Estimating Value-at-Risk using Multilevel Monte Carlo Maximum Entropy method

Wenhui Gou

University of Oxford

A thesis submitted for the degree of

MSc Mathematical and Computational Finance

June 24, 2016

Acknowledgements

I would like to thank my supervisor for his guidance and help on this project. His guidance led me throughout the three months. I would also like to thank Prof. Alexey Chernov from University of Oldenburg. The explanations have been very helpful. I also want to thank my friends. They are always ready to help when I'm stuck somewhere on my way. Finally, thanks to my parents for encouraging and supporting me.

Abstract

We compare two methods to estimate Value-at-Risk of a complex portfolio made up of vanilla options: the standard Monte Carlo (MC) method and Multilevel Monte Carlo Maximum Entropy (MLMC-ME) method. The MC method will estimate the VaR as an empirical quantile from the random samples of the portfolio Profit-and-Loss (PnL). The MLMC-ME method will first estimate the generalised moments of the distribution of PnL using the MLMC method, with the level estimators formed by repricing different numbers of positions in the portfolio, and then reconstruct the distribution of PnL with the estimated series of generalised moments using the ME method. For an accuracy of ε, the cost required by MC is $\mathcal{O}(\varepsilon^{-2}K)$, where K is the number of positions in the portfolio. MLMC-ME requires only a cost of $\mathcal{O}(\varepsilon^{-2})$, giving large savings when K is large.

Contents

Chapter 1

Introduction

Banks are required by regulators to calculate at a daily basis the risk statistics, such as Value-at-risk (VaR) and Conditional VaR (CVaR, or Expected Shortfall, ES), etc. Mathematically, the α-VaR is the α quantile of the distribution of the portfolio Profit-and-Loss (PnL). The α-CVaR is the expectation of PnL conditional that the PnL is smaller than the α-VaR. In this paper, we denote profits as positive values and losses as negative values. As banks usually have a large and complex portfolio with different financial instruments, the distribution of PnL is not explicitly known. To estimate the risk statistics, random samples of PnL need to be simulated. This can be computationally expensive. In this paper, we introduce two methods to estimate VaR and compare their computational cost and accuracy: standard Monte Carlo (MC) method and Multilevel Monte Carlo (MLMC) Maximum Entropy (ME) method. We show that the combined MLMC-ME method is much cheaper computationally than the MC method, when the number of different positions in the portfolio is large.

The standard MC method is the brute force way to estimate VaR. It involves simulating random samples of the underlying risk factors, and then repricing all the financial assets in the portfolio under each risk scenario. The difference between the new portfolio value and the current value is the PnL. With the random samples of PnL, we can calculate the VaR as an empirical quantile. Assume that there are K different positions in the portfolio. For a given VaR level α and a given accuracy ε, the MC method requires a computational cost of $\mathcal{O}(\varepsilon^{-2}K)$. The K comes into play because K different positions need to be repriced for each random sample of PnL. As K increases, this cost will increase proportionally.

Instead, the key step of MLMC-ME method is to estimate the distribution density function of PnL. We first estimate the generalised moments of PnL using MLMC method, then reconstruct the PnL density function using ME method and finally obtain the quantiles and conditional expectations from this reconstructed distribution. We will form the MLMC estimator with different numbers of sub-samples of the portfolio on different levels, and prove that if an accuracy of ε is required for the moments, MLMC only requires a cost of $\mathcal{O}(\varepsilon^{-2})$, which is independent of K. We also show numerically that if the moments can be obtained at accuracy ε, then the VaR can be estimated at accuracy $\varepsilon' = c\varepsilon$, if other conditions are met. Overall, the MLMC-ME method requires a total cost of $\mathcal{O}(\varepsilon^{-2})$, which is much superior to MC method.

The ME method is a powerful tool to recover the distribution density function $\rho(x)$ when only a series of truncated generalised moments of the form $\mu_r = \int \phi_r(x)\rho(x)dx, \quad r = 0, ..., R$ are known. One example of this is just the usual moment functions $\mu_r = \int x^r \rho(x)dx$, but other forms of ϕ_r are also possible and may have different and sometimes better numerical features. There may exist many density functions that satisfy the moment constraints, among which the "best" one is the distribution with the Maximum Entropy, with the entropy defined as $-\int \rho(x)\ln(\rho(x))dx$. The details of this method will be discussed in Chapter 3. The MLMC method is one way to estimate the moments μ_r at a relatively low cost.

When estimating the moments, MLMC method uses the idea of nested simulation, which involves an outer-level simulation and an inner-level simulation. The outer-level simulation involves simulating the risk scenarios, the same as the MC method, under which we will calculate the PnL; the inner level involves random sub-sampling of the whole portfolio, i.e. instead of repricing the whole portfolio under each risk scenario, we independently and randomly select the positions to be repriced. The number of positions to be repriced on level l is 2^l. The MLMC method can achieve a cost of $O(\varepsilon^{-2})$ in estimating the moments, which is independent of K.

In this paper, we consider a portfolio composed of only call options that can be priced by Black-Scholes (BS) formula and do not need to be priced using numerical schemes, thus we exclude the complexity and randomness arising from pricing the options themselves. The key point that we would like to make is that when K increases, the MC cost will increase proportionally, while the MLML cost will stay roughly the same, thus MLMC can greatly reduce the computational cost when K is very large. When considering the process that K tends to infinity, we need to include more different positions in our portfolio. In this paper, we carry out our numerical analysis on an idealised portfolio composed of only call options on the same stock with different strike prices and maturity dates, each of them having equal weighting. The strike prices are equally spaced over a fixed interval. Every time we fix the total amount of money in the portfolio, so as to double K, we halve the weighting of each position and halve the interval of strike price or maturity. For the outer-level risk scenarios, we only consider the movement of the stock price S and hold all the other risk factors fixed. For a given S, the PnL of the k^{th} position is $L_k(S)$, $k = 1, ..., K$.

The outline of the paper is as follows. In Chapter 2, we will discuss the standard MC method. In Chapter 3, we will discuss in details the ME method, including the theory, the algorithm and practical issues. In Chapter 4, we will show the MLMC complexity theorem introduced in [5] and explain in details how the levels are formed. We will also calculate the constants in the complexity theorem and thus prove the order of the MLMC cost. In Chapter 5, we will demonstrate some numerical results, including the MLMC moments estimation, the ME reconstruction and compare the accuracy of MLMC-ME and MC methods using the same cost.

Chapter 2

Standard Monte Carlo method

Denote t as the current time point, h the VaR horizon, S_0 the current stock price, S the stock price in time h. The current value of the k^{th} position in the portfolio is $a_k V_k(S_0, t)$ where a_k is the unit of the option, V_k is the standard Black Scholes price of an option on one share of stock, and the value at the end of the horizon h is $a_k V_k(S, t+h)$, the PnL is $L_k(S) = a_k[V_k(S, t+h) - V_k(S_0, t)]$.

For simplicity, we only consider the change in the stock price. All the other risk factors, such as volatility, interest rate, will stay the same. Assume $\{S_{(n)}\}_{n=1}^N$ is a random sample of the stock price S at time $t+h$. A random sample of the portfolio PnL is

$$Y_{(n)} = \sum_{k=1}^{K} L_k(S_{(n)})$$

The empirical distribution of the PnL based on N independent random samples is

$$\widehat{F}_N(x) = \frac{1}{N}\sum_{n=1}^{N} \mathbb{1}\{Y_{(n)} \leq x\} \tag{2.0.1}$$

The α-VaR is calculated as:

$$\widehat{VaR}_{\alpha,N} = \widehat{F}_N^{-1}(\alpha) \tag{2.0.2}$$

According to [15] and [9], this empirical estimate of VaR has asymptotic distribution

$$\sqrt{N}(\widehat{VaR}_{\alpha,N} - VaR_\alpha) \sim \frac{\sqrt{\alpha(1-\alpha)}}{\rho(VaR_\alpha)}\mathcal{N}(0,1) \tag{2.0.3}$$

as $N \to \infty$, where $\mathcal{N}(0,1)$ is the standard normal distribution and $\rho(\cdot)$ is the probability density function of the portfolio PnL. If we want to achieve an accuracy of ε, i.e.

$$\sqrt{\mathbb{E}\left[\left(\widehat{VaR}_{\alpha,N} - VaR_\alpha\right)^2\right]} \leq \varepsilon \tag{2.0.4}$$

we need

$$\frac{1}{\sqrt{N}}\frac{\sqrt{\alpha(1-\alpha)}}{\rho(VaR_\alpha)} \leq \varepsilon \tag{2.0.5}$$

thus the number of independent samples of PnL should be roughly

$$N \approx \frac{\sqrt{\alpha(1-\alpha)}}{\rho(VaR_\alpha)}\varepsilon^{-2} \tag{2.0.6}$$

For each random sample, we need to evaluate the BS formula for K times, thus the total cost needed to achieve an accuracy of ε is roughly $\frac{\sqrt{\alpha(1-\alpha)}}{\rho(VaR_\alpha)}\varepsilon^{-2}K$.

It is also noteworthy that this cost will increase as we aim at more extreme quantile values. The density function of PnL usually decreases at the two tails and has a hump in the middle. As α tends to 0, ρ will usually become very small, thus the cost required will be fairly large.

Chapter 3

Maximum Entropy method

3.1 The theory

The goal of the Maximum Entropy method is to reconstruct a "best" distribution when only a series of generalised truncated moments is available. Mathematically the goal is to find ρ such that

$$\int \phi_r(x)\rho(x)dx = \mu_r, \quad r = 0, ..., R \tag{3.1.1}$$

where $\{\mu_r\}_{r=0}^R$ is a given series of constants and $\{\phi_r\}_{r=0}^R$ is a series of basis functions. This problem may have no solution or multiple solutions. The latter is always true when the series of moments is admissible, i.e. $(\mu_0, ..., \mu_R)$ are indeed the generalised moments of some probability density function. If multiple solutions exist, the "best" solution is the distribution function that has the maximum entropy, which in fact requires minimal prior information on the distribution. The Shannon entropy of a given distribution ρ is defined as

$$\mathbb{E}[-\ln(\rho(X))] = -\int \rho(x)\ln(\rho(x))dx \tag{3.1.2}$$

The $\{\phi_r\}_{r=0}^R$ in equation (3.1.1) is a series of basis functions, the following three series are commonly used:

1. Monomials: $\phi_r(x) = x^r$ for $r \geq 0$
2. Legendre polynomials: $P_0(x) = 1$, $P_1(x) = x$, $P_r(x) = \frac{2r-1}{r}xP_{r-1}(x) - \frac{r-1}{r}P_{r-2}(x)$ for $r \geq 2$
3. Fourier functions (trigonometric functions): $\phi_0(x) = 1$, $\phi_{2r-1}(x) = \sin(rx)$ and $\phi_{2r}(x) = \cos(rx)$ for $r \geq 1$

This problem now has the form of an optimisation problem:

$$\begin{aligned} &\text{maximise} \quad && -\int \rho(x)\ln(\rho(x))dx \\ &\text{subject to} && \int \phi_r(x)\rho(x)dx = \mu_r, \quad r = 0, ..., R \end{aligned} \tag{3.1.3}$$

After expressing the problem in the Lagrangian form, the solution to problem (3.1.3) is given by

$$\rho_R(x) = \exp\left[-\sum_{r=0}^{R} \lambda_r \phi_r(x)\right] \tag{3.1.4}$$

where the $\{\lambda_r\}_{r=0}^R$ are Lagrangien parameters to be obtained by solving the following set of non-linear equations

$$\int \phi_r(x)\exp\left[-\sum_{s=0}^{R} \lambda_s \phi_s(x)\right] dx = \mu_r, \quad r = 0, ..., R \tag{3.1.5}$$

The detailed derivation of this result can be found in for example [3].

When the exact moments $\{\mu_r\}_{r=0}^R$ are not known, but some estimations $\{\tilde{\mu}_r\}_{r=1}^R$ can be obtained using numerical methods, e.g. in our case, $\{\tilde{\mu}_r\}_{r=1}^R$ can be obtained by MC or MLMC method, the solution to the perturbed problem is

$$\tilde{\rho}_R(x) = \exp\left[-\sum_{r=0}^{R} \tilde{\lambda}_r \phi_r(x)\right] \tag{3.1.6}$$

where $\{\tilde{\lambda}_r\}_{r=0}^R$ is the solution to

$$\int \phi_r(x)\exp\left[-\sum_{s=0}^{R} \tilde{\lambda}_s \phi_s(x)\right] dx = \tilde{\mu}_r, \quad r = 0, ..., R \tag{3.1.7}$$

The accuracy of $\tilde{\rho}_R$ as an estimate of ρ depends on how large R is and how close the $\tilde{\mu}_r$ are to μ_r. In [2], the author uses Kullback-Leibler (KL) distance as the criterion of distance between two density functions:

$$D_{KL}(\rho||\eta) = \int \rho(x)\ln\frac{\rho(x)}{\eta(x)}dx \tag{3.1.8}$$

It is proved that

$$D_{KL}(\rho||\tilde{\rho}_R) = D_{KL}(\rho||\rho_R) + D_{KL}(\rho_R||\tilde{\rho}_R) \tag{3.1.9}$$

We call the first item the truncation error, the second the discretisation error. The author proves that for given R, as $\tilde{\mu}_r$ become more accurate, the discretisation error will become smaller; when μ_r are estimated within some accuracy by some numerical scheme and are random themselves, as R increases, the truncation error becomes smaller on average, but the probability that problem (3.1.3) does not admit a solution also increases. In our numerical experiments to estimate VaR, we won't determine R using rigorous mathematical deduction due to time limit, instead we use a moderate choice of R for a tradeoff between truncation error and stability.

3.2 The algorithm

The most complex step of the ME method is to solve the non-linear system (3.1.7) with $R+1$ equations. This can be done by Newton method. Following [14], denote $\boldsymbol{\lambda} = [\lambda_0, ..., \lambda_R]^t$, $\boldsymbol{\mu} = [\mu_0, ..., \mu_R]^t$

$$G_s(\boldsymbol{\lambda}) = \int \phi_s(x) \exp\left[-\sum_{r=0}^{R} \lambda_r \phi_r(x)\right] dx, \quad s = 0, ..., R \tag{3.2.1}$$

$$\boldsymbol{G}(\boldsymbol{\lambda}) = [G_0(\boldsymbol{\lambda}), ..., G_R(\boldsymbol{\lambda})]^t \tag{3.2.2}$$

The Jacobian matrix of $\boldsymbol{\lambda}$ is

$$\boldsymbol{H} = [h_{rs}] = \left[\frac{\partial G_r(\boldsymbol{\lambda})}{\partial \lambda_s}\right], \quad r, s = 0, ..., R \tag{3.2.3}$$

The matrix $\boldsymbol{H}$ is symmetric and we have

$$h_{rs} = h_{sr} = -\int \phi_r(x)\phi_s(x) \exp\left[-\sum_{p=0}^{R} \lambda_p \phi_p(x)\right] dx, \quad r, s = 0, ..., R \tag{3.2.4}$$

Given the n iteration $\boldsymbol{\lambda}^n$, the $n+1$ iteration of the Newton algorithm is

$$\boldsymbol{\lambda}^{n+1} - \boldsymbol{\lambda}^n = \boldsymbol{H}(\boldsymbol{\lambda}^n)^{-1}(\boldsymbol{\mu} - \boldsymbol{G}(\boldsymbol{\lambda}^n)) \tag{3.2.5}$$

A Matlab code is developed in [2] to solve problem (3.1.3). To solve (3.1.7), the author uses the damped Newton method. Instead of doing (3.2.5), set

$$\boldsymbol{\lambda}^{n+1} - \boldsymbol{\lambda}^n = p\boldsymbol{H}(\boldsymbol{\lambda}^n)^{-1}(\boldsymbol{\mu} - \boldsymbol{G}(\boldsymbol{\lambda}^n)) \tag{3.2.6}$$

where p is the damping factor between 0 and 1. The damped Newton method helps to improve convergence in some cases. In order to calculate the integrals in (3.2.1) and

(3.2.4), Gaussian quadrature rule is used (for Gaussian quadrature rule, see e.g. [17]), because of its sufficiently high order . The full algorithm works as follows.

