_new)) > eps*(M-1)/sqrt(2) & l < 12){
 print(paste("Step n?", l, sep = ""))                    # Print stage number
 inverse = matrix(rinvGauss(M^l*N_base, 1/k, k), nrow = M^l, ncol = N_base)
 normal = matrix(rnorm(M^l*N_base), nrow = M^l, ncol = N_base)
 sim = m*t/M^l + theta*inverse + sigma*sqrt(inverse)*normal
 sim = S * exp(apply(sim, 2, cumsum))
 condition = ifelse(apply(sim, 2, max) < B, 1, 0)
 p1 = pmax((sim[nrow(a),]-K)*condition, 0)
 if(l>1){
  condition = ifelse(apply(sim[M*(1:(M^(l-1))),], 2, max)< B, 1,0)
  p0 = pmax((sim[nrow(a),]-K)*condition, 0)
 } else {
  condition = ifelse(sim < B, 1, 0)
```

```
  p0 = pmax((sim - K)*condition, 0)
 }
 h[l] <- t/(M^l)
 V[l] <- var(p1-p0)
 N[l] <- ceiling(2/eps^2*sqrt(V[l]*h[l])*sum(sqrt(V/h)))
 if(N[l] >= N_base){
  inverse2 = matrix(rinvGauss(M^l*(N[l]-N_base), 1/k, 1), nrow = M^l, ncol = N[l]-N_base)
  normal2 = matrix(rnorm(M^l*(N[l]-N_base)), nrow = M^l, ncol = N[l]-Nbase)
  sim2 = m*t/M^l + theta*inverse2 + sigma*sqrt(inverse2)*normal2
  sim2 = S*exp(apply(sim2, 2, cumsum))
  condition2 = ifelse(apply(sim2, 2, max) < B, 1, 0)
  sim = cbind(sim, sim2)
  condition = cbind(condition, condition2)
 }
 p1 = pmax((sim[nrow(a),]-K)*condition, 0)
 if(l>1){
  condition = ifelse(apply(sim[M*(1:(M^(l-1))),], 2, max)< B, 1,0)
  p0 = pmax((sim[nrow(a),]-K)*condition, 0)
 } else {
  condition = ifelse(sim < B, 1, 0)
  p0 = pmax((sim - K)*condition, 0)
 }
 Y <- Y_new               # Update
 Y_new <- mean(p1)-mean(p0)  # Update
 C <- C + Y_new           # Update price
 l <- l + 1               # Increment stage
}
C * exp(-r*t)
```

Standard Monte Carlo for Variance Gamma

```{r}
```

```
#Classical Monte Carlo for Variance Gamma
PayoffVG = function(numberSimu, S0, Time, deltaT, typeProduct, K, r, Barrier){
 numberPoints = Time/deltaT
 sigma = 0.1213
 theta = -0.1436
 kv = 0.1686
 m = r + (1/kv)*log(1+theta*kv-0.5*sigma^2*kv)
 Payoff = 0
 for(i in 1:numberSimu){
  Indicator = 1
  St = S0
  X0 = 0
  Xt = X0
  sumAsian = St
  for(j in 1:numberPoints){
   G = rgamma(1, shape = deltaT/kv, scale = kv)
   N = rnorm(1, mean = 0, sd = 1)
   Xt = Xt + m*deltaT + theta*G + sigma*sqrt(G)*N
   St = S0*exp(Xt)
   sumAsian = sumAsian + St
   if(St>Barrier & typeProduct == 'BarrierOption'){
    Indicator = 0
    break
   }
  }
  if(typeProduct == 'AsianOption'){
   Payoff = Payoff + max(0, sumAsian/numberPoints - K)
  }
  if(typeProduct == 'BarrierOption'){
   Payoff = Payoff + Indicator*max(0, St - K)
  }
 }
```