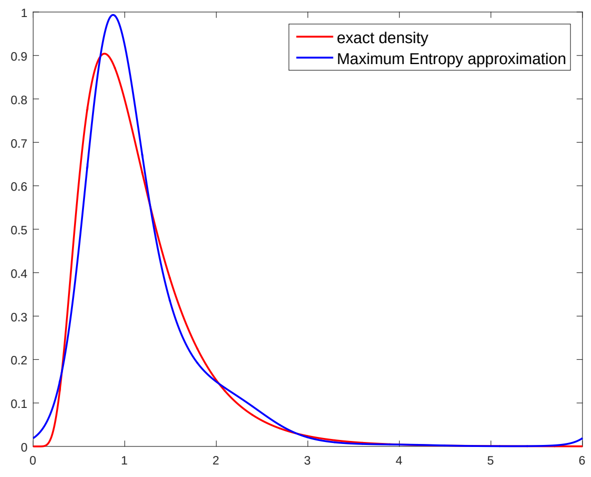
Algorithm 1: Maximum Entropy algorithm

```
Set $\phi$ to be monomials, Legendre polynomials or Fourier functions;
Set the support of $\rho$: $(a, b)$;
Determine the quadrature points in $(a, b)$;
Calculate $\phi$ at the quadrature points;
Set $\boldsymbol{\lambda} = \boldsymbol{\lambda}_0$;
while $|d\boldsymbol{\lambda}| > tolerance$ & $number\ of\ steps < maximum\ number\ of\ steps$ do
    Calculate $\boldsymbol{G}(\boldsymbol{\lambda})$;
    Calculate $\boldsymbol{H}(\boldsymbol{\lambda})$;
    Calculate $d\boldsymbol{\lambda} = damping * \boldsymbol{H}(\boldsymbol{\lambda})^{-1}(\boldsymbol{\mu} - \boldsymbol{G}(\boldsymbol{\lambda}))$;
    $\boldsymbol{\lambda} \leftarrow d\boldsymbol{\lambda} + \boldsymbol{\lambda}$;
end
Calculate $\rho_R$;
```

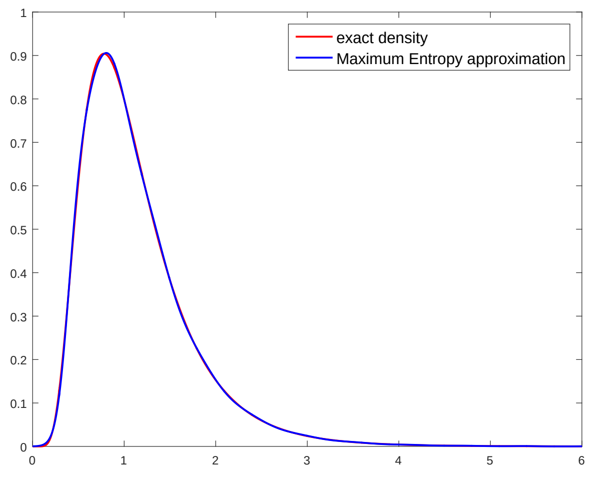
3.3 Numerical issues

Practically the performance of this algorithm can be influenced by many factors.

The number of moments $\boldsymbol{R}$: In [2], the author finds that bigger R increases the accuracy of the distribution approximation, but at the same time increases the probability that the perturbed problem does not admit a solution even though the problem using the exact moments does admit a solution. When solving the problem using Newton method, a matrix of dimension $(R+1)\times(R+1)$ needs to be inverted, thus a larger R also increases computation burden and will more often bring problems when inverting the matrices due to ill-conditioning. The following graphs show the ME approximation for the lognormal distribution with parameters $\mu = 0$, $\sigma = 0.5$ using up to the 5^{th} and 20^{th} Fourier functions respectively, with the moments estimated using 10^7 random samples. When using $R = 20$, the ME approximated distribution is very close to the true distribution.



(a) $R = 5$



(b) $R = 20$

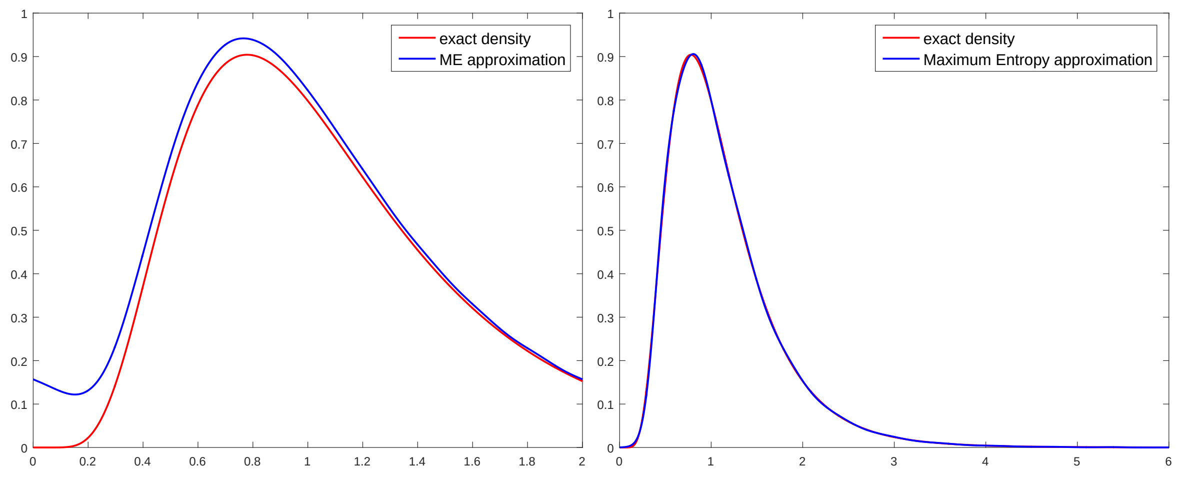
Figure 3.1: ME approximation with different R

The choice of ϕ: Theoretically all the three choices of ϕ will work, however, it does significantly affect the numerical results. The Legendre polynomials usually have a better performance than the monomials. Mathematically, the monomials and Legendre polynomials should be equivalent — the difference is entirely due to ill-conditioning and round-off error problems. Another observation is that when the kurtosis of the true distribution is high, Fourier functions work much better than the monomials and Legendre polynomials. When R is big, the monomials and Legendre polynomials will usually make the Jacobian matrix close to singular, thus cannot be inverted when doing the Newton iteration, which will stop us from increasing R further to achieve a better accuracy, thus limiting the usage of monomials and Legendre polynomials. We have done some tests on lognormal distributions with different parameters $\sigma_1 = 0.2$, $\sigma_2 = 0.5$ and $\sigma_3 = 1$ using the three series of polynomials. We estimate the moments using 10^7 random samples and increase R from 1, stop and record R when the algorithm cannot converge before generating the error of singular matrix. Even though the estimation of moments brings randomness into the results, it is still clear that in all cases, the Fourier functions work more stably than the monomials and Legendre polynomials.

Lognormal distribution parameter σ	kurtosis	Monomials	Legendre polynomials	Fourier functions
0.2	3.7	9	22	> 40
0.5	8.9	6	6	> 40
1.0	110	5	4	> 40

Table 3.1: The biggest R with which the algorithm works

The support of ρ (a, b): The numerical algorithm must take in a bounded support for ρ as an input. In Algorithm 1, this interval is provided by the user. Generally it should be neither too small nor too big. The algorithm will assume the integral of ρ over this interval is equal to 1. If the interval is too small, then the estimated distribution has high bias; if this interval is too big, the algorithm will usually have worse convergence than using an appropriate interval. For distributions with bounded support, this interval can be slightly wider than the support of the distribution; for distributions with unbounded support, such as the normal distribution, we can make (a, b) to be the interval over which the integral of the density function is equal to 99.99%. If the support of the underlying density is unknown, experimenting with it may help to get better results. The following graphs show the results of ME approximation of the lognormal distribution with $\sigma = 0.5$ using up to the 20^{th} function functions, the first figure using $(a, b) = (0, 2)$ and the second using $(a, b) = (0, 6)$ respectively.



(a) $(a, b) = (0, 2)$ (b) $(a, b) = (0, 6)$

Figure 3.2: ME approximation with different (a, b)

Chapter 4

Multilevel Monte Carlo method

Following the discussion in Chapter 3, instead of simulating random samples of PnL directly, we try to first recover its distribution using the ME method. In order to obtain the moments required for the ME reconstruction, in this section, we will discuss how the MLMC method can estimate the moments with a relatively low cost, i.e. $\mathcal{O}(\varepsilon^{-2})$. As K increases, the cost using MLMC would stay roughly the same.

4.1 MLMC and the complexity theorem

MLMC is a powerful technique introduced in [5]. It is used to reduce the computational cost by performing most calculations with low accuracy at low cost, and relatively few calculations with high accuracy at high cost. We want to estimate the expectation of an arbitrary random variable $\mathbb{E}[P]$ and there exist some approximations P_l to P at different accuracies that can be obtained at different costs. l represents the parameter of approximation, for example, it can be the grid space in solving PDE or SPDE.

For a given L, we have the following equality

$$\mathbb{E}[P_L] = \mathbb{E}[P_0] + \sum_{l=1}^{L} \mathbb{E}[P_l - P_{l-1}] \tag{4.1.1}$$

and we can form an estimator for $\mathbb{E}[P_L]$, where $\mathbb{E}[P_l - P_{l-1}]$ are estimated independently and P_l and P_{l-1} are formed using the same random factor such that these two values are close, thus the variance of $P_l - P_{l-1}$ is low.

An estimator based on N_0 simulations for $\mathbb{E}[P_0]$ is

$$\widehat{Y}_0 = \frac{1}{N_0} \sum_{n=1}^{N_0} P_0^{(0,n)}$$

An estimator of $\mathbb{E}[P_l - P_{l-1}]$ using N_l independent samples is

$$\frac{1}{N_l}\sum_{n=1}^{N_l}(P_l^{(l,n)} - P_{l-1}^{(l,n)})$$

Combining them gives an estimator for $\mathbb{E}[P_L]$

$$\widehat{Y} = \frac{1}{N_0}\sum_{n=1}^{N_0} P_0^{(0,n)} + \sum_{l=1}^{L}\left[\frac{1}{N_l}\sum_{n=1}^{N_l}\left(P_l^{(l,n)} - P_{l-1}^{(l,n)}\right)\right] \tag{4.1.2}$$

Let C_0, V_0 denote the cost and variance of one sample of P_0, and C_l, V_l denote those of $P_l - P_{l-1}$, then the total cost and variance of $\widehat{Y}$ are $\sum_{l=0}^{L} N_l C_l$ and $\sum_{l=0}^{L} N_l^{-1} V_l$. The mean square error (MSE) of $\widehat{Y}$ as an approximation to $\mathbb{E}[P]$ is

$$MSE = \mathbb{E}[(\widehat{Y} - \mathbb{E}[P])^2] = \mathbb{V}[\widehat{Y}] + (\mathbb{E}[\widehat{Y}] - \mathbb{E}[P])^2 \tag{4.1.3}$$

the first item corresponding to the variance of the estimator, the second corresponding the the bias. We also have

$$\mathbb{E}[\widehat{Y}] = \mathbb{E}[P_L], \quad \mathbb{V}[\widehat{Y}] = \sum_{l=0}^{L} N_l^{-1} V_l, \quad V_l = \mathbb{V}[P_l - P_{l-1}] \tag{4.1.4}$$

One sufficient condition to ensure the MSE to be less than ε^2 is that both $\mathbb{V}[\widehat{Y}]$ and $(\mathbb{E}[\widehat{Y}] - \mathbb{E}[P])^2$ are less than $\frac{1}{2}\varepsilon^2$. This idea leads to the following complexity theorem introduced in [5].

Theorem 4.1.1 (MLMC, [6]) *Let P denote a random variable, and P_l denote the corresponding level l approximation. If there exist independent estimators Y_l based on N_l Monte Carlo samples, each with expected cost C_l and variance V_l, and positive constants α, β, γ, c_1, c_2, c_3 such that $\alpha \geq \frac{1}{2}\min(\beta, \gamma)$ and*

1. $|\mathbb{E}[P_l - P]| \;\leq\; c_1 2^{-\alpha l}$
2. $\mathbb{E}[Y_l] = \begin{cases} \mathbb{E}[P_0], & l = 0 \\ \mathbb{E}[P_l - P_{l-1}], & l \geq 1 \end{cases}$
3. $V_l \leq c_2 2^{-\beta l}$
4. $C_l \leq c_3 2^{\gamma l}$

then there exists a positive constant c_4 such that for any $\varepsilon < e^{-1}$ there are values L and N_l for which the multilevel estimator

$$Y = \sum_{l=0}^{L} Y_l$$

has a mean square error with bound

$$\mathbb{E}\left[(\widehat{Y} - \mathbb{E}[P(X)])^2\right] \le \varepsilon^2$$

with a computational cost C with bound

$$\mathbb{E}[C] \le \begin{cases} c_4\varepsilon^{-2}, & \beta > \gamma \\ c_4\varepsilon^{-2}(\log\varepsilon)^2, & \beta = \gamma \\ c_4\varepsilon^{-2-(\gamma-\beta)/\alpha}, & \beta < \gamma \end{cases} \tag{4.1.5}$$

The proof of this theorem consist of two parts: first, L is chosen such that $(\mathbb{E}[\widehat{Y}]-\mathbb{E}[P])^2 < \frac{1}{2}\varepsilon^2$; then the optimal number of samples N_l on level l is proportional to $2^{-(\beta+\gamma)l/2}$ and the constant is chosen so that $\mathbb{V}[\widehat{Y}] < \frac{1}{2}\varepsilon^2$, and N_l is rounded up to the nearest integer. The cost on level l is proportional to $2^{(\gamma-\beta)l/2}$. Particularly, when $\beta > \gamma$, most of the calculations are performed on the coarsest level. More details can be found in [5] and [6].

In the context of estimating VaR, the first step is to estimate the expectation of some basis function ϕ_r on PnL

$$\mu_r = \mathbb{E}\left[\phi_r\left(\sum_{k=1}^{K} L_k(S)\right)\right], \quad 0 \le r \le R \tag{4.1.6}$$

then the μ_r can be used for the ME reconstruction. We will show later that we can form an MLMC estimator for the moments with $\alpha = 1$, $\beta = 2$ and $\gamma = 1$, thus the cost is $\mathcal{O}(\varepsilon^{-2})$.

Conditional on S, let X be a discrete uniform random variable from the sample $\{L_k\}_{k=1}^K$, i.e.

$$\mathbb{P}[X = L_k(S)|S] = \frac{1}{K}, \quad k = 1, ..., K \tag{4.1.7}$$

and $\{X_{(q)}\}_{q=1}^Q$ is a random sample of X, then $P = \phi_r(\sum_{k=1}^K L_k(S))$ can be seen as a conditional expectation

$$P = \phi_r(\mathbb{E}(\frac{K}{Q}\sum_{q=1}^{Q} X_{(q)}|S)) \tag{4.1.8}$$

From equation (4.1.8), we can form estimators of different accuracies using different Q. As a starting point, we assume that K is a power of $M = 2$. Denote $M_l = 2^l$ and

$$P_l = \phi_r(K\frac{1}{M_l}\sum_{m=1}^{M_l} X_{(m)}|S)) \tag{4.1.9}$$

is the level l approximation to P. In (4.1.9), though $X_{(m)}$ is a random sample from $\{L_k\}_{k=1}^K$, we don't really need to know all the values of L_k. We only need to randomly select the indices from $1, ..., K$ and reprice the positions whose indices are selected in $\{X_{(m)}\}_{m=1}^{M_l}$. Following the idea in [6], we can use an antithetic approach to construct a low variance estimate for $\mathbb{E}[Y_l] = \mathbb{E}[P_l - P_{l-1}]$. The antithetic method have been used by several authors, e.g. [4] and [8].

Denote $f = \phi_r$ as the polynomial function to be estimated. On level l, we first select M_l samples for the finer level, and then split them into two groups of equal size for the coarser levels and form Y_l as:

$$Y_l = f\left(\frac{K}{M_l}\sum_{m=1}^{M_l} X_{(m)}(S)\right) - \frac{1}{2}f\left(\frac{K}{M_{l-1}}\sum_{m=1}^{M_{l-1}} X_{(m)}(S)\right) - \frac{1}{2}f\left(\frac{K}{M_{l-1}}\sum_{m=M_{l-1}+1}^{M_l} X_{(m)}(S)\right) \tag{4.1.10}$$

We naturally have

$$\mathbb{E}\left[f\left(\frac{K}{M_l}\sum_{m=1}^{M_l} X_{(m)}(S)\right)\right] = \mathbb{E}[P_l] \tag{4.1.11}$$

$$\mathbb{E}\left[\frac{1}{2}f\left(\frac{K}{M_{l-1}}\sum_{m=1}^{M_{l-1}} X_{(m)}(S)\right) + \frac{1}{2}f\left(\frac{K}{M_{l-1}}\sum_{m=M_{l-1}+1}^{M_l} X_{(m)}(S)\right)\right] = \mathbb{E}[P_{l-1}] \tag{4.1.12}$$

The estimator of $\mathbb{E}[Y_l]$ based on N_l random samples is:

$$\begin{aligned}\widehat{Y}_l = \frac{1}{N_l}\sum_{n=1}^{N_l}\Bigg[& f\left(\frac{K}{M_l}\sum_{m=1}^{M_l} X_{(n,m)}(S_{(n)})\right) \\ & -\frac{1}{2}f\left(\frac{K}{M_{l-1}}\sum_{m=1}^{M_{l-1}} X_{(n,m)}(S_{(n)})\right) - \frac{1}{2}f\left(\frac{K}{M_{l-1}}\sum_{m=M_{l-1}+1}^{M_l} X_{(n,m)}(S_{(n)})\right)\Bigg]\end{aligned} \tag{4.1.13}$$

Until now, we have formed the estimators satisfying the second condition in Theorem 4.1.1. Next, in order to decide the optimal number of simulations to be carried out on each level, we need some knowledge on the constants α, β and γ as in Theorem 4.1.1. Only considering the cost of evaluating the BS formula and omitting the other additional calculations, we have naturally $C_l = 2^l$, as we only reprice the positions whose indices are sampled according to X, thus $\gamma = 1$.

To calculate α, we will try to estimate the speed at which $|\mathbb{E}[Y_l]|$ decreases. As

$$|\mathbb{E}[P_L - P]| = |\sum_{l=L}^{\infty}\mathbb{E}[P_l - P_{l+1}]| \leq \sum_{l=L}^{\infty}|\mathbb{E}[Y_{l+1}]|$$

then if $|\mathbb{E}[Y_{l+1}]|$ is $\mathcal{O}(2^{-\alpha l})$, $|\mathbb{E}[P_L - P]|$ is also $\mathcal{O}(2^{-\alpha l})$. We write

$$\mathbb{E}[Y_l] = \mathbb{E}[\mathbb{E}[Y_l|S]] \tag{4.1.14}$$

The variance of Y_l comes from two sources: the randomness of the outer-level risk factor S and the inner-level random sampling of the positions. Then according to the law of total variance,

$$\mathbb{V}(Y_l) = \mathbb{E}\left[\mathbb{V}(Y_l|S)\right] + \mathbb{V}[\mathbb{E}(Y_l|S)] \tag{4.1.15}$$

When sampling the M_l positions from the finite population, we can use either sampling with replacement or without replacement: with replacement means that the samples are independently selected and without replacement means that the samples are mutually different (under the condition that the number of samples does not exceed the number of positions). We will show in the following sections that there is no big difference in terms of C_l and V_l between these two methods.

We will prove in the following two subsections that $\alpha = 1$ and $\beta = 2$ for both sampling with replacement and without replacement, thus the computational complexity to ensure the moments are within an error of ε is $\mathcal{O}(\varepsilon^{-2})$. We also want to argue that as K increases, if nothing else changes this cost does not scale with K. However, it is quite tricky to define a limiting process for K with all the other factors staying the same. As we increase K, the portfolio must have more different positions, hence the new positions will have an effect on the variance of P_l. In order to eliminate the effects of those newly added positions, we construct the portfolios in a careful way such that as we double K to $2K$, the newly added positions come from the same universe of options, i.e. their strike prices and maturity dates follow a similar distribution to that of the already existing positions. For example, if we have two call options with strike prices K_1 and K_2 in the portfolio of size K, we add an option with strike price $\frac{1}{2}(K_1 + K_2)$ to the portfolio. The details of how the testing portfolios are formed will be shown in Chapter 5.

We will also show numerically if the 1^{st} order moment of PnL can be estimated within an accuracy of ε, with the same number of samples on each level, the estimations of the other moments can achieve an accuracy of $d_i\varepsilon$, $d = 2, ..., R$. Thus knowing this, we estimate the different moments at the same time with little cost. With all these moment functions, the VaR will be estimated within the accuracy of $c\varepsilon$, thus the error that translate from the moments estimation to the VaR estimation keeps the same order. Thus the cost of MLMC-ME stays at $\mathcal{O}(\varepsilon^{-2})$, while the cost of standard MC method is $\mathcal{O}(\varepsilon^{-2}K)$. Due to time limit, we won't try to determine how c can be determined from the other factors in the scope of this project.

4.2 Sub-sampling with replacement

We want to prove in this section that $\mathbb{E}[Y_l] \approx c_1 2^{-l}$ and $\mathbb{V}[Y_l] \approx c_2 2^{-2l}$ and c_1, c_2 do not depend on K, if the $X_{(m)}$ are sampled with replacement, i.e. independently.

For a given S, when M_{l-1} is large, $\frac{K}{M_{l-1}} \sum_{m=1}^{M_{l-1}} X_{(m)}(S)$ is equal to a constant times the average of M_{l-1} independent random samples, thus according Central Limit Theorem, it follows asymptotically normal distribution conditional on S with mean

$$\mathbb{E}\left[\frac{K}{M_{l-1}} \sum_{m=1}^{M_{l-1}} X_{(m)}|S\right] = \sum_{k=1}^{K} L_k(S) \triangleq L(S) \tag{4.2.1}$$

and variance

$$\begin{aligned}
&\mathbb{V}\left[\frac{K}{M_{l-1}} \sum_{m=1}^{M_{l-1}} X_{(m)}|S\right] \\
&= \frac{K^2}{M_{l-1}} \mathbb{V}\left[X|S\right] \\
&= \frac{K^2}{M_{l-1}} \{\mathbb{E}[X^2|S] - (\mathbb{E}[X|S])^2\} \\
&= \frac{1}{M_{l-1}} \left[K \sum_{k=1}^{K} L_k(S)^2 - (\sum_{k=1}^{K} L_k(S))^2\right] \\
&\triangleq \frac{1}{M_{l-1}} V(S)
\end{aligned} \tag{4.2.2}$$

Hence we can write

$$\frac{K}{M_{l-1}} \sum_{m=1}^{M_{l-1}} X_{(m)}(S) - L(S) \approx \sqrt{\frac{V(S)}{M_{l-1}}} Z_1 \tag{4.2.3}$$

where Z_1 follows the standard normal distribution. Similarly we write

$$\frac{K}{M_{l-1}} \sum_{M_{l-1}+1}^{M_l} X_{(m)}(S) - L(S) \approx \sqrt{\frac{V(S)}{M_{l-1}}} Z_2 \tag{4.2.4}$$

If f is twice differentiable, which is the case as f is one of the basis functions, a Taylor expansion of Y_l gives

$$Y_l \approx -\frac{1}{4} f''(L(S)) \frac{V(S)}{M_{l-1}} (Z_1 - Z_2)^2 \tag{4.2.5}$$

Z_1 and Z_2 are independent of S; the sub-sampling is performed with replacement, thus Z_1 and Z_2 are also independent. We always have

$$\mathbb{E}[(Z_1 - Z_2)^2] = 2, \quad \mathbb{V}[(Z_1 - Z_2)^2] = 8 \tag{4.2.6}$$

thus

$$\mathbb{E}[Y_l|S] = \mathbb{E}\left[-\frac{1}{4} f''(L(S)) \frac{V(S)}{M_{l-1}} (Z_1 - Z_2)^2 |S\right] = -\frac{1}{2} f''(L(S)) \frac{V(S)}{M_{l-1}} \tag{4.2.7}$$

$$\mathbb{V}[Y_l|S] = \mathbb{V}\left[-\frac{1}{4}f''(L(S))\frac{V(S)}{M_{l-1}}(Z_1 - Z_2)^2|S\right] = \frac{1}{2}f''(L(S))^2\frac{V(S)^2}{M_{l-1}^2} \tag{4.2.8}$$

Plugging (4.2.7) and (4.2.8) into (4.1.14) we have

$$\mathbb{E}[Y_l] = \mathbb{E}[\mathbb{E}[Y_l|S]] = \mathbb{E}\left[-\frac{1}{2}f''(L(S))\frac{V(S)}{M_{l-1}}\right] = -\frac{1}{2^l}\mathbb{E}[f''(L(S))V(S)] \tag{4.2.9}$$

We can see that $\mathbb{E}[Y_l] \approx c_1 2^{-l}$ thus $\alpha = 1$ and $c_1 = -\mathbb{E}\left[f''(L(S))V(S)\right]$.

Plugging (4.2.7) and (4.2.8) into (4.1.15) we have

$$\begin{aligned}\mathbb{V}[Y_l] &= \mathbb{V}[\mathbb{E}(Y_l|S)] + \mathbb{E}[\mathbb{V}(Y_l|S)] \\ &= \frac{1}{4}\mathbb{V}[f''(L(S))\frac{V(S)}{M_{l-1}}] + \frac{1}{2}\mathbb{E}[f''(L(S))\frac{V(S)^2}{M_{l-1}^2}] \\ &= \frac{1}{2^{2l}}\mathbb{V}[f''(L(S))V(S)] + 2\frac{1}{2^{2l}}\mathbb{E}[f''(L(S))^2V(S)^2]\end{aligned} \tag{4.2.10}$$

From here we can see that $\mathbb{V}[Y_l] \approx c_2 2^{-2l}$ thus $\beta = 2$ and

$$c_2 = \mathbb{V}[f''(L(S))V(S)] + 2\mathbb{E}[f''(L(S))^2V(S)^2] \tag{4.2.11}$$

where

$$V(S) = K\sum_{k=1}^{K} L_k^2(S) - \left(\sum_{k=1}^{K} L_k(S)\right)^2, \quad L(S) = \sum_{k=1}^{K} L_k(S)$$

Though we don't know how much c_1 and c_2 are exactly, we only need c_1 and c_2 to be independent of l and K. The independence with l is natural. The independence with K when K tends to infinity needs some condition on the portfolio. When we double K, we add the positions that are different but come from the same prior distribution, and at the same time halve the weighting of every position. In this way, in $L(s)$, each of the items in the sum will be roughly halved, but the number of items will be doubled, thus $L(S)$ does not change much overall. Similarly $V(S)$ will not change much either. This means that the constants in the MLMC cost analysis will not change, thus the cost will stay roughly the same, compared to the standard Monte Carlo case whether the cost will be simply doubled.

4.3 Sub-sampling without replacement

When the sub-sampling is done without replacement, all the $X_{(m)}$ must have distinct values for a given S. We first present two theorems on the sample mean when sampling without replacement. The first theorem follows the deduction in [19].

Theorem 4.3.1 *X is a discrete random variable that can take values from L_1,...,L_K with equal probability, $\bar{X} = \frac{1}{M}\sum_{m=1}^{M} X_{(m)}$ is the average of M random samples without replacement and $M \leq K$, then*

$$\mathbb{V}[\bar{X}] = \frac{K-M}{M(K-1)}\mathbb{V}[X] \tag{4.3.1}$$

Proof: First the expectation and variance of X are

$$\mathbb{E}[X] = \frac{1}{K}\sum_{m=1}^{K} L_m, \quad \mathbb{V}[X] = \frac{1}{K}\sum_{m=1}^{K} L_m^2 - \left(\frac{1}{K}\sum_{m=1}^{K} L_m\right)^2 \tag{4.3.2}$$

The expectation of $\bar{X}$ is the same as X, the variance is

$$\begin{aligned}\mathbb{V}[\bar{X}] &= \frac{1}{M^2}\sum_{m,n}\mathrm{Cov}(X_{(m)}, X_{(n)}) \\ &= \frac{1}{M^2}\left[\sum_{m=1}^{M}\mathbb{V}[X_{(m)}] + \sum_{m\neq n}\mathrm{Cov}(X_{(m)}, X_{(n)})\right] \\ &= \frac{1}{M}\mathbb{V}[X] + \frac{M-1}{M}\mathrm{Cov}(X_{(1)}, X_{(2)})\end{aligned} \tag{4.3.3}$$

Now,

$$\begin{aligned}&\mathrm{Cov}(X_{(1)}, X_{(2)}) = \mathbb{E}[X_{(1)}X_{(2)}] - (\mathbb{E}[X])^2 \\ &= \frac{1}{K(K-1)}\sum_{m\neq n} L_m L_n - (\frac{1}{K}\sum_{m=1}^{K} L_m)^2 \\ &= \frac{1}{K(K-1)}\left[\left(\sum_{m=1}^{K} L_m\right)^2 - \sum_{m=1}^{K} L_m^2\right] - \frac{1}{K^2}\left(\sum_{m=1}^{K} L_m\right)^2 \\ &= \frac{1}{K^2(K-1)}\left[\left(\sum_{m=1}^{K} L_m\right)^2 - K\sum_{m=1}^{K} L_m^2\right] \\ &= -\frac{1}{K-1}\mathbb{V}[X]\end{aligned} \tag{4.3.4}$$

Plugging this into (4.3.3), we have

$$\mathbb{V}(\bar{X}) = \frac{K-M}{M(K-1)}\mathbb{V}(X) \tag{4.3.5}$$

□

Theorem 4.3.2 *X is a discrete random variable that can take values from L_1,...,L_K with equal probability, $\bar{X}_1 = \frac{1}{M}\sum_{m=1}^{M} X_{(m)}$ and $\bar{X}_2 = \frac{1}{M}\sum_{m=M+1}^{2M} X_{(m)}$ are the averages* *of the first and second halves of $2M$ random samples selected without replacement and $2M \leq K$, then*

$$\mathrm{Cov}(\bar{X}_1, \bar{X}_2) = -\frac{1}{K-1}\mathbb{V}[X], \quad \mathrm{Corr}(\bar{X}_1, \bar{X}_2) = -\frac{M}{K-M} \tag{4.3.6}$$

Proof:

$$\begin{aligned}
&\mathrm{Cov}(\bar{X}_1, \bar{X}_2) \\
&= \mathrm{Cov}(\frac{1}{M}\sum_{m=1}^{M} X_{(m)}, \frac{1}{M}\sum_{n=M+1}^{2M} X_{(n)}) \\
&= \mathbb{E}\left[\frac{1}{M}\sum_{m=1}^{M} X_{(m)} \frac{1}{M}\sum_{n=M+1}^{2M} X_{(n)}\right] - (\mathbb{E}[X])^2 \\
&= \mathbb{E}\left[\frac{1}{M}\sum_{m=1}^{M} X_{(m)} \mathbb{E}\left[\frac{1}{M}\sum_{n=M+1}^{2M} X_{(n)} | X_{(1)}, ..., X_{(M)}\right]\right] - (\mathbb{E}[X])^2 \\
&= \mathbb{E}\left[\frac{1}{M}\sum_{m=1}^{M} X_{(m)} \frac{1}{K-M}(K\mathbb{E}[X] - \sum_{m=1}^{M} X_{(m)})\right] - (\mathbb{E}[X])^2 \\
&= \frac{M}{K-M}(\mathbb{E}[X])^2 - \frac{M}{K-M}\mathbb{E}\left[\left(\frac{1}{M}\sum_{m=1}^{M} X_{(m)}\right)^2\right]
\end{aligned} \tag{4.3.7}$$

From (4.3.5),

$$\mathbb{E}\left[\left(\frac{1}{M}\sum_{m=1}^{M} X_{(m)}\right)^2\right] = \mathbb{V}\left[\left(\frac{1}{M}\sum_{m=1}^{M} X_{(m)}\right)^2\right] + (\mathbb{E}[X])^2 = \frac{K-M}{M(K-1)}\mathbb{V}[X] + (\mathbb{E}[X])^2 \tag{4.3.8}$$

Plugging this into (4.3.7), we have

$$\mathrm{Cov}(\bar{X}_1, \bar{X}_2) = -\frac{1}{K-1}\mathbb{V}(X) \tag{4.3.9}$$

Additionally,

$$\mathrm{Corr}(\bar{X}_1, \bar{X}_2) = -\frac{M}{K-M} \tag{4.3.10}$$

□

When the sampling is performed without replacement, the expectation and variance of $\frac{K}{M_{l-1}}\sum_{m=1}^{M_{l-1}} X_{(m)}$ are respectively:

$$\mathbb{E}\left[\frac{K}{M_{l-1}}\sum_{m=1}^{M_{l-1}} X_{(m)}(S)\right] = L(S), \quad \mathbb{V}\left[\frac{K}{M_{l-1}}\sum_{m=1}^{M_{l-1}} X_{(m)}(S)\right] = \frac{K-M_{l-1}}{(K-1)M_{l-1}}V(S) \tag{4.3.11}$$

According to [11] and [16], under some conditions the normalised sample mean

$$\frac{\frac{K}{M_{l-1}}\sum_{m=1}^{M_{l-1}} X_{(m)}(S) - L(S)}{\sqrt{\frac{K-M_{l-1}}{(K-1)M_{l-1}}V(S)}}$$

also follows asymptotically normal distribution. We simply assume these conditions hold and write

$$\frac{K}{M_{l-1}}\sum_{m=1}^{M_{l-1}} X_{(m)}(S) - L(S) = \sqrt{\frac{K-M_{l-1}}{(K-1)M_{l-1}}V(S)}Z_1 \tag{4.3.12}$$

and similarly

$$\frac{K}{M_{l-1}}\sum_{m=M_{l-1}+1}^{M_l} X_{(m)}(S) - L(S) = \sqrt{\frac{K-M_{l-1}}{(K-1)M_{l-1}}V(S)}Z_2 \tag{4.3.13}$$

where Z_1 and Z_2 follow standard normal distribution and

$$\operatorname{Corr}(Z_1, Z_2) = \operatorname{Corr}(\frac{1}{M_{l-1}}\sum_{m=1}^{M_{l-1}} X_{(m)}, \frac{1}{M_{l-1}}\sum_{m=M_{l-1}+1}^{M_l} X_{(m)}) = -\frac{M_{l-1}}{K-M_{l-1}}$$

On the coarse levels, M_{l-1} are relatively small and the correlation is approximately 0; on the fine levels, M_ℓ are closes to $\frac{K}{2}$ thus the correlation is closer to -1.We also assume without proof (Z_1, Z_2) follows multivariate normal distribution with correlation matrix

$$[\sigma_{ij}]_{2\times 2} = \begin{bmatrix} 1 & \operatorname{Corr}(Z_1, Z_2) \\ \operatorname{Corr}(Z_1, Z_2) & 1 \end{bmatrix}$$