```
 Payoff = Payoff/numberSimu
 return(exp(-r*Time)*Payoff)
}

numberSimu = 25000
S0 = 100
Time = 1
deltaT = 0.005
typeProduct = 'AsianOption'
K = 100
r = 0.05
Barrier = 65
Result = PayoffVG(numberSimu, S0, Time, deltaT, typeProduct, K, r, Barrier)
print(Result)
```

Multilevel for Barrier Option with Normal Inverse Gaussian

```{r}
#Multilevel for Barrier Option with Normal Inverse Gaussian
library(SafeBayes)
eps = 0.005          # Sought precision for RMSE
Y = 1000             # Initialisation of Y
Y_new = 0            # Iteration Y update
M = 2                # M parameter
t = 1                # Time to maturity
h = t/M              # First time-step
N_base = M^14        # Base number of trajectories = 16384
S = 100
K = 100
theta = -0.1313
sigma = 0.1836
k = 15.5
r = 0.05
```

```
Barrier = 115
A = theta/sigma^2
B = sqrt(theta^2+(sigma^2)/k)/sigma^2
C = sqrt(theta^2+2*(sigma^2)/k)/(2*pi*sigma*sqrt(k))
m = r - (1/k) + pi*C*B*(1/k)*sqrt(B^2-(A+1)^2)
inverse = rinvGauss(N_base, 1/k, 1)
normal = rnorm(N_base)
sim = S*exp(m*t + theta*inverse + sigma*sqrt(inverse)*normal)
condition = ifelse(sim < B, 1, 0)
pay = pmax((sim - K)*condition, 0)
C = mean(pay)
V = 0
l = 1
N = N_base

while(max(abs(Y)/M,abs(Y_new)) > eps*(M-1)/sqrt(2) & l < 12){
 print(paste("Step n?", l, sep = ""))                    # Print stage number
 inverse = matrix(rinvGauss(M^l*N_base, 1/k, k), nrow = M^l, ncol = N_base)
 normal = matrix(rnorm(M^l*N_base), nrow = M^l, ncol = N_base)
 sim = m*t/M^l + theta*inverse + sigma*sqrt(inverse)*normal
 sim = S * exp(apply(sim, 2, cumsum))
 condition = ifelse(apply(sim, 2, max) < B, 1, 0)
 p1 = pmax((sim[nrow(a),]-K)*condition, 0)
 if(l>1){
  condition = ifelse(apply(sim[M*(1:(M^(l-1))),], 2, max)< B, 1,0)
  p0 = pmax((sim[nrow(a),]-K)*condition, 0)
 } else {
  condition = ifelse(sim < B, 1, 0)
  p0 = pmax((sim - K)*condition, 0)
 }
 h[l] <- t/(M^l)
 V[l] <- var(p1-p0)
```

```
 N[l] <- ceiling(2/eps^2*sqrt(V[l]*h[l])*sum(sqrt(V/h)))
 if(N[l] >= N_base){
  inverse2 = matrix(rinvGauss(M^l*(N[l]-N_base), 1/k, 1), nrow = M^l, ncol = N[l]-N_base)
  normal2 = matrix(rnorm(M^l*(N[l]-N_base)), nrow = M^l, ncol = N[l]-Nbase)
  sim2 = m*t/M^l + theta*inverse2 + sigma*sqrt(inverse2)*normal2
  sim2 = S*exp(apply(sim2, 2, cumsum))
  condition2 = ifelse(apply(sim2, 2, max) < B, 1, 0)
  sim = cbind(sim, sim2)
  condition = cbind(condition, condition2)
 }
 p1 = pmax((sim[nrow(a),]-K)*condition, 0)
 if(l>1){
  condition = ifelse(apply(sim[M*(1:(M^(l-1))),], 2, max)< B, 1,0)
  p0 = pmax((sim[nrow(a),]-K)*condition, 0)
 } else {
  condition = ifelse(sim < B, 1, 0)
  p0 = pmax((sim - K)*condition, 0)
 }
 Y <- Y_new               # Update
 Y_new <- mean(p1)-mean(p0)  # Update
 C <- C + Y_new           # Update price
 l <- l + 1               # Increment stage
}
C * exp(-r*t)
```