According to the properties of multivariate random variable in [19], we have

$$\mathbb{E}[Z_1^4] = \mathbb{E}[Z_2^4] = 3\sigma_{11}^2 = 3$$

$$\mathbb{E}[Z_1^3 Z_2] = \mathbb{E}[Z_1 Z_2^3] = 3\sigma_{11}\sigma_{12} = 3\operatorname{Corr}(Z_1, Z_2)$$

$$\mathbb{E}[Z_1^2 Z_2^2] = \sigma_{11}\sigma_{22} + 2\sigma_{12}^2 = 1 + 2\operatorname{Corr}(Z_1, Z_2)^2$$

The same as sampling with replacement, we have

$$Y_l \approx -\frac{1}{4}f''(L(S))\frac{K-M_{l-1}}{K-1}\frac{V(S)}{M_{l-1}}(Z_1 - Z_2)^2 \tag{4.3.14}$$

then

$$\mathbb{E}[(Z_1 - Z_2)^2] = \mathbb{E}[Z_1^2] + \mathbb{E}[Z_2^2] - 2\mathbb{E}[Z_1 Z_2] = \frac{2K}{K-M_{l-1}} \tag{4.3.15}$$

$$\begin{aligned} \mathbb{V}[(Z_1 - Z_2)^2] &= \mathbb{E}[(Z_1 - Z_2)^4] - (\mathbb{E}[(Z_1 - Z_2)^2])^2 \\ &= \mathbb{E}[Z_1^4 - 4Z_1^3 Z_2 + 6Z_1^2 Z_2^2 - 4Z_1 Z_2^3 + Z_2^4] - (\mathbb{E}[(Z_1 - Z_2)^2])^2 \\ &= 8[1 - \operatorname{Corr}(Z_1, Z_2)]^2 \\ &= \frac{8K^2}{(K-M_{l-1})^2} \end{aligned} \tag{4.3.16}$$

Plugging (4.3.15) and (4.3.16) into (4.1.14) we have

$$\begin{aligned}\mathbb{E}[Y_l] &= \mathbb{E}[\mathbb{E}[Y_l|S]] \\ &= \mathbb{E}[-\frac{1}{4}f''(L(S))\frac{V(S)}{M_{l-1}}(Z_1 - Z_2)^2] \\ &= -\frac{1}{2^l}\frac{K}{K-1}\mathbb{E}[f''(L(S))V(S)]\end{aligned} \tag{4.3.17}$$

We can see that $\mathbb{E}[Y_l]| \approx c_1 2^{-l}$ thus $\alpha = 1$ and

$$c_1 = |-\frac{K}{K-1}\mathbb{E}[f''(L(S))V(S)]|$$

Plugging (4.3.15) and (4.3.16) into (4.1.15), we have

$$\begin{aligned}\mathbb{V}[Y_l] &= \mathbb{V}[\mathbb{E}(Y_l|S)] + \mathbb{E}[\mathbb{V}(Y_l|S)] \\ &= \frac{1}{4}\mathbb{V}\left[f''(L(S))\frac{V(S)}{M_{l-1}}\frac{K}{K-1}\right] + \frac{1}{2}\mathbb{E}\left[f''(L(S))\frac{V(S)^2}{M_{l-1}^2}\frac{K^2}{(K-1)^2}\right] \\ &= \frac{1}{2^{2l}}\mathbb{V}[f''(L(S))V(S)\frac{K}{K-1}] + 2\frac{1}{2^{2l}}\mathbb{E}[f''(L(S))^2V(S)^2\frac{K^2}{(K-1)^2}] \\ &\approx \frac{1}{2^{2l}}\mathbb{V}[f''(L(S))V(S)] + 2\frac{1}{2^{2l}}\mathbb{E}[f''(L(S))^2V(S)^2]\end{aligned} \tag{4.3.18}$$

From here we can see that the variance of the MLMC estimator when doing sampling without replacement is nearly equal to the case with replacement, and $\beta = 2$ is also true in the case of sampling without replacement.

One difference between sampling with replacement and without replacement is: with replacement, the levels can be as many as needed, depending on the accuracy requirement; without replacement, the maximum of levels is $l_{max} = \log_2 K$, as the number of samples cannot exceed the size of the population. When the level goes up to $\log_2 K$, the MLMC estimator is unbiased.

Another thing to mention is that, until now, we have only considered the portfolio with K equal to a power of 2. When this is not the case, sampling with replacement can be generalised easily: it does not require K equal to a power of 2. For sampling without replacement, the estimator on the last level needs some adjustment. We propose two solutions. Assume $2^L < K < 2^{L+1}$. the levels can range from 0 to $L+1$. The first way is to change the level $L+1$ estimator to

$$Y_{L+1} = f\left(\frac{K}{K}\sum_{m=1}^{K} X_{(m)}(S)\right) - f\left(\frac{K}{M_L}\sum_{m=1}^{M_L} X_{(m)}(S)\right) \tag{4.3.19}$$

This correction makes the MLMC estimator unbiased as before, but the variance on the final level does not follow the previous behaviour, i.e. $\mathbb{V}[Y_{L+1}] \neq 2^{-2}\mathbb{V}[Y_L]$, but it should be very small anyway. The second adjustment is to couple some of the positions, i.e. if we have the option indices from 1 to K, options $(1, 2^L + 1)$ will be regarded as the first position, options $(2, 2^L + 2)$ will be regarded as the second, ..., options $(K - 2^L, K)$ will be the 2^L position, so that the population size when sampling without replacement is reduced to 2^L, thus we can follow the previous steps. When a pair is selected during the sub-sampling, e.g. the last position is selected, then options $(K - 2^L, K)$ will both be repriced.

4.4 Delta-Gamma-Theta approximation as control variate

The control variate method is a variance reduction technique when some highly correlated or anticorrelated variate is available, whose expectation is known or cheap to estimate.

Generally if we want to approximate $\mathbb{E}[f]$ using the simple average of Monte Carlo samples. There is another random variable g for which we know the expectation, then another unbiased estimator based on the simple average $\bar{f}'$ of N random samples of the variable

$$f' = f - \nu(g - \mathbb{E}[g])$$

satisfying $\mathbb{E}[f'] = \mathbb{E}[f]$. The variance of f' is

$$\mathbb{V}[f - \lambda(g - \mathbb{E}[g])] = \mathbb{V}[f] - 2\nu \operatorname{Cov}[f, g] + \nu^2 \mathbb{V}[g]$$

The minimum variance is $\mathbb{V}[f](1 - \operatorname{Corr}[f, g]^2)$ and it is achieved when $\nu = \frac{\operatorname{Cov}[f,g]}{\mathbb{V}[g]}$. Hence when the absolute value of the correlation is high, the use of control variate can be greatly reduce the variance, thus the computational cost.

In our context, Delta-Gamma-Theta (DGT) approximation is a very natural control variate for the accurate PnL for further variance reduction. It has been used by several authors, e.g. in [13]. Denote Δ_k, Γ_k and Θ_k, $k = 1, ..., K$ as the Delta, Gamma, Theta of each of the positions in the portfolio. The DGT approximation of a single position is simply the Taylor expansion of the option value function, i.e.

$$L_k^{DGT}(S) = \Delta_k * \Delta S + \frac{1}{2}\Gamma_k * \Delta S^2 + \Theta_k * \Delta t$$

where $\Delta S = S - S_0$. The Delta, Gamma and Theta of the portfolio are simply the sum of those of all the positions:

$$\Delta = \sum_{k=1}^{K} \Delta_k, \quad \Gamma = \sum_{k=1}^{K} \Gamma_k, \quad \Theta = \sum_{k=1}^{K} \Theta_k, \tag{4.4.1}$$

then the DGT approximation of the portfolio PnL is

$$L^{DGT}(S) = \Delta * \Delta S + \frac{1}{2}\Gamma * \Delta S^2 + \Theta * \Delta t \tag{4.4.2}$$

We write $L(S)$ and $L^{DGT}(S)$ simply as L and L^{DGT}. The Greek letters are calculated by the banks on a regular basis, thus can be achieved without any additional cost. A simple simulation shows that the correlation between L and L^{DGT} is close to 1, so we take $\nu = 1$ and write

$$\mathbb{E}[\phi_r(L)] = \mathbb{E}[\phi_r(L^{DGT})] + \mathbb{E}[\phi_r(L) - \phi_r(L^{DGT})] \tag{4.4.3}$$

According to equation (4.4.2), L^{DGT} is in fact a second order polynomial function of S. If $\{\phi_r\}_{r=1}^{R}$ are the monomials or Legendre polynomials, $\phi_r(L)$ is also a polynomial function of S. As S follows the lognormal distribution, the expectation of the form $\mathbb{E}[S^r]$ are explicitly known according to [18], thus $\phi_r(L)$ can be calculated explicitly. If $\{\phi_r\}_{r=1}^{R}$ are the Fourier functions, then $\mathbb{E}[X_r]$ can be calculated by numerical integration.

Thus instead of constructing an estimator for $\mathbb{E}[\phi_r(L)]$, we try to construct an estimator for $\mathbb{E}[\phi_r(L) - \phi_r(L^{DGT})]$. To estimate $\mathbb{E}[\phi_r(L) - \phi_r(L^{DGT})]$, we use the previously mentioned MLMC method, the level l estimator of $\mathbb{E}[\phi_r(L) - \phi_r(L^{DGT})]$ is

$$\begin{aligned} Y_l^{DGT} &= f\left(\frac{K}{M_l}\sum_{m=1}^{M_l} X_{(n,m)}(S_{(n)})\right) - f\left(\frac{K}{M_l}\sum_{m=1}^{M_l} X_{(n,m)}^{DGT}(S_{(n)})\right) \\ &- \frac{1}{2}\left[f\left(\frac{K}{M_{l-1}}\sum_{m=1}^{M_{l-1}} X_{(n,m)}(S_{(n)})\right) - f\left(\frac{K}{M_{l-1}}\sum_{m=1}^{M_{l-1}} X_{(n,m)}^{DGT}(S_{(n)})\right)\right] \\ &- \frac{1}{2}\left[f\left(\frac{K}{M_{l-1}}\sum_{m=M_{l-1}+1}^{M_l} X_{(n,m)}(S_{(n)})\right) - f\left(\frac{K}{M_{l-1}}\sum_{m=M_{l-1}+1}^{M_l} X_{(n,m)}^{DGT}(S_{(n)})\right)\right] \end{aligned} \tag{4.4.4}$$

Note that the X and X^{DGT} are sampled simultaneously, i.e. for a given $S_{(n)}$, $(X_{(n,m)}, X_{(n,m)}^{DGT})$ is a random sample from the uniform distribution on $\{(L_k, L_k^{DGT})\}_{k=1}^{K}$. The numerical results in section 5 will show that the variance of Y_l^{DGT} will be much smaller than that of Y_l: c_2 will be reduced greatly and β will stay the same, thus it is an effective way to reduce the computational cost for the level l estimator.

To conclude this chapter, we present the MLMC algorithm to estimate the moments within an accuracy of ε from [6] in Algorithm 2. On level l, the sub-sampling algorithm to estimate $\hat{Y}_l$ based on N_l samples is shown as Algorithm 3, where DGT is a dummy of whether the DGT approximation is used.

Algorithm 2: MLMC algorithm

```
Build the testing portfolio and calculate the values and greeks of each position;
Estimate $\alpha, \beta, \gamma$ with a given number of samples;
Start with $L = 1$ and initialise $L$ and $N_l$ for $l = 0, ..., L$;
while extra samples need to be evaluated do
    evaluate more samples on each level;
    update $\widehat{Y}_l$, $V_l$, $l = 0, ..., L$;
    calculate optimal $N_l$ using:
        $N_l^* = \left\lceil \varepsilon^{-2} \sqrt{V_l / C_l} \sum_l \sqrt{V_l C_l} \right\rceil$;
    weak convergence test;
    if not converged, set $L = L + 1$ and initialise $N_l$;
end
Set $\widehat{Y} = \sum_l \widehat{Y}_L$;
```

Algorithm 3: Sub-sampling algorithm

```
Initialise $f = \phi_r$, $DGT = 0$ or 1;
Generate $N_l$ samples of $S$;
for $i = 1$ to $N_l$ do
    Generate $M^l$ random samples from the finite population 1 to K: $I_{(i,1)}$,...,$I_{(i,M_l)}$;
    for $j = 1$ to $M_l$ do
        Reprice position $I_{(i,j)}$ and calculate PnL: $X_{(i,j)}$;
        if $DGT$ then
            Calculate DGT approximation of position $I_j$: $X_{(i,j)}^{DGT}$;
        else

        end
    end
    if $DGT$ then
        Calculate $Pf_{(i)} = f(\frac{K}{M_l} \sum_{j=1}^{M_l} X_{(i,j)}) - f(\frac{K}{M_l} \sum_{j=1}^{M_l} X_{(i,j)}^{DGT})$;
        Calculate $Pc_{(i)}^{(1)} = f(\frac{K}{M_l/2} \sum_{j=1}^{M_l/2} X_{(i,j)}) - f(\frac{K}{M_l/2} \sum_{j=1}^{M_l/2} X_{(i,j)}^{DGT})$;
        Calculate $Pc_{(i)}^{(2)} = f(\frac{K}{M_l/2} \sum_{j=M_l/2+1}^{M_l} X_{(i,j)}) - f(\frac{K}{M_l/2} \sum_{j=M_l/2+1}^{M_l} X_{(i,j)}^{DGT})$;
    else
        Calculate $Pf_{(i)} = f(\frac{K}{M_l} \sum_{j=1}^{M_l} X_{(i,j)})$;
        Calculate $Pc_{(i)}^{(1)} = f(\frac{K}{M_l/2} \sum_{j=1}^{M_l/2} X_{(i,j)})$;
        Calculate $Pc_{(i)}^{(2)} = f(\frac{K}{M_l/2} \sum_{j=M_l/2+1}^{M_l} X_{(i,j)})$;
    end
    Calculate $Y_l^{(i)} = Pf_{(i)} - \frac{1}{2} Pc_{(i)}^{(1)} - \frac{1}{2} Pc_{(i)}^{(2)}$;
end
if $DGT$ then
    Calculate $\hat{Y}_l = \frac{1}{N_l} \sum_{i=1}^{N_l} Y_l^{(i)} + E[Y_l^{DGT}]$;
else
    Calculate $\hat{Y}_l = \frac{1}{N_l} \sum_{i=1}^{N_l} Y_l^{(i)}$;
end
```

Chapter 5

Numerical results

In the numerical experiments, we consider portfolios composed of only call options on a single stock with current stock price $S_0 = 100$, interest rate $r = 0.05$, volatility $\sigma = 0.2$. The stock price under the physical measure follows lognormal distribution with annualised drift $\mu = 0.1$. We consider a VaR calculation horizon of 5 days, i.e. $\frac{5}{252}$ year. Table 5.1 shows the three portfolios of different K that we use to obtain the results in this paper. For the ME reconstruction, we use the Fourier functions with $R = 10$.

K	Portfolio value	Weighting	Strike price	Maturity
512 (2^9)	10,000	$\frac{1}{512}$	from 84 to 116 equally spaced by 2	from 0.68 to 1.32 equally spaced by 0.02
1024 (2^{10})	10,000	$\frac{1}{1024}$	from 84 to 116 equally spaced by 1	from 0.68 to 1.32 equally spaced by 0.02
2048 (2^{11})	10,000	$\frac{1}{2048}$	from 84 to 116 equally spaced by 0.5	from 0.68 to 1.32 equally spaced by 0.02

Table 5.1: Portfolios of different K

5.1 Matlab program

From the MLMC software provided provided in [6], adjustments are made to cater for this project.

First in order to verify that all the $R + 1$ moment functions will behave in the same way in terms of weak convergence and variance, the MLMC standard plot is generated for each of moment functions, either sampling with or without replacement, with or without DGT approximation. The program will do this for the three testing portfolios. The program will also record the variances of the estimators on each level under the different circumstances and save them for plotting purposes. The code is provided in appendix A.1.

After confirming that each of moment functions behaves in the same way with $\alpha = 1$ and $\beta = 2$, we adjust the code that it can generate the estimations for all the moments at the same time using the same random samples. When deciding the optimal number of samples needed on each level for a given ε, we adjust the program so as to target at the first moment function, then the error of the other moment functions will be scaled by a constant. With these estimated moment values, we run the ME algorithm to reconstruct the distribution and calculate the VaR. The program will also estimate the VaR using MC method with the same cost and calculate the accurate VaR. Another function is designed to carry out this estimation for a given number of times in order to estimate the RMSE of the MLMC-ME estimator and MC estimator. The program will also save the independent estimates for future plotting purposes. The Matlab codes are provided in appendix A.2.

Some other scripts are also designed to generate the plots in the following sections and will be presented in appendix A.3.

5.2 MLMC estimator

We first want to verify our previous conclusions in section 4 with the portfolios that we have constructed

1. the variance of the MLMC estimators on each level will not change with K
2. there is no big difference in variance between sampling with replacement and sampling without replacement
3. DGT approximation can greatly reduce the variance of the estimators

Figure 5.1 shows the comparison of variances of each level estimator when K is equal to 512, 1024 and 2048 respectively, either sampling with or without replacement, with or without DGT approximation. In each of the four graphs, we can tell that the variance is nearly the same on the same level for different K, showing that c_2 does not change with K. Comparing the upper two graphs or the lower two graphs, it is clear that there is no difference between the variances when sub-sampling with replacement and without replacement. Comparing the left two graphs or the right two graphs, we can tell that the

DGT approximation does not change of speed of variance reduction on each level, but reduces c_2 roughly by a factor of 10^{-3}.

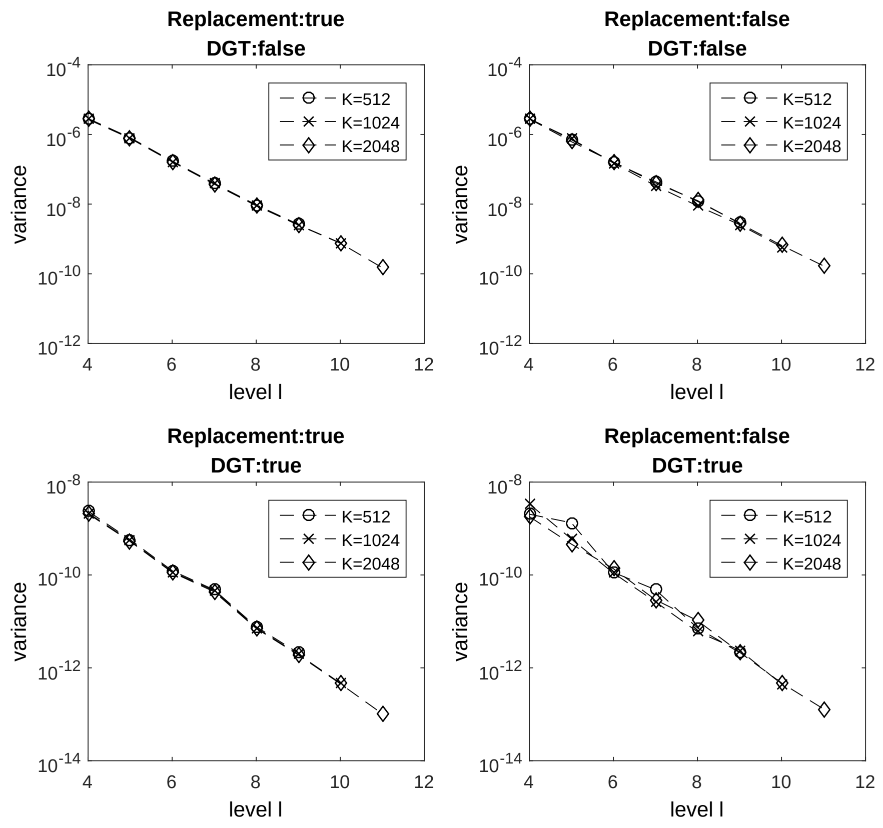


Figure 5.1: Variance of each level for different K

Figure 5.2 shows the MLMC standard plot when estimating the 2^{nd} order Fourier function sampling without replacement and with DGT approximation, and 5×10^4 samples are used to generate this plot. The other moment functions show the same trend, thus we won't show them here.

In the upper left graph, the solid line represents the variance of $\widehat{P_l}$ on each level and the dotted line represents the variance of $\widehat{P_l} - \widehat{P_{l-1}}$. Seen from the dotted line, $log_2 variance$ drops from -30 to -40 when l increases from 1 to 6, the slope is -2, verifying our conclusion that β is equal to 2. In the upper right graph, the solid line represents the absolute of the mean of $\widehat{P_l}$ on each level and the dotted line represents that of $\widehat{P_l} - \widehat{P_{l-1}}$. Seen from the dotted line, $log_2|mean|$ drops from -16 to -21 when l increases from 1 to 6, verifying our conclusion that α is equal to 1.

The middle left graph shows no problem with the consistency. The middle right graph shows the kurtosis of the MLMC estimator on each level. On some level, the kurtosis can be as high as several thousand. Particularly, the kurtosis is usually higher for the MLMC estimators with DGT approximation than without DGT approximation. This is not surprising, as when DGT approximation is used, most of the samples on each level are close to 0, usually when the stock price does not show extreme change, and only a few samples are not close to 0, thus yielding a high kurtosis. This also indicates that importance sampling, i.e. simulating more samples in the tails, may help to reduce the kurtosis.

The lower left graph shows the number of simulations needed on each level to achieve the given accuracy. It can be seen that N_l decreases with l. In addition, the maximum number of levels also increases as ε decreases. The lower right graph shows the value of $\varepsilon^2 Cost$ for each ε. The dotted line is flat, verifying that the cost for the MLMC estimator is $\mathcal{O}(\varepsilon^{-2})$.

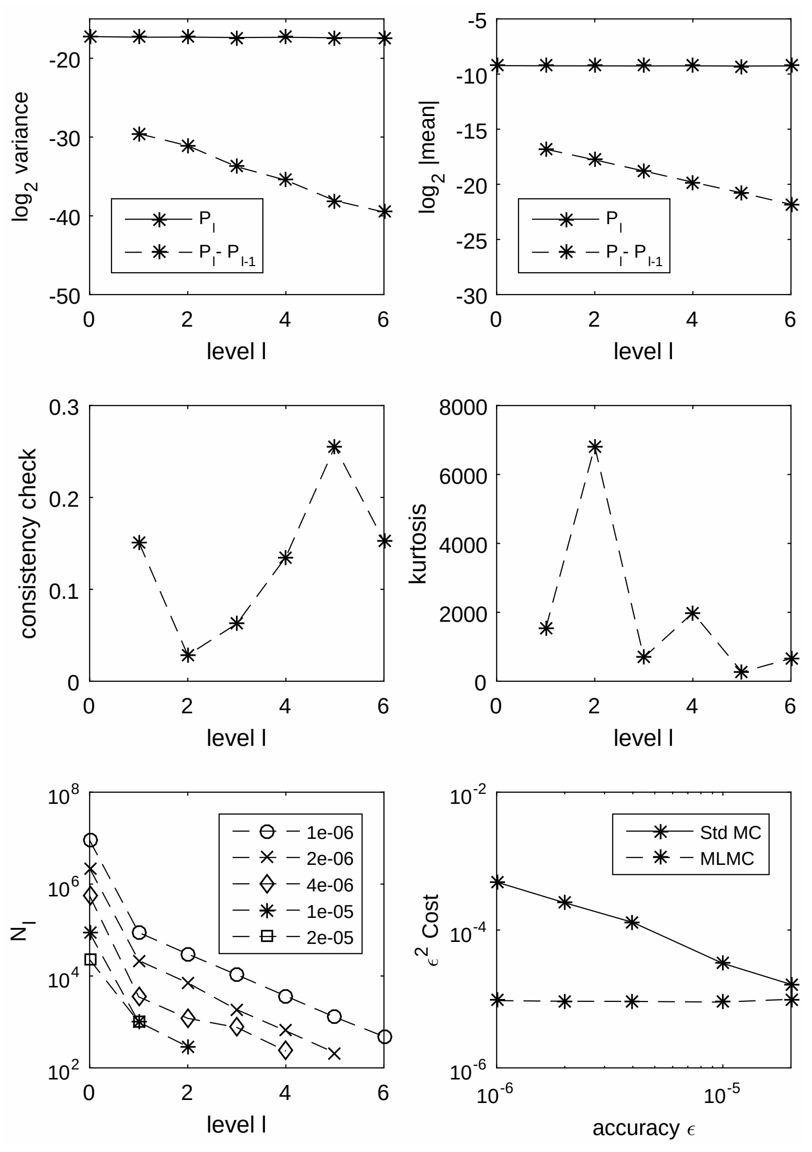


Figure 5.2: MLML plot of 2nd order moment

We have plotted the same graph for all the moment functions up to the order 10. One observation is that the variance of the estimators usually increase as r increases, while $\beta = 2$ is true for all the moments. If using Fourier functions, the kurtosis is slightly higher for the higher moments; if using Legendre or monomials polynomials, the kurtosis increases extremely fast as r increases.

5.3 ME reconstruction

The following two graphs show the numerical results related to the ME distribution reconstruction. Figure 5.3 gives a straightforward example of the recovered PnL distribution, using the Fourier moments estimated by MLMC using $R = 10$ and allowing for a fairly small ε for the moments, versus the smoothed empirical distribution from plain MC samples. The ME approximation is fairly accurate in this example.

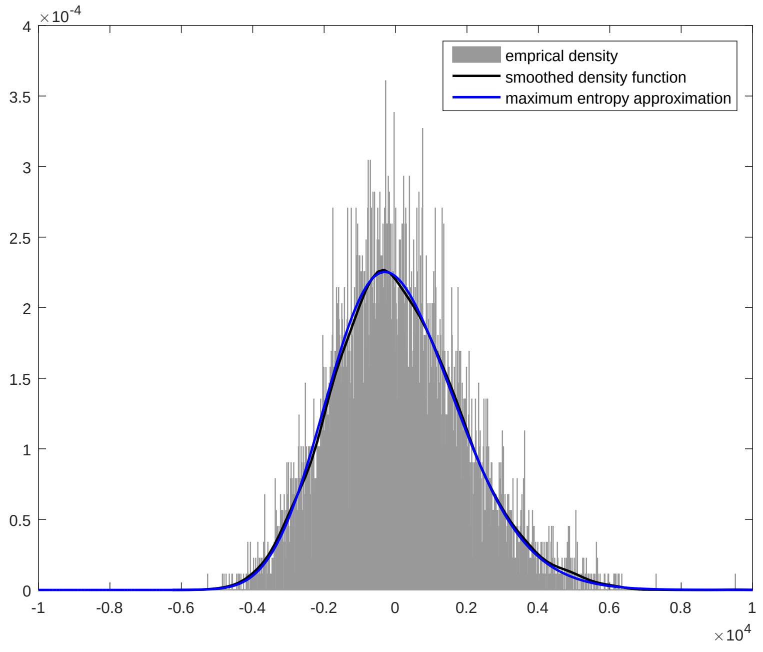


Figure 5.3: Empirical distribution of PnL v.s. ME reconstruction

Figure 5.4 is generated by changing the target ε of the first-order moment and then calculating the RMSE of the VaR estimations. This graph shows numerically how the errors in the moment estimations will translate into the errors in VaR estimation. When we talk about the errors in the moments, we take the error of the 1^{st} moment as the benchmark, the errors in the other moments are usually scaled by a constant. The slope of the loglog plot of RMSE of moments versus the RMSE of VaR is roughly 1, showing that the two RMSEs will be at the same order. It is also noteworthy that the RMSE is usually bigger for the more extreme quantiles. This is also true for the MC method. This confirms our conclusion that the order of the cost for estimating the VaR at accuracy ε is $\mathcal{O}(\varepsilon^{-2})$, the same as estimating the moments. Note that in this plot, we deliberately use a very large ε for the moment estimation, so that this error can dominate the errors rising from other steps, e.g. the choice of R, the accuracy of the Newton step etc. The target errors for the 1^{st} order moment are 0.04, 0.02, 0.01 and 0.005 and K is equal to 512.

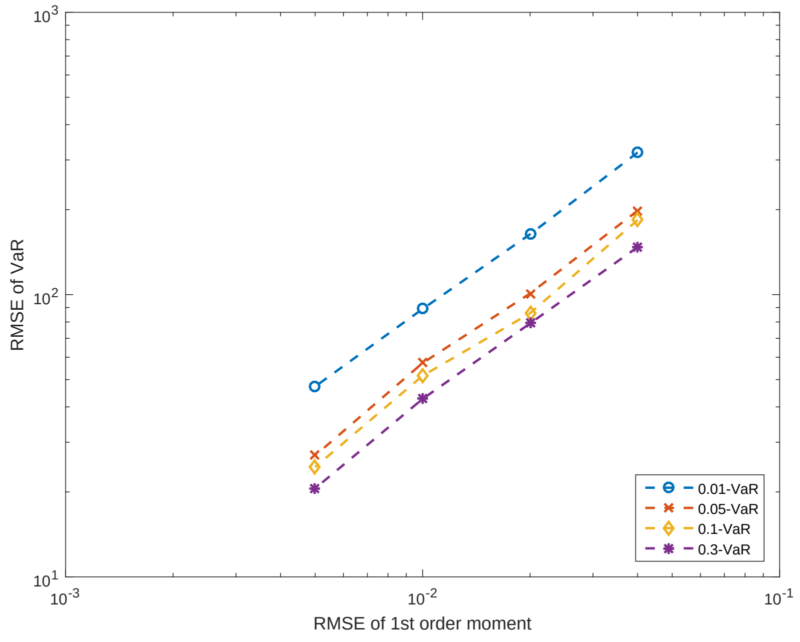


Figure 5.4: Linear relationship between RMSE of moments and RMSE of VaR

5.4 VaR variance

In this section we compare the cost and accuracy to estimate the VaR using MLMC-ME and MC methods respectively. For simplicity, instead of fixing the required accuracy, we fix the calculation cost and compare the accuracy of the VaR estimators. As we only use call options in our mock portfolios, the accurate VaR values are available. As the call option values are monotonously increasing functions of stock price, the quantile of PnL is simply the PnL calculated using the corresponding quantile of stock price. With the accurate VaR, we are able to estimate the root mean squared error (RMSE) of the MLMC-ME and MC estimators. The following graphs show the accuracy of MC, MLMC-ME and MLMC-ME estimators of 0.01-VaR, 0.05-VaR, 0.1-VaR and 0.3-VaR using around 5×10^5 calculations. The MLMC-ME estimators usually have a stable RMSE across all the three portfolios, while the MC estimators will have higher RMSE when K increases.

In addition, the MLMC-ME with DGT approximation has a lower RMSE than without DGT approximation.

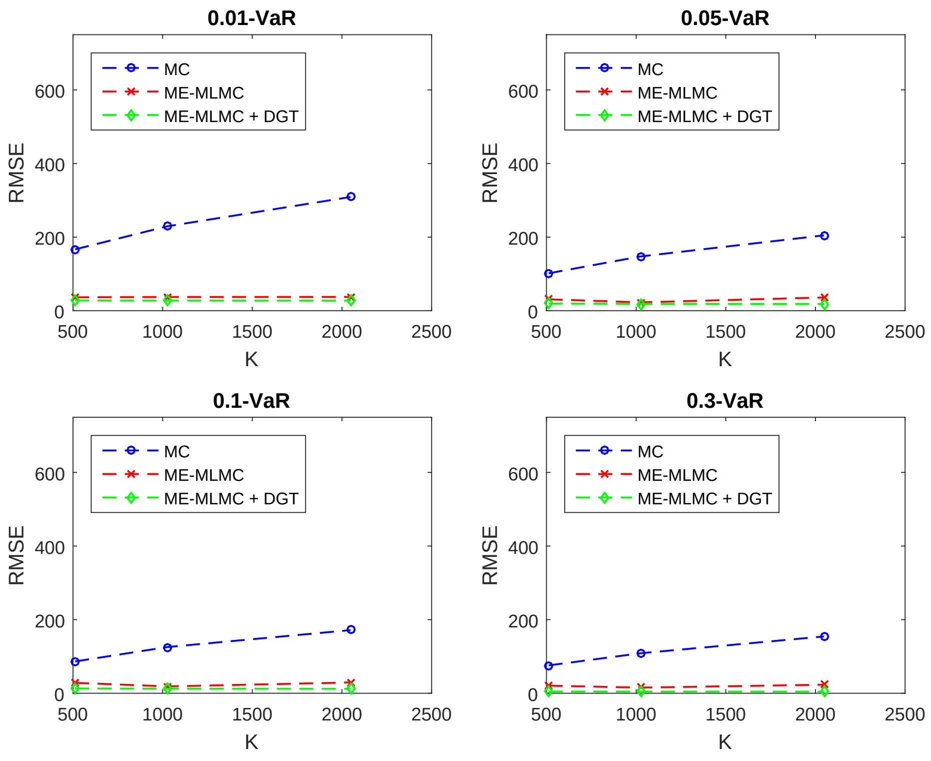


Figure 5.5: RMSE of MC and ME-MLMC estimators

Chapter 6

Conclusion and future work

In this paper, we have shown that the MLMC-ME method can achieve a cost of $\mathcal{O}(\varepsilon^{-2})$ when estimating VaR to the accuracy of ε and this does not depend on K, which is the number of potions in the portfolio, while the MC method requires a cost of $\mathcal{O}(\varepsilon^{-2}K)$. When K is big, MLMC-ME can result in huge computational cost savings.

Due to time limit, there are still some aspects that need future work. Even though we have carried out some numerical experiments on the accuracy of moments estimation and the accuracy of VaR, the rigorous relationship between them still needs to be explored. There are various steps in between where additional errors can arise. We need to understand how the errors will translate first from the moments estimation to the distribution estimation, then from the distribution to the VaR estimation. The first step can be affected by the choice of R, ϕ, the Newton steps, the shape of the distribution etc. The second step can be influenced by the VaR level α and the shape of the distribution.

Additionally, we simply assume the stock price follows lognormal distribution, but in reality, estimating the distribution of the risk factor itself is already non-trivial work. Usually these distributions have fat tails, thus may require us to explore further how the ME approximation will perform in the tails.

Bibliography

[1] Moulinath Banerjee. Simple Random Sampling, 2012. [Online; accessed 22-June-2016, URL: http://dept.stat.lsa.umich.edu/~moulib/sampling.pdf].

[2] Claudio Bierig and Alexey Chernov. Approximation of probability density functions by the Multilevel Monte Carlo Maximum Entropy method. *Journal of Computational Physics*, **314**:661–681, 2016.

[3] Jonathan M Borwein and Adrian S Lewis. Duality relationships for entropy-like minimization problems. *SIAM Journal on Control and Optimization*, **29**(2):325–338, 1991.

[4] Karolina Bujok, BM Hambly, and Christoph Reisinger. Multilevel simulation of functionals of Bernoulli random variables with application to basket credit derivatives. *Methodology and Computing in Applied Probability*, **17**(3):579–604, 2015.

[5] Michael B Giles. Multilevel Monte Carlo path simulation. *Operations Research*, **56**(3):607–617, 2008.

[6] Michael B Giles. Multilevel Monte Carlo methods. *Acta Numerica*, **24**:259–328, 2015.

[7] Michael B Giles. Numerical methods ii, 2016. [Online; accessed 22-June-2016, URL: http://people.maths.ox.ac.uk/~gilesm/mc/mc/lec3.pdf].

[8] Michael B Giles and Lukasz Szpruch. Antithetic multilevel Monte Carlo estimation for multidimensional SDEs. In *Monte Carlo and Quasi-Monte Carlo Methods 2012*, pages 367–384. Springer, 2013.

[9] P. Glasserman. *Monte Carlo Methods in Financial Engineering*. Stochastic Modelling and Applied Probability. Springer New York, 2010.

[10] Peter W Glynn. Importance sampling for Monte Carlo estimation of quantiles. In *Mathematical Methods in Stochastic Simulation and Experimental Design: Proceedings of the 2nd St. Petersburg Workshop on Simulation*, pages 180–185, 1996.

[11] Jaroslav Hajek. Limiting distributions in simple random sampling from a finite population. *Publ. Math. Inst. Hung. Acad. Sci., Ser. A*, **5**:361–374, 1960.

[12] R. Korn, E. Korn, and G. Kroisandt. *Monte Carlo Methods and Models in Finance and Insurance*. Chapman and Hall/CRC Financial Mathematics Series. CRC Press, 2010.

[13] Ralf Korn and Mykhailo Pupashenko. A new variance reduction technique for estimating Value-at-Risk. *Applied Mathematical Finance*, **22**(1):83–98, 2015.

[14] Ali Mohammad-Djafari. A Matlab program to calculate the maximum entropy distributions. In *Maximum Entropy and Bayesian Methods*, pages 221–233. Springer, 1992.

[15] Robert J Serfling. *Approximation theorems of mathematical statistics*, **162**. John Wiley & Sons, 2009.

[16] James H Stapleton. *Models for probability and statistical inference: theory and applications*, **652**. John Wiley & Sons, 2007.

[17] Wikipedia. Gaussian quadrature — Wikipedia, the free encyclopedia, 2016. [Online; accessed 22-June-2016, URL: https://en.wikipedia.org/wiki/Gaussian_quadrature].

[18] Wikipedia. Log-normal distribution — Wikipedia, the free encyclopedia, 2016. [Online; accessed 22-June-2016, URL: https://en.wikipedia.org/wiki/Log-normal_distribution#Geometric_moments].

[19] Wikipedia. Multivariate normal distribution — Wikipedia, the free encyclopedia, 2016. [Online; accessed 22-June-2016, URL: https://en.wikipedia.org/wiki/Multivariate_normal_distribution].

Appendix A

Matlab Code

The code of Maximum Entropy algorithm can be download from Prof. Alexey Chernov's website: https://www.uni-oldenburg.de/fileadmin/user_upload/mathe/personen/alexey.chernov/GeneralizedMaxEnt/GeneralizedMaxEnt.zip. We just build more tests based on this algorithm, thus won't show them here. The algorithm consists of the following functions:

- `MonomialArray.m`: function to evaluate the monomials
- `LegendreArray.m`: function to evaluate the Legendre polynomials
- `FourierArray.m`: function to evaluate the Fourier functions
- `GLquad.m`: function to compute the nodes and weights of Gauss-Legendre Quadrature Rule on an interval
- `generalizedME.m`: function to compute the lambda using Newton method
- `test.m`: test cases

The code of Multilevel Monte Carlo can be found on Prof. Mike Giles's website: http://people.maths.ox.ac.uk/~gilesm/mlmc/. The original algorithm consists of the following functions:

- `mlmc.m`: function to calculate the number of simulations on each level to achieve the given accuracy
- `mlmc_test.m`: function to estimate the weak error and variance of each level; estimate α, β, γ
- `mlmc_plot.m`: function to plot MLMC results
- `opre.m`: function specified to the application

The Matlab codes used in the project are listed below.

A.1 Moment estimation

The scripts `moment.m`, `mlmc_test_moment.m` and `mlmc_test.m` are used to run the standard MLMC tests on each of the moment functions, either sampling with or without replacement, either with or without DGT approximation for the three testing portfolios. `portfolio.m` is used to initialise the portfolio for a given number of strike prices and maturities.

`moment.m`

```
function moment
    close all; clear all;
    addpath('..');
    rng('default');
    global Poly;
    global S0 r sig K  T dK dT nT nK;   % parameter to specify the ...
        financial option
    global nPos;
    global pos V0 ∆0 gamma0 theta0;
    global horizon drift;
    global p;
    global M;
    global a b;
    global DG;
    global varWR varWOR varWR_DG varWOR_DG levels j;

    levels=zeros(20,3);
    varWR=zeros(20,3);varWOR=zeros(20,3);
    varWR_DG=zeros(20,3);varWOR_DG=zeros(20,3);
    for DG=0:1
    for replacement=0:1
    for j=1:3
        nK=2^(3+j);
        nT=2^5;
        portfolio(nK,nT);

        M     = 2;       % refinement cost factor
        N0    = 1000;    % initial samples on coarse levels
        Lmin  = 4;       % minimum refinement level
        Lmax  = round(log(nPos)/log(2));      % maximum refinement ...
            level
        Lstart = 3;       %starting level

        N      =50000;       % samples for convergence tests
        L      = Lmax;             % levels for convergence tests
        Eps    = 0.0002*[ 0.005 0.01 0.02 0.05 0.1 ];

        %     Poly = @LegendrePArray;
        %     Poly = @MonomialsArray;
        Poly = @FourierArray;

        R=10;
        a=-10000;
        b=10000;
        if DG
            moments=[0];
        else
            moments=[1];
        end

    for p=1:R
        %moments
        fprintf(1,'\n ---- Order-%d moments ---- \n',p);
        filename = ['moments' num2str(p) '_' num2str(replacement)...
            '_' num2str(DG) '_' num2str(nK) '_' num2str(nT)  ...
            '_' num2str(dK) '_'  num2str(dT)];
        fp = fopen([filename '.txt'],'w');

        temp=mlmc_test_moment(@opre_l, M, N,L, N0,Eps,Lmin,Lmax,...
           Lstart,fp, replacement);
        moments=[moments, temp];
        fclose(fp);

        nvert = 3;
        mlmc_plot(filename, nvert);
        print('-deps2',[filename '.eps'])
        %       size = size(3:4);
        %      set(gcf,'PaperSize',size)
        %      set(gcf,'PaperPosition',[0,0,6,8.4])
        %      print('-deps2','test.eps')
    end
    end
    end
        if DG;
            N1=2000000;
            ret = sig*sqrt(horizon)*normrnd(0,1, N1, 1)+ (drift-0.5*...
               sigˆ2)*horizon;
            S = S0*exp(ret);
            PnL_DG=(∆0'*pos)*(S-S0)...
                      +0.5*gamma0'*pos*power(S-S0,2)...
                  +theta0'*pos*horizon;
            moments_DG=Poly(PnL_DG,p,a,b);
            moments=moments+mean(moments_DG);
        end
    end

%-------------------------------------------------------
%
% level l estimator for Operations Research paper
%

function sums = opre_l(l,N, Lstart,replacement)
    % rng('default');
    global Poly;
    global S0 r sig K T;    % parameter to specify the financial ...
        option
    global nPos;
    global type pos V0 ∆0 gamma0 theta0;
    global horizon drift drift2;
    global p;
    global M;
    global a b;
    global DG;

    nf = M^l;
    nc = nf/M;
    sums(1:6) = 0;

    for N1 = 1:3000:N
        N2 = min(3000,N-N1+1);
        ret = sig*sqrt(horizon)*normrnd(0,1, N2, 1)+ (drift-0.5*sig...
           ^2)*horizon;
        S = S0*exp(ret);
        subf = zeros(N2, nf);
        for i=1:N2
            subf(i,:)=randsample(nPos,nf,replacement);
        end

        V1 = zeros(N2, nf);
        V2 = zeros(N2, nf);
        V = zeros(N2, nf);
        PnL = zeros(N2, nf);
        PnL_DG = zeros(N2, nf);
        PnL_f = zeros(N2, 1);
        PnL_f_DG = zeros(N2, 1);
        PnL_c =zeros(N2, M);
        PnL_c_DG =zeros(N2, M);
        Pf = zeros(N2,p+1);
        Pc = zeros(N2,p+1);

        for i=1:nf
            V(:,i)=blsprice(S, K(subf(:,i)), r, T(subf(:,i))-horizon,...
                sig);
            PnL(:,i)= (V(:,i)-V0(subf(:,i))).*pos(subf(:,i));
            PnL_DG(:,i)= (∆0(subf(:,i)).*(S-S0)...
                 +0.5*gamma0(subf(:,i)).*power(S-S0,2)...
             +theta0(subf(:,i))*horizon).*pos(subf(:,i));
        end
        PnL_f = sum(PnL, 2)/nf*nPos;
        PnL_f_DG = sum(PnL_DG, 2)/nf*nPos;
        if DG
            Pf = Poly(PnL_f,p,a,b)-Poly(PnL_f_DG,p,a,b);
        else
            Pf = Poly(PnL_f,p,a,b);
        end
        if l>Lstart
            PnL_c = zeros(N2, M);
            for j=1:M
                PnL_c(:,j)=sum(PnL(:,nc*(j-1)+1:nc*j),2)/nc*nPos;
                PnL_c_DG(:,j)=sum(PnL_DG(:,nc*(j-1)+1:nc*j),2)/nc*...
                    nPos;
                if DG
                    Pc = Pc+Poly(PnL_c(:,j),p,a,b)/M-Poly(PnL_c_DG(:,...
                        j),p,a,b)/M;
                else
```

```
                    Pc = Pc+Poly(PnL_c(:,j),p,a,b)/M;
                end
            end
        end

         sums(1) = sums(1) + sum(Pf(:,p+1)-Pc(:,p+1));
         sums(2) = sums(2) + sum((Pf(:,p+1)-Pc(:,p+1)).^2);
         sums(3) = sums(3) + sum((Pf(:,p+1)-Pc(:,p+1)).^3);
         sums(4) = sums(4) + sum((Pf(:,p+1)-Pc(:,p+1)).^4);
         sums(5) = sums(5) + sum(Pf(:,p+1));
         sums(6) = sums(6) + sum(Pf(:,p+1).^2);
end
```

mlmc_test_moment.m

```
%
% function mlmc_test(mlmc_fn,M, N,L, N0,Eps,Lmin,Lmax, fp)
%
% multilevel Monte Carlo test routine
%
% sums = mlmc_fn(l,N)     low-level routine
%
% inputs:  l = level
%          N = number of paths
%
% output: sums(1) = sum(Pf-Pc)
%         sums(2) = sum((Pf-Pc).^2)
%         sums(3) = sum((Pf-Pc).^3)
%         sums(4) = sum((Pf-Pc).^4)
%         sums(5) = sum(Pf)
%         sums(6) = sum(Pf.^2)
%
% M      = refinement cost factor (2^gamma in general MLMC Thm)
%
% N      = number of samples for convergence tests
% L      = number of levels for convergence tests
%
% N0     = initial number of samples for MLMC calcs
% Eps    = desired accuracy array for MLMC calcs
%

function Pa=mlmc_test_moment(mlmc_fn,M, N,L, N0,Eps,Lmin,Lmax,Lstart,...
    fp, replacement)
    global dK dT nK nT;   % parameter to specify the financial option
    global varWR varWOR varWR_DG varWOR_DG levels j;
    global DG;
    %
    % first, convergence tests
    %
    PRINTF2(fp,'\n');
    PRINTF2(fp,'...
        **********************************************************\n')...
        ;
    PRINTF2(fp,'*** Convergence tests, kurtosis, telescoping sum ...
        check ***\n');
    PRINTF2(fp,'...
        **************************************************************\n')...
        ;
    PRINTF2(fp, 'dK: %f   dT: %f nK: %d  nT: %d replacement: %s DG: %...
        s\n', ...
        dK, dT, nK, nT,int2str(replacement),int2str(DG));
    PRINTF2(fp,'\n l   ave(Pf-Pc)    ave(Pf)    var(Pf-Pc)    var(Pf)'...
        );
    PRINTF2(fp,'    kurtosis     check \n-------------------------');
    PRINTF2(fp,'--------------------------------------------------\n'...
        );

%     rng('default');    % reset random number generator
    P=0;
    del1 = [];
    del2 = [];
    var1 = [];
    var2 = [];
    kur1 = [];
    chk1 = [];
    cost = [];
    est=[];

    for l = Lstart:Lmax
    %   disp(sprintf('%d',l))
      tic;
      sums = feval(mlmc_fn,l,N, Lstart, replacement);
      cost = [ cost toc ];
      sums = sums/N;
      if (l==Lstart)
        kurt = 0.0;
      else
        kurt = (      sums(4)             ...
                  - 4*sums(3)*sums(1)     ...
                  + 6*sums(2)*sums(1)^2   ...
                  - 3*sums(1)*sums(1)^3 ) ...
                / (sums(2)-sums(1)^2)^2;
      end

      del1 = [del1 sums(1)];
      del2 = [del2 sums(5)];
      var1 = [var1 sums(2)-sums(1)^2 ];
      var2 = [var2 sums(6)-sums(5)^2 ];
      var2 = max(var2, 1e-10);  % fix for cases with var=0
      kur1 = [kur1 kurt ];

      if l==Lstart
        check = 0;
      else
        check = abs(       del1(l-Lstart+1)  +      del2(l-Lstart)  -...
                   del2(l-Lstart+1)) ...
          /     ( 3.0*(sqrt(var1(l-Lstart+1)) + sqrt(var2(l-Lstart)) +...
                sqrt(var2(l-Lstart+1)) )...
          /sqrt(N));
      end
      chk1 = [chk1 check];


      PRINTF2(fp,'%2d   %8.4e  %8.4e  %8.4e  %8.4e  %8.4e  %8.4e \n',...
        ...
            l,del1(l-Lstart+1),del2(l-Lstart+1),var1(l-Lstart+1),...
                ...
            var2(l-Lstart+1),kur1(l-Lstart+1),chk1(l-Lstart+1));
        P=P+del1(l-Lstart+1);
    end


    %
    % print out a warning if kurtosis or consistency check looks bad
    %

    if ( kur1(end) > 100.0 )
      PRINTF2(fp,'\n WARNING: kurtosis on finest level = %f \n',kur1...
          (end));
      PRINTF2(fp,' indicates MLMC correction dominated by a few rare ...
          paths; \n');
      PRINTF2(fp,' for information on the connection to variance of ...
          sample variances,\n');
      PRINTF2(fp,' see http://mathworld.wolfram.com/...
          SampleVarianceDistribution.html\n\n');
    end

    if ( max(chk1) > 1.0 )
      PRINTF2(fp,'\n WARNING: maximum consistency error = %f \n',max(...
          chk1));
      PRINTF2(fp,' indicates identity E[Pf-Pc] = E[Pf] - E[Pc] not ...
          satisfied \n\n');
    end

    %
    % use linear regression to estimate alpha, beta and gamma
    %

    L1 = 2;
    L2 = L-Lstart+1;
    % L2 = L;
    pa    = polyfit(L1:L2,log2(abs(del1(L1:L2))),1);  alpha = -pa(1);
    pb    = polyfit(L1:L2,log2(abs(var1(L1:L2))),1);  beta  = -pb(1);
    if replacement
        if DG
            varWR_DG(1:(L2-L1+1),j)=var1(L1:L2);
        else
            varWR(1:(L2-L1+1),j)=var1(L1:L2);
        end
    else
        if DG
            varWOR_DG(1:(L2-L1+1),j)=var1(L1:L2);
        else
            varWOR(1:(L2-L1+1),j)=var1(L1:L2);
        end
    end
    levels(1:(L2-L1+1),j)=L1+Lstart-1:L2+Lstart-1;
    % just last two points to minimise effect of MATLAB overhead
    gamma = log2(cost(end)/cost(end-1));

    PRINTF2(fp,'\n...
       *******************************************************\n');
    PRINTF2(fp,'*** Linear regression estimates of MLMC parameters ...
        ***\n');
    PRINTF2(fp,'...
        *******************************************************\n');
    PRINTF2(fp,'\n alpha = %f  (exponent for MLMC weak convergence)\n...
        ',alpha);
    PRINTF2(fp,' beta  = %f  (exponent for MLMC variance) \n',beta);
    PRINTF2(fp,' gamma = %f  (exponent for MLMC cost) \n',gamma);

    %
    % second, mlmc complexity tests
    %

    PRINTF2(fp,'\n');
    PRINTF2(fp,'***************************** \n');
    PRINTF2(fp,'*** MLMC complexity tests *** \n');
    PRINTF2(fp,'***************************** \n\n');
    PRINTF2(fp,'   eps       value    mlmc_cost   std_cost  savings ...
            N_l \n');
    PRINTF2(fp,'...
        ------------------------------------------------------------ \n...
        ');

    rng('default');    % reset random number generator

    alpha = max(alpha,0.5);
    beta  = max(beta,0.5);
    gamma = log2(M);
    theta = 0.25;

    for i = 1:length(Eps)
      eps = Eps(i);
      [P, Nl] = mlmc_moment(Lmin,Lmax,Lstart,N0,eps,mlmc_fn,alpha,...
         beta,gamma,replacement);
      if i==1; Pa=P; end;
      l = length(Nl)-1;

      mlmc_cost = sum(Nl.*M.^(0:l));
      l2 = min(l+1,length(var2));
      std_cost  = var2(l2)*M^l / ((1.0-theta)*eps^2);


      PRINTF2(fp,'%.3e  %.3e  %.3e  %.3e  %7.2f ', ...
          eps, P, mlmc_cost, std_cost, std_cost/mlmc_cost);
      PRINTF2(fp,'%9d',Nl);
      PRINTF2(fp,'\n');
    end

    PRINTF2(fp,'\n');

    end

    %
    % function to print to both a file and stdout
    %

    function PRINTF2(fp,varargin)
      fprintf(fp,varargin{:});
      fprintf( 1,varargin{:});
    end
```

mlmc_moment.m

```
% function [P, Nl] = mlmc(Lmin,Lmax,N0,eps,mlmc_l, alpha,beta,gamma)
%
% multi-level Monte Carlo estimation
%
% P      = value
% Nl     = number of samples at each level
%
% Lmin   = minimum level of refinement       ≥ 2
% Lmax   = maximum level of refinement       ≥ Lmin
% N0     = initial number of samples         > 0
% eps    = desired accuracy (rms error)      > 0
%
% alpha -> weak error is  O(2^{-alpha*l})
% beta  -> variance is    O(2^{-beta*l})
% gamma -> sample cost is O(2^{gamma*l})    > 0
%
% if alpha, beta are not positive then they will be estimated
%
% mlmc_l = function for level l estimator
%
% sums = mlmc_fn(l,N)     low-level routine
%
% inputs:  l = level
%          N = number of paths
%
% output: sums(1) = sum(Y)
%         sums(2) = sum(Y.^2)
%         where Y are iid samples with expected value:
%         E[P_0]           on level 0
%         E[P_l - P_{l-1}] on level l>0

function [P, Nl] = mlmc_moment(Lmin,Lmax,Lstart, N0,eps,mlmc_l, ...
    alpha_0,beta_0,gamma,replacement)

%
% check input parameters
%
  if (Lmin<2)
    error('error: needs Lmin ≥ 2');
  end

  if (Lmax<Lmin)
    error('error: needs Lmax ≥ Lmin');
  end

  if (Lmin<Lstart)
      error('error: needs Lstart ≥ Lmin');
  end

  if (N0≤0 || eps≤0 || gamma ≤ 0)
    error('error: needs N>0, eps>0, gamma>0 \n');
  end

%
% initialisation
%
  alpha = max(0, alpha_0);
  beta  = max(0, beta_0);

  theta = 0.25;

  L = Lmin-Lstart;

  Nl(1:L+1)       = 0;
  suml(1:2,1:L+1) = 0;
  dNl(1:L+1)      = N0;

  while sum(dNl) > 0

%
% update sample sums
%
    for l=0:L
      if dNl(l+1) > 0
        sums        = feval(mlmc_l,l+Lstart,dNl(l+1),Lstart,...
            replacement);
        Nl(l+1)     = Nl(l+1) + dNl(l+1);
        suml(1,l+1) = suml(1,l+1) + sums(1);
        suml(2,l+1) = suml(2,l+1) + sums(2);
      end
    end

%
% compute absolute average and variance
%
    ml = abs(   suml(1,:)./Nl);
    Vl = max(0, suml(2,:)./Nl - ml.^2);

%
% fix to cope with possible zero values for ml and Vl
% (can happen in some applications when there are few samples)
%
    for l = 3:L+1
      ml(l) = max(ml(l), 0.5*ml(l-1)/2^alpha);
      Vl(l) = max(Vl(l), 0.5*Vl(l-1)/2^beta);
    end

%
% use linear regression to estimate alpha, beta if not given
%
    if alpha_0 <= 0
      A     = repmat((1:L)',1,2).^repmat(1:-1:0,L,1);
      x     = A \ log2(ml(2:end))';
      alpha = max(0.5,-x(1));
    end

    if beta_0 <= 0
      A    = repmat((1:L)',1,2).^repmat(1:-1:0,L,1);
      x    = A \ log2(Vl(2:end))';
      beta = max(0.5,-x(1));
    end
%
% set optimal number of additional samples
%
    Cl  = 2.^(gamma*(0:L))*2^Lstart;
    Ns  = ceil( sqrt(Vl./Cl) * sum(sqrt(Vl.*Cl)) ...
                              / ((1-theta)*eps^2) );
    dNl = max(0, Ns-Nl);
%
% if (almost) converged, estimate remaining error and decide
% whether a new level is required
%
    if sum( dNl > 0.01*Nl ) == 0
      rem = ml(L+1) / (2^alpha - 1);

      if rem > sqrt(theta)*eps
        if (L==Lmax-Lstart)
          fprintf(1,'*** failed to achieve weak convergence *** \n');
        else
          L       = L+1;
          Vl(L+1) = Vl(L) / 2^beta;
          Nl(L+1) = 0;
          suml(1:4,L+1) = 0;

          Cl  = 2.^(gamma*(0:L));
          Ns  = ceil( sqrt(Vl./Cl) * sum(sqrt(Vl.*Cl)) ...
                                    / ((1-theta)*eps^2) );
          dNl = max(0, Ns-Nl);
        end
      end
    end
  end

%
% finally, evaluate multilevel estimator
%
  P = sum(suml(1,:)./Nl);
end
```

portfolio.m

```
function portfolio(nK,nT)
    global S0 r sig K  T dK dT;   % parameter to specify the ...
        financial option
    global nPos;
    global type pos V0 Δ0 gamma0 theta0;
    global horizon drift drift2;
    % model parameters
    S0 = 100;
    r = 0.05;
    sig = 0.2;

    % contract parameters\
    % nL=10;
    % nK=2^(nL/2+1);
    % nT=2^(nL/2);
    nPos=nT*nK;

    dK=1/(nK/2^5);
    K = 100-dK*(nK/2-1):dK:100+dK*nK/2;
    dT=0.02/(nT/2^5);
    T = 1-dT*(nT/2-1):dT:1+dT*nT/2;
    K = repmat(K, 1, nT)';
    T = repelem(T,  nK)';

    % VaR settings
    horizon = 5/252;
    drift =0.1;
    drift2=-0.2;

    V0 =blsprice(S0, K, r, T, sig);
    ∆0 = bls∆(S0, K, r, T, sig);
    gamma0 = blsgamma(S0, K, r, T, sig);
    theta0 = blstheta(S0, K, r, T, sig);

    % positions
    pos=1e+04/nPos./V0;
    type=[ones(nPos,1)];

    value_p = V0'*pos;
    ∆_p = ∆0'*pos;
    gamma_p = gamma0'*pos;
    theta_p = theta0'*pos;
```

A.2 VaR estimation

This is the series of scripts function estimate VaR using the MC and MLMC-ME method. `estimation.m` is used to run independent estimations using the two methods. `VaR_est.m`

```
function VaR_est
close all; clear all;
addpath('..');
rng('default');
global Poly;
global S0 r sig K  T dK dT nT nK;   % parameter to specify the ...
    financial option
global nPos;
global pos V0 ∆0 gamma0 theta0;
global horizon drift;
global p;
global M;
global a b;
global DG;
%    global varMLMC varMC;
%    global varWR varWOR levels j;
%    levels=zeros(20,4);varWR=zeros(20,4);varWOR=zeros(20,4);

%     Poly = @LegendrePArray;
%     Poly = @MonomialsArray;
Poly = @FourierArray;

R=10;
a=-10000;
b=10000;

for DG=1:1
for replacement=0:0
for j=1:1
    nK=2^(3+j);
    nT=2^5;
    portfolio(nK,nT);

    M     = 2;       % refinement cost factor
    N0    = 1000;    % initial samples on coarse levels
    Lmin  = 4;       % minimum refinement level
    Lmax  = round(log(nPos)/log(2));       % maximum refinement level
    Lstart = 3;       %starting level

    N      =5e+04;        % samples for convergence tests
    L      = Lmax;              % levels for convergence tests
    Eps    = 0.01*[ 0.005 0.01 0.02 0.05 0.1 ];

    for p=1:R
        %moments
        fprintf(1,'\n ---- Order-%d moments ---- \n',p);
        filename = ['moments' num2str(p) '_' num2str(replacement)...
            '_' num2str(DG) '_' num2str(nK) '_' num2str(nT)  ...
            '_' num2str(dK) '_'  num2str(dT)];
        fp = fopen([filename '.txt'],'w');

        moments=mlmc_test_VaR(@opre_l, M, N,L, N0,Eps,...
            Lmin,Lmax,Lstart,fp, replacement);
        fclose(fp);

        nvert = 3;
        mlmc_plot(filename, nvert);
        print('-deps2',[filename '.eps'])
    end
end
end
    if DG;
        N1=500000;
        ret = sig*sqrt(horizon)*normrnd(0,1, N1, 1)+ (drift-0.5*sig...
           ^2)*horizon;
        S = S0*exp(ret);
        PnL_DG=(Δ0'*pos)*(S-S0)...
                 +0.5*gamma0'*pos*power(S-S0,2)...
             +theta0'*pos*horizon;
        moments_DG=Poly(PnL_DG,p,a,b);
        moments=moments+mean(moments_DG);
    end;
end

% var_plot(levels(4:end,:), varWR(4:end,:), varWOR(4:end,:))
pr=[0.001 0.005 0.01 0.05 0.1 0.3 0.5];
```

```

%accurate VaR
ret_q=norminv(pr,(drift-0.5*sig^2)*horizon,sig*sqrt(horizon));
S_q=S0*exp(ret_q);
for i=1:length(pr)
    VaR_acc(i)=(sum(blsprice(S_q(i), K, r, T-horizon, sig).*pos)-sum(...
        V0.*pos));
end

%VaR using MC
VaR_MC=empiricalDist(pr);

%VaR using MLMC
[x,w] = GLquad(100*R,a,b);                          % Gaussian quadrature of ...
    sufficiently high order
phi = Poly(x,R,a,b);                                % evaluate polynomials at...
     quadrature points
                                                    % Maximum Entropy method
[lambda,pp,entr,Nstep] = generalizedME(moments',phi,w,0.5,10^-9,1000)...
    ;
                                                    % plot results
t = linspace(a,b,10000)';                            % plot-points at the x-...
    axis
phi = Poly(t,R,a,b);                                % evaluate polynomials at...
     plot-points
density = exp(phi*lambda);                          % evaluate Maximum ...
    Entropy density
hold on;plot(t,density,'-b');                        % plot Maximum Entropy ...
    density
legend('emprical density','smoothed density function', 'maximum ...
    entropy approximation');
for i=1:length(pr)
    ind=find(cumsum(density)*(b-a)/10000>pr(i),1);
    VaR_MLMC(i)=t(ind);
end


%--------------------------------------------------------
%
% level l estimator for Operations Research paper
%
%--------------------------------------------------------

function sums = opre_l(l,N, Lstart,replacement)
rng('default');
    global Poly;
    global S0 r sig K T;    % parameter to specify the financial ...
        option
    global nPos;
    global pos V0 ∆0 gamma0 theta0;
    global horizon drift;
    global p;
    global M;
    global a b;
    global DG;

    nf = M^l;
    nc = nf/M;


    sums(1:6,1:p+1) = 0;

    for N1 = 1:4000:N
        N2 = min(4000,N-N1+1);
        ret = sig*sqrt(horizon)*normrnd(0,1, N2, 1)+ (drift-0.5*sig...
            ^2)*horizon;
        S = S0*exp(ret);
        subf = zeros(N2, nf);
        for i=1:N2
            subf(i,:)=randsample(nPos,nf,replacement);
        end

        V1 = zeros(N2, nf);
        V2 = zeros(N2, nf);
        V = zeros(N2, nf);
        PnL = zeros(N2, nf);
        PnL_DG = zeros(N2, nf);
        PnL_f = zeros(N2, 1);
        PnL_f_DG = zeros(N2, 1);
        PnL_c =zeros(N2, M);
        PnL_c_DG =zeros(N2, M);
        Pf = zeros(N2,p+1);
        Pc = zeros(N2,p+1);

        for i=1:nf
            V(:,i)=blsprice(S, K(subf(:,i)), r, T(subf(:,i))-horizon,...
                 sig);
            PnL(:,i)= (V(:,i)-V0(subf(:,i))).*pos(subf(:,i));
             PnL_DG(:,i)= (∆0(subf(:,i)).*(S-S0)...
                 +0.5*gamma0(subf(:,i)).*power(S-S0,2)...
             +theta0(subf(:,i))*horizon).*pos(subf(:,i));
        end
        PnL_f = sum(PnL, 2)/nf*nPos;
        PnL_f_DG = sum(PnL_DG, 2)/nf*nPos;
        if DG
            Pf = Poly(PnL_f,p,a,b)-Poly(PnL_f_DG,p,a,b);
        else
            Pf = Poly(PnL_f,p,a,b);
        end
        if l>Lstart
            PnL_c = zeros(N2, M);
            for j=1:M
                PnL_c(:,j)=sum(PnL(:,nc*(j-1)+1:nc*j),2)/nc*nPos;
                PnL_c_DG(:,j)=sum(PnL_DG(:,nc*(j-1)+1:nc*j),2)/nc*...
                    nPos;
                if DG
                    Pc = Pc+Poly(PnL_c(:,j),p,a,b)/M-Poly(PnL_c_DG(:,...
                        j),p,a,b)/M;
                else
                    Pc = Pc+Poly(PnL_c(:,j),p,a,b)/M;
                end
            end
        end

         sums(1,:) = sums(1,:) + sum(Pf-Pc);
         sums(2,:) = sums(2,:) + sum((Pf-Pc).^2);
         sums(3,:) = sums(3,:) + sum((Pf-Pc).^3);
         sums(4,:) = sums(4,:) + sum((Pf-Pc).^4);
         sums(5,:) = sums(5,:) + sum(Pf);
         sums(6,:) = sums(6,:) + sum(Pf.^2);

 end
```

mlmc_test_VaR.m

```
%
% function mlmc_test(mlmc_fn,M, N,L, N0,Eps,Lmin,Lmax, fp)
%
% multilevel Monte Carlo test routine
%
% sums = mlmc_fn(l,N)     low-level routine
%
% inputs:  l = level
%          N = number of paths
%
% output: sums(1) = sum(Pf-Pc)
%         sums(2) = sum((Pf-Pc).^2)
%         sums(3) = sum((Pf-Pc).^3)
%         sums(4) = sum((Pf-Pc).^4)
%         sums(5) = sum(Pf)
%         sums(6) = sum(Pf.^2)
%
% M      = refinement cost factor (2^gamma in general MLMC Thm)
%
% N      = number of samples for convergence tests
% L      = number of levels for convergence tests
%
% N0     = initial number of samples for MLMC calcs
% Eps    = desired accuracy array for MLMC calcs
%

function Pa=mlmc_test_VaR(mlmc_fn,M, N,L, N0,Eps,Lmin,Lmax,Lstart,fp,...
    replacement)
global dK dT nK nT;   % parameter to specify the financial option
global nPos;
global p;
global M;
global varMLMC varMC;
global varWR varWOR levels j;
global DG;

%
% first, convergence tests
%
PRINTF2(fp,'\n');
PRINTF2(fp,'...
    **********************************************************\n');
PRINTF2(fp,'*** Convergence tests, kurtosis, telescoping sum check ...
    ***\n');
PRINTF2(fp,'...
    **********************************************************\n');
PRINTF2(fp, 'dK: %f   dT: %f nK: %d  nT: %d replacement: %s DG: %s\n'...
    , ...
    dK, dT, nK, nT,int2str(replacement),int2str(DG));
PRINTF2(fp,'\n l   ave(Pf-Pc)    ave(Pf)   var(Pf-Pc)    var(Pf)');
PRINTF2(fp,'    kurtosis     check \n-------------------------');
PRINTF2(fp,'---------------------------------------------------\n');

% rng('default');    % reset random number generator
P=0;
del1 = [];
del2 = [];
var1 = [];
var2 = [];
kur1 = [];
chk1 = [];
cost = [];
est=[];
Pa=zeros(1,p+1);

for l = Lstart:Lmax
%   disp(sprintf('%d',l))
  tic;
  sums_total = feval(mlmc_fn,l,N, Lstart, replacement);
  cost = [ cost toc ];
  sums_total = sums_total/N;
%   Pa=Pa+sums_total(1,:);
%   sums=sums_total(:,end);
  sums=sums_total(:,2);
  if (l==Lstart)
    kurt = 0.0;
  else
    kurt = (      sums(4)              ...
              - 4*sums(3)*sums(1)      ...
              + 6*sums(2)*sums(1)^2    ...
              - 3*sums(1)*sums(1)^3 )  ...
            / (sums(2)-sums(1)^2)^2;
  end

  del1 = [del1 sums(1)];
  del2 = [del2 sums(5)];
  var1 = [var1 sums(2)-sums(1)^2 ];
  var2 = [var2 sums(6)-sums(5)^2 ];
  var2 = max(var2, 1e-10);  % fix for cases with var=0
  kur1 = [kur1 kurt ];

  if l==Lstart
    check = 0;
  else
    check = abs(       del1(l-Lstart+1)  +      del2(l-Lstart)   - ...
             del2(l-Lstart+1)) ...
      /     ( 3.0*(sqrt(var1(l-Lstart+1)) + sqrt(var2(l-Lstart)) + ...
          sqrt(var2(l-Lstart+1)) )/sqrt(N));
  end
  chk1 = [chk1 check];


  PRINTF2(fp,'%2d   %8.4e  %8.4e  %8.4e  %8.4e  %8.4e  %8.4e \n', ...
          l,del1(l-Lstart+1),del2(l-Lstart+1),var1(l-Lstart+1),var2(l...
            -Lstart+1),kur1(l-Lstart+1),chk1(l-Lstart+1));
    P=P+del1(l-Lstart+1);
end


%
% print out a warning if kurtosis or consistency check looks bad
%

if ( kur1(end) > 100.0 )
  PRINTF2(fp,'\n WARNING: kurtosis on finest level = %f \n',kur1(end)...
      );
  PRINTF2(fp,' indicates MLMC correction dominated by a few rare ...
      paths; \n');
  PRINTF2(fp,' for information on the connection to variance of ...
      sample variances,\n');
  PRINTF2(fp,' see http://mathworld.wolfram.com/...
      SampleVarianceDistribution.html\n\n');
end

if ( max(chk1) > 1.0 )
  PRINTF2(fp,'\n WARNING: maximum consistency error = %f \n',max(chk1...
      ));
  PRINTF2(fp,' indicates identity E[Pf-Pc] = E[Pf] - E[Pc] not ...
      satisfied \n\n');
end

%
% use linear regression to estimate alpha, beta and gamma
%

L1 = 2;
L2 = L-Lstart+1;
% L2 = L;
pa    = polyfit(L1:L2,log2(abs(del1(L1:L2))),1);  alpha = -pa(1);
pb    = polyfit(L1:L2,log2(abs(var1(L1:L2))),1);  beta  = -pb(1);
if replacement
    varWR(1:(L2-L1+1),j)=var1(L1:L2);
else
    varWOR(1:(L2-L1+1),j)=var1(L1:L2);
end
levels(1:(L2-L1+1),j)=L1:L2;
% just last two points to minimise effect of MATLAB overhead
gamma = log2(cost(end)/cost(end-1));

PRINTF2(fp,'\n*******************************************************\...
    n');
PRINTF2(fp,'*** Linear regression estimates of MLMC parameters ***\n'...
    );
PRINTF2(fp,'*******************************************************\n'...
    );
PRINTF2(fp,'\n alpha = %f  (exponent for MLMC weak convergence)\n',...
    alpha);
PRINTF2(fp,' beta  = %f  (exponent for MLMC variance) \n',beta);
PRINTF2(fp,' gamma = %f  (exponent for MLMC cost) \n',gamma);

%
% second, mlmc complexity tests
%
```

```

PRINTF2(fp,'\n');
PRINTF2(fp,'**************************** \n');
PRINTF2(fp,'*** MLMC complexity tests *** \n');
PRINTF2(fp,'**************************** \n\n');
PRINTF2(fp,'   eps       value    mlmc_cost    std_cost  savings     ...
    N_l \n');
PRINTF2(fp,'...
    ---------------------------------------------------------- \n');

% rng('default');    % reset random number generator

alpha = max(alpha,0.5);
beta  = max(beta,0.5);
gamma = log2(M);
theta = 0.25;

for i = 1:length(Eps)
  eps = Eps(i);
  [P, Nl] = mlmc_together(Lmin,Lmax,Lstart,N0,eps,mlmc_fn,alpha,beta,...
      gamma,replacement);
  if i==1; Pa=P; end;
  l = length(Nl)-1;

  mlmc_cost = sum(Nl.*M.^(0:l))*M^Lstart;
  l2 = min(l+1,length(var2));
  std_cost  = var2(l2) / (eps^2)*nPos;

  PRINTF2(fp,'%.3e  %.3e  %.3e  %.3e  %7.2f ', ...
      eps, P(end), mlmc_cost, std_cost, std_cost/mlmc_cost);
  PRINTF2(fp,'%9d',Nl);
  PRINTF2(fp,'\n');
end

PRINTF2(fp,'\n');

end

%
% function to print to both a file and stdout
%

function PRINTF2(fp,varargin)
  fprintf(fp,varargin{:});
  fprintf( 1,varargin{:});
end
```

mlmc_VaR.m

```
%% function [P, Nl] = mlmc(Lmin,Lmax,N0,eps,mlmc_l, alpha,beta,gamma)
%
% multi-level Monte Carlo estimation
%
% P     = value
% Nl    = number of samples at each level
%
```

```
% Lmin  = minimum level of refinement        ≥ 2
% Lmax  = maximum level of refinement        ≥ Lmin
% N0    = initial number of samples          > 0
% eps   = desired accuracy (rms error)       > 0
%
% alpha -> weak error is  O(2^{-alpha*l})
% beta  -> variance is    O(2^{-beta*l})
% gamma -> sample cost is O(2^{gamma*l})     > 0
%
% if alpha, beta are not positive then they will be estimated
%
% mlmc_l = function for level l estimator
%
% sums = mlmc_fn(l,N)     low-level routine
%
% inputs:  l = level
%          N = number of paths
%
% output: sums(1) = sum(Y)
%         sums(2) = sum(Y.^2)
%         where Y are iid samples with expected value:
%         E[P_0]           on level 0
%         E[P_l - P_{l-1}] on level l>0

function [P, Nl] = mlmc_VaR(Lmin,Lmax,Lstart, N0,eps,mlmc_l, alpha_0,...
    beta_0,gamma,replacement)
    global p;
%
% check input parameters
%
  if (Lmin<2)
    error('error: needs Lmin ≥ 2');
  end

  if (Lmax<Lmin)
    error('error: needs Lmax ≥ Lmin');
  end

  if (Lmin<Lstart)
      error('error: needs Lstart ≥ Lmin');
  end

  if (N0≤0 || eps≤0 || gamma ≤ 0)
    error('error: needs N>0, eps>0, gamma>0 \n');
  end

%
% initialisation
%
  alpha = max(0, alpha_0);
  beta  = max(0, beta_0);

  theta = 0.25;

  L = Lmin-Lstart;

  Nl(1:L+1)        = 0;
  suml(1:2,1:L+1)  = 0;
  dNl(1:L+1)      = N0;
  mo=zeros(Lmax-Lstart+1,p+1) ;

  while sum(dNl) > 0

%
% update sample sums
%
    for l=0:L
      if dNl(l+1) > 0
        sums        = feval(mlmc_l,l+Lstart,dNl(l+1),Lstart,...
           replacement);
        Nl(l+1)     = Nl(l+1) + dNl(l+1);
%         suml(1,l+1) = suml(1,l+1) + sums(1,end);
%         suml(2,l+1) = suml(2,l+1) + sums(2,end);
        suml(1,l+1) = suml(1,l+1) + sums(1,2);
        suml(2,l+1) = suml(2,l+1) + sums(2,2);
        mo(l+1,:)=mo(l+1,:)+sums(1,:);
      end
    end

%
% compute absolute average and variance
%
    ml = abs(   suml(1,:)./Nl);
    Vl = max(0, suml(2,:)./Nl - ml.^2);

%
% fix to cope with possible zero values for ml and Vl
% (can happen in some applications when there are few samples)
%
    for l = 3:L+1
      ml(l) = max(ml(l), 0.5*ml(l-1)/2^alpha);
      Vl(l) = max(Vl(l), 0.5*Vl(l-1)/2^beta);
    end

%
% use linear regression to estimate alpha, beta if not given
%
    if alpha_0 ≤ 0
      A     = repmat((1:L)',1,2).^repmat(1:-1:0,L,1);
      x     = A \ log2(ml(2:end))';
      alpha = max(0.5,-x(1));
    end

    if beta_0 ≤ 0
      A    = repmat((1:L)',1,2).^repmat(1:-1:0,L,1);
      x    = A \ log2(Vl(2:end))';
      beta = max(0.5,-x(1));
    end
%
% set optimal number of additional samples
%
    Cl  = 2.^(gamma*(0:L))*2^Lstart;
    Ns  = ceil( sqrt(Vl./Cl) * sum(sqrt(Vl.*Cl)) ...
                         / ((1-theta)*eps^2) );
    dNl = max(0, Ns-Nl);
%
```

```
% if (almost) converged, estimate remaining error and decide
% whether a new level is required
%
    if sum( dNl > 0.01*Nl ) == 0
      rem = ml(L+1) / (2^alpha - 1);

      if rem > sqrt(theta)*eps
        if (L==Lmax-Lstart)
          fprintf(1,'*** failed to achieve weak convergence *** \n');
        else
          L        = L+1;
          Vl(L+1)  = Vl(L) / 2^beta;
          Nl(L+1)  = 0;
          suml(1:4,L+1) = 0;

          Cl   = 2.^(gamma*(0:L));
          Ns   = ceil( sqrt(Vl./Cl) * sum(sqrt(Vl.*Cl)) ...
                                  / ((1-theta)*eps^2) );
          dNl = max(0, Ns-Nl);
        end
      end
    end
  end

%
% finally, evaluate multilevel estimator
%
%   P = sum(suml(1,:)./Nl);
    for l= 1:length(Nl)
     mo(l,:)=mo(l,:)/Nl(l);
    end
    P=sum(mo);
end
```

empiricalDist.m

```
function VaR_MC=empiricalDist(pr)
%     rng('default');
    global Poly;
    global S0 r sig K  T dK dT nT nK;   % parameter to specify the ...
        financial option
    global nPos;
    global type pos V0 ∆0 gamma0 theta0;
    global horizon drift;
    global p;
    global M;
    global a b;
    global DG;%empirical distribution

    nsample=840;

    PnL=zeros(nsample,1);
    ret = sig*sqrt(horizon)*normrnd(0,1, nsample, 1)+ (drift-0.5*sig...
        ^2)*horizon;
    S = S0*exp(ret);
    for i=1:nsample
```

```
        V=blsprice(S(i), K, r, T-horizon, sig);
        PnL(i,1)=sum(V.*pos)-sum(V0.*pos);
    end
    PnL_sorted=sort(PnL);
    ind=nsample*pr;
    ind_ceil=ceil(ind);
    ind_floor=max(floor(ind),ones(1, length(pr)));

    VaR_MC=0.5*(PnL_sorted(ind_floor)+PnL_sorted(ind_ceil));
    nx = 1000;
    xinput=linspace(min(PnL), max(PnL), nx);
    freq = hist(PnL,nx);
    prob = freq/nsample/(max(PnL)- min(PnL))*nx;
    % empirical = bar(xinput, prob-1)
    % uistack(empirical,'bottom')
    figure()
    grey = [0.6 0.6 0.6];
    empirical = bar(xinput, prob, 'FaceColor',grey);
    uistack(empirical,'bottom')
    VaR_MC=VaR_MC';

    hold on;
    [f,xi] = ksdensity(PnL);
    plot(xi,f, '-k', 'LineWidth',1.5);
```

estimation.m

```
VaR_acc=[];
VaR_MC=[];
VaR_MLMC=[];
for i=1:1000
    try
        [a,b,c]=testMLMC_new_all;
            VaR_acc=a;
        VaR_MC = [VaR_MC;b];
        VaR_MLMC = [VaR_MLMC;c];
        CATCH
    end
end
% pr=[0.001 0.005 0.01 0.05 0.1 0.3 0.5];
```

A.3 Plotting scripts

The scripts to generate the figures in Chapter 5 are listed below.

- var_plot.m is the function to generate figure 5.1.
- error_plot.m is the function to generate figure 5.4 and 5.5.

var_plot.m

```
load var.mat;
set(0,'DefaultAxesColorOrder',[0 0 0]);
set(0,'DefaultAxesLineStyleOrder','--o|--x|--d|--*|--s');
levels=levels(:,1:3);
varWR=varWR(:,1:3);
varWOR=varWOR(:,1:3);
varWR_DG=varWR_DG(:,1:3);
varWOR_DG=varWOR_DG(:,1:3);
for i=1:4
    subplot(2,2,i);
if i==1
    plot(levels, log2(varWR));
    title({'Replacement: true','DGT: false'});
elseif i==2
    plot(levels, log2(varWOR));
    title({'Replacement: false','DGT: false'});
elseif i==3
    plot(levels, log2(varWR_DG));
    title({'Replacement: true','DGT: true'});
else
    plot(levels, log2(varWOR_DG));
    title({'Replacement: false','DGT: true'});
end


for j=1:size(levels,2)
    labels{j} = strcat('K=',num2str(power(2,max(levels(:,j)))));
end
legend(labels,'Location','NorthEast')
ylabel('log_2 variance')
ylabel('variance')
xlabel('level l');
end
print('-deps2',['var.eps'])
```

error_plot.m

```
pr=[0.001 0.005 0.01 0.05 0.1 0.3 0.5];
error=0.01*[4 2 1 0.5 0.25 0.125 0.0625];
S(1)=load('4e-2.mat');
S(2)=load('2e-2.mat');
S(3)=load('1e-2.mat');
S(4)=load('5e-3_2.mat');
S(5) = load('25e-3.mat');
S(6) = load('125e-3_2.mat');
S(7) = load('625e-4.mat');
RMSE_MLMC=zeros(7,7);
RMSE_MC=zeros(7,7);
bias_MLMC=zeros(7,7);
bias_MC=zeros(7,7);

for i=1:7
    [n,m]=size(S(i).VaR_MLMC);
        RMSE_MLMC(i,:)=sqrt(mean((S(i).VaR_MLMC-repmat(S(i).VaR_acc,n...
            ,1)).^2));
        RMSE_MC(i,:)=sqrt(var(S(i).VaR_MC-repmat(S(i).VaR_acc,n,1)));
        bias_MLMC(i,:)=mean(S(i).VaR_MLMC-repmat(S(i).VaR_acc,n,1));
        bias_MC(i,:)=mean(S(i).VaR_MC-repmat(S(i).VaR_acc,n,1));
end
loglog(error(1:4), RMSE_MLMC(1:4,3),'--o', 'LineWidth',1.5);hold on;
loglog(error(1:4), RMSE_MLMC(1:4,4),'--x', 'LineWidth',1.5);hold on;
loglog(error(1:4), RMSE_MLMC(1:4,5),'--d', 'LineWidth',1.5);hold on;
loglog(error(1:4), RMSE_MLMC(1:4,6),'--*', 'LineWidth',1.5);
for i=1:4
    labels{i} = strcat(num2str(pr(i+2)),'-VaR');
end
legend(labels,'Location','SouthEast')
ylabel('RMSE of VaR')
xlabel('RMSE of 1st order moment');
print  -depsc RMSE_colored.eps


E(1)=load('512.mat');
E(2)=load('1024.mat');
E(3)=load('2048.mat');

K=2^9*[1 2 4];
for i=1:3
     [n,m]=size(E(i).VaR_MLMC);
     RMSE_MLMC2(i,:)=sqrt(mean((E(i).VaR_MLMC-repmat(E(i).VaR_acc,n...
        ,1)).^2));
     RMSE_MC2(i,:)=sqrt(mean((E(i).VaR_MC-repmat(E(i).VaR_acc,n,1))....
        ^2));
end


for i=1:4
    subplot(2,2,i);
    plot(K,RMSE_MC2(:,i+2),'--o',K,RMSE_MLMC2(:,i+2),'--x','LineWidth...
       ',1.5);
    legend('MC', 'MLMC','Location','northwest');
    title(strcat(num2str(pr(2+i)),'-VaR'));
    xlabel('K');
    ylabel('RMSE');
    a=round(max(RMSE_MC2(:,i+2))/100)+2;
    ylim([0 500]);
end
print  -depsc RMSE_K_colored.eps
